View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by DSpace at VU

ETAG

a Formal Model of Competence Knowledge for User Interface Design

Geert de Haan

SIKS Dissertation Series No. 2000-X.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

Promotiecommissie:

prof. dr. J.C. van Vliet (promotor)
dr. G.C. van der Veer (co-promotor)
dr. M.J. Tauber (co-promotor - University of Paderborn)
dr. T.R.G. Green (University of Leeds)
prof. dr. G.W.M. Rauterberg (Eindhoven University of Technology)
dr. ir. C.A.P.G. van der Mast (Delft University of Technology)
prof. dr. F.M.T. Brazier

Copyright © 2000 by Geert de Haan.
Permission is given to distribute this material or part of it for not-for-profit purposes. Distribution of substantively modified versions of this document, derivatives of the work, or distribution for profit purposes is prohibited without prior permission.

Vrije Universiteit

ETAG, A Formal Model Of Competence Knowledge For User Interface Design

Academisch Proefschrift

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam
op gezag van de rector magnificus
prof. dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
wiskunde en informatica
van de faculteit der exacte wetenschappen
op XXXX-dag MMMMM 2000 om UU.MM uur
in het hoofdgebouw van de universiteit,
De Boelelaan 1105

door

Geert de Haan

geboren te Eindhoven

Promotor: prof. dr. J.C. van Vliet

Acknowledgements

I know at last what I want to be when I grow up.
When I grow up I want to be a little boy.
Heller; J. (1989).

If it takes a village to raise a child, raising a doctor takes such a truly remarkable number of people, which, unfortunately, this page is too small to contain. Let those whom I failed to mention here be assured that my gratitude extends beyond this one page. I thank my family, my brothers, my mother and my grandmother for providing the foundation on which this work is built.

I thank my past and present friends, play-friends, study-friends, running-friends, house-mates, lovers and regular friends for making life bearable, interesting, wonderful and often much more. In particular, I wish to thank Joris Huijser, Frank Wassenaar, and Sue Bowen for their friendship, and Ab de Haan for being a very strange brother indeed.

I thank all my colleagues and friends who contributed to my person and work when I was at MRC's Applied Psychology Unit in Cambridge, the unit of Experimental Psychology of Leiden University, the unit of Ergonomics of the University of Twente, the division of Mathematics and Computer Science of the Vrije Universiteit Amsterdam, Origin's Distributed Systems department in Eindhoven, and IPO, Center for User-System Interaction of the University of Technology Eindhoven. I am particularly indebted to all the people at IPO for the motivating and friendly atmosphere.

I thank a number of people for influencing my scientific thinking in a positive way, even though, possibly, unwittingly: Piet Vroon inspired me to psychology as a science; Lex van der Heijden taught me experimental psychology; Harke, Wim and Gert let me play with PDP's (8, 10 and 11) which drew my attention to Human-Computer Interaction (HCI); Ino Flores D'Arcais arranged that I could stay with Thomas at the APU; Patrick Hudson taught me to write-down my ideas *before* rejecting them; Thos Green, my flute-playing barn-dancing master who taught me to think HCI; Ted White taught me to be practical and established the contact with Gerrit; Gerrit van der Veer supervised and inspired this thesis with huge amounts of energy, absent-minded busy-bodiedness and yet-another-idea-ideas; Elly Lammers enabled Gerrit; Michael Tauber provided ETAG when I was looking for a TAG^{++}; John Long's critical mind inspired me to create ETAG-based design; Hans van Vliet led the NFI-project which made this all possible, patiently kept the thesis on (a straight) track, and provided numerous improvements to its contents and language; finally, I am very grateful to Matthias Rauterberg for arranging the means and putting me on the spot to finish the work.

I thank all participants of the NWO/NFI Project "Systematic Design of User Interfaces" for our fruitful co-operation, and particularly my fellow OIO's for what I have learned from them and the students who worked on ETAG for their contributions to this thesis. I thank the Netherlands Organization for Scientific Research (NWO) for making this project (NF 82/62-312) financially possible.

Last but not least, I thank my bicycles and running shoes for helping me keeping up a sane body, my tents and books for keeping up a sane mind, and all the beautiful people who struggle to make this world a better place for all of us, for keeping up hope.

Chapter 1:
Cognitive Ergonomics and User Interface Design 1
Abstract 1
1.1 Cognitive Ergonomics 1
1.2 Usable Computer Systems 4
1.3 Essentials of Cognitive Ergonomics 5
1.3.1 Cognitive Ergonomics is Science and Engineering 6
1.3.2 Cognitive Ergonomics is about User Interfaces 9
1.3.3 Cognitive Ergonomics is about Design 13
1.3.4 The Main Research Questions and the Structure of the Thesis 16

Chapter 2:
Formal Modelling Techniques in Human Computer Interaction 21
Abstract 21
2.1 Introduction 21
2.2 Models and Levels Of Abstraction in HCI 23
2.3 Methods to Enhance the Usability of Computer Systems 26
2.4 An Overview of Formal Modelling Techniques 28
2.4.1 Models for Task Environment Analysis 29
2.4.1.1 ETIT *29*
2.4.2 Models to Analyse User Knowledge 30
2.4.2.1 Action Language *31*
2.4.2.2 TAG *32*
2.4.3 Models of User Performance 33
2.4.3.1 GOMS *34*
2.4.3.2 CCT *37*
2.4.4 Models of the User Interface 38
2.4.4.1 CLG *39*
2.4.4.2 ETAG *41*
2.5 An Evaluation of Formal Modelling Techniques 43
2.5.1 Completeness 45
2.5.2 Width of Applicability 46
2.5.3 Validity 46
2.5.4 Usability 47
2.6 Conclusions 49

Chapter 3:
The Psychological Basis of Extended Task-Action Grammar 53
Abstract 53
3.1 Introduction 53
3.2 Extended Task-Action Grammar 54
3.2.1 What ETAG represents about Users 55
3.2.2 What ETAG represents about User Interfaces 56
3.2.3 Related Approaches to User Interface Representation 58
3.2.4 The Structure of ETAG Representations 59
3.3 ETAG as an Eclectic Choice of Theories - Structure Choices 59
3.3.1 A Feature Grammar 61

3.3.2 Levels of Rewrite Rules 63
3.3.3 Knowledge Representation 64
3.4 A Canonical Basis for System Knowledge - Content Choices 66
3.4.1 Semantic structures 67
3.4.2 Klix' contribution 68
3.5 Modelling Reasoning 71
3.5.1 Ontological Categories for Knowledge and Reasoning 74
3.5.2 The Structure of Knowledge 75
3.5.3 Human Reasoning 76
3.6 The structure of ETAG in terms of psychological considerations 77
3.7 Conclusions 80

Chapter 4:
An ETAG-based Approach to the Design of User-interfaces 83
Abstract 83
4.1 Introduction 83
4.2 The Received View on User Interface Design in HCI 87
4.2.1 Task and Context Analysis 88
4.2.2 Global Design 88
4.2.3 Detail Design 89
4.2.4 Implementation and Testing 89
4.2.5 Installation and Evaluation 90
4.2.6 Discussion 90
4.3 Users, Tasks and Task Concepts. 93
4.3.1 Views on the Concept of the User Interface 94
4.3.2 ETAG's View on the User Interface in Design Practice 96
4.4 ETAG-Based Design 97
4.4.1 Task and Context Analysis 98
4.4.2 Task Design 101
4.4.3 Conceptual Design 102
4.4.4 Perceptual Design 104
4.4.5 Evaluation and Implementation, Installation and Use 106
4.5 Discussion 107
4.5.1 Related Approaches 107
4.5.2 Conclusions 109

Chapter 5:
Task Analysis for User Interface Design 111
Abstract 111
5.1 Introduction 111
5.2 What is Task Analysis? 113
5.2.1 Task analysis as Activity or Result 113
5.2.2 The Depth of Task Analysis 114
5.2.3 The Detailedness of Task Analysis 115
5.2.4 The Scope of Task Analysis 116
5.2.5 The Contents of Task Analysis 116
5.3 Missing Issues in Early Approaches to Task Analysis 118

5.3.1 Physical Actions ... **118**
5.3.2 Objects and Tools ... **119**
5.3.3 Cognitive Task Aspects ... **119**
5.3.4 Social and Organisational Aspects ... **119**
5.3.5 Task Organisation ... **120**
5.4 ETAG and Task Analysis ... **120**
5.4.1 Formality ... **121**
5.4.2 Grammaticality ... **121**
5.4.3 Object-orientedness ... **122**
5.4.4 Conclusions about ETAG and Task Analysis ... **123**
5.5 An Eclectic Approach to Task Analysis ... **124**
5.5.1 Context ... **124**
5.5.2 Tasks and Actions ... **125**
5.5.3 Purposes ... **126**
5.5.4 Things ... **127**
5.5.5 People ... **127**
5.5.6 Three Steps in Task Analysis ... **128**
5.6 An ETAG Task Analysis Example ... **129**
5.6.1 The Problem ... **129**
5.6.2 Establish a Background ... **130**
5.6.3 Delineate the Problem ... **131**
5.6.4 Describe the Work Situation ... **136**
5.6.5 An ETAG Task Analysis Specification ... **141**
5.6.5.1 Object Specification ... *141*
5.6.5.2 Task Specification ... *144*
5.6.5.3 Conclusions ... *149*
5.7 Conclusions ... **151**

Chapter 6:
How to Create an ETAG Representation ... **155**
Abstract ... **155**
6.1 Introduction ... **155**
6.2 How to Cook an ETAG Representation ... **156**
6.2.1 Step 1: A Raw List of Tasks, Objects, Attributes, etc. ... **158**
6.2.2 Step 2: Select Concepts for the Canonical Basis ... **160**
6.2.3 Step 3: The Preliminary List of Tasks and Basic Tasks ... **161**
6.2.4 Step 4: A First Specification of the Elements of the UVM ... **163**
6.2.5 Step 5: Refine the Specification of the UVM until Completion ... **170**
6.2.6 Step 6: The Specification of the Perceptual Interface ... **171**
6.3 ETAG-based User Interface Design ... **175**
6.4 Conclusions ... **176**

Chapter 7:
Analyzing User Interfaces: ETAG Validation Studies ... **179**
Abstract ... **179**
7.1 Introduction ... **179**
7.2 Studies in Analysing User Interfaces ... **181**

7.2.1 Analysis of Two Page Tools ... **182**
7.2.2 Variants of a Spreadsheet for Currency Exchange ... **183**
7.2.2.1 The User's Virtual Machine ... *185*
7.2.2.2 The Basic Tasks ... *189*
7.2.2.3 The Event Type Specification ... *190*
7.2.2.4 The Production Rules ... *193*
7.2.2.5 General Comments on Modelling the Currency System ... *195*
7.2.3 Analysis of Two Electronic Mail Systems ... **198**
7.3 An Evaluation of ETAG as a Tool for Analysis ... **200**
7.3.1 What Has Been Learned From These Studies ... **200**
7.3.2 Conclusions ... **201**

Chapter 8:
Etag as the Basis for Intelligent Help Systems ... **203**
Abstract ... **203**
8.1 Introduction ... **203**
8.2 Etag as the Basis for Help Systems ... **204**
8.2.1 The General Structure of an ETAG Help System ... **204**
8.3 Examples of ETAG-based Help Systems ... **206**
8.3.1 Help Systems to Provide On-line Manuals ... **206**
8.3.2 Help Systems to Provide Static Information ... **207**
8.3.3 A Help Systems for Dynamic Information using an ETAG Simulator ... **210**
8.4 Summary and Future Outlook about ETAG-Based Help Systems ... **214**
8.5 Conclusions ... **216**

Chapter 9:
Conclusions ... **217**
9.1 Summary of Findings ... **217**
9.1.1 Assessment Criteria for Design Representation Models ... **218**
9.1.2 Models for User Interface Design Representation ... **218**
9.1.3 A Most Suitable Model for Design Representation ... **219**
9.1.4 Using ETAG to Answer Design Questions ... **221**
9.1.5 The Received View as a Method for User Interface Design ... **222**
9.1.6 A Defining View of the User Interface as a Concept ... **223**
9.1.7 The Structure of ETAG-based User Interface Design ... **223**
9.1.8 Using ETAG throughout User Interface Design ... **225**
9.2 Discussion and Future Research ... **226**
9.2.1 ETAG-based Design and the Presentation Interface ... **227**
9.2.2 ETAG-based Design and different Types of User Interfaces ... **230**

References ... **235**
Citations ... **249**
Dutch Summary ... **250**
SIKS Dissertations ... **251**
Biography ... **253**

Chapter 1:

Cognitive Ergonomics and User Interface Design

Because psychology lacks a generally accepted theoretical uniform that fits all figures and pleases all tastes, it is especially prone to changing fashion. Psychologists do not even agree about what the basic items their science's wardrobe should contain - what the right questions are to ask. [...] Given this disagreement over what style of theorizing best suits the mind, any new approach is likely to be hailed as the missing paradigm, the link transporting psychology from myth to science.

Boden, M. (1989).

Abstract

This book is about a particular solution to the problem of how to design usable computer systems. Apart from this chapter, it may be read as a short history of the research efforts with respect to ETAG, which is the main subject of the book as well as the tool that is proposed to help create usable computer systems. Since answers are meaningless without the questions they address, this chapter discusses the questions underlying this research and, perhaps even more important, the reasons for asking these particular questions: the theoretical context which gave rise to them. This is especially important since a generally accepted approach to study and design in Human-Computer Interaction (HCI) is lacking. Moreover, the field might rather be characterised as a loosely organised collection of competing and often incompatible approaches.

First, the chapter provides a general account of the subject matter of cognitive ergonomics, stressing the 'cognitive' part of it, and distinguishing it from related fields of research. Thereafter, the chapter focuses on creating usable computer systems as the primary aim of cognitive ergonomics, and it explains why this is not an easy task. Finally, the chapter defines cognitive ergonomics more precisely, as a **science and engineering** trade, that is concerned with **user interfaces**, and more specifically, their **design**. Guided by these three main characteristics of cognitive ergonomics, the major questions put forward in this thesis are derived and their background is discussed.

1.1 Cognitive Ergonomics

Cognitive ergonomics is the study of human behaviour that is mediated by cognitive tools and devices. Cognitive tools are natural or artificial tools which require and determine the human ability to process information. The purpose of cognitive ergonomics is to adapt such cognitive tools and their usage so as to improve human information processing in terms of improved efficiency, fewer errors and accidents, and increased well-being. To this end it is necessary that cognitive ergonomics is able to analyse how such tools are used, to synthesise recommendations for design, and to evaluate uses and recommendations.

An important area in cognitive ergonomics is the study of Human-Computer Interaction. Cognitive ergonomics is based on the premise that human interaction with computer devices is essentially a matter of knowledge representation and information processing, or: cognitive behaviour.

Interacting with a computer system takes place by means of physical interaction: pressing buttons on a keyboard changes the physical state of the computer system which is fed back to

the user by means of intensity changes of the light on the display unit. This description may be accurate, but it is as irrelevant as describing driving a car in terms of opening valves and pulling cables. It would rather be more sensible to describe interacting with a computer system in terms of writing a book with a word processor or calculating turnovers using a spreadsheet.
To acquire 'common' human goals like writing a thesis or calculating the day's turnovers by means of a computer (or, for that matter any other tool, like a typewriter, pencil and paper, etc.), it is necessary to recursively subdivide and translate goals into commands for the word processor or spreadsheet. Since computer systems are not too sophisticated with respect to supporting human goals, the reformulation and translation processes of human goals and the interpretation of results of command invocations are necessarily human cognitive tasks. In cognitive ergonomics, interacting with a computer is assumed to involve different *stages* of human behaviour, such as formulating an intention and executing an action, and different levels of activity, such as the intention to improve a text and the intention behind physically executing a particular command (Norman, 1984).

In cognitive ergonomic terms, users have to apply their knowledge about the computer system, in the form of a mental representation, to find the difference between a current state of affairs and a goal state within the user's task domain. Once established, the user has to devise a plan to diminish the difference, and reformulate the plan into the commands and command arguments of the computer system used. After issuing the commands, again, the user has to apply the knowledge about the system to transform feedback data from the computer into meaningful information about the success or failure of reaching the goal state within the task domain.
As an example, consider making a paper copy of a report. In a paper office, one would acquire the report from its file, bring it to the copy machine to make a copy, and try not to forget to store the archive copy back in its place. In a 'paperless' office, it is necessary to know that reports are stored as computer files and that, in order to make a paper copy of a file, it has to be sent or copied to a printer device. The instruction how to copy a file varies between computer systems: dragging an icon representation of the report to a printer icon, typing in the name of the print command and the filename, etc. When the printer is not within visual or auditory reach, additional command specifications may be necessary to acquire information about the progress of the print command.

Since computer usage primarily involves acquiring, transforming and applying (human) knowledge it may be clear that the basic thesis or central premise of cognitive ergonomics states that investigations should focus on cognitive factors in order to improve Human-Computer Interaction.

The focus on cognitive factors distinguishes cognitive ergonomics from traditional or 'classical' *ergonomics*. As the name suggests, cognitive ergonomics may be seen as a mere branch of ergonomics, with which it shares the goal of facilitating human performance through adaptation of the tools to human characteristics and preferences. On the other hand, cognitive ergonomics differs from 'classical' ergonomics in that the focus is not so much on externally measurable quantities, such as movements, forces and body measures, but on

psychological phenomena, such as knowledge, perception, and planning; phenomena that, generally, allow for indirect measurement only.

Cognitive ergonomics is also closely related to *cognitive psychology*, in that both investigate a mental phenomena. Whereas cognitive psychology often uses computers to study human mental phenomena for the sake of acquiring general theories about mental behaviour, cognitive ergonomics studies mental phenomena and applies theoretical knowledge in order to solve the practical problems related to using computers. The abstract nature of cognitive psychological knowledge generally precludes applying it to practical problems without first making additional assumptions that may undermine the validity of solutions. Cognitive ergonomic theories are more directly connected to the domain of application.

Finally, cognitive ergonomics is related to *computer science* in that both study the use of computer systems, but, whereas computer science investigates the technical requirements for using computers, cognitive ergonomics studies the human and cognitive requirements for doing so. Whereas cognitive ergonomics is related to general ergonomics and cognitive psychology via the subject matter of the investigations, computer science and cognitive ergonomics may be seen as mutual clients, where cognitive ergonomics is responsible for the design of the user interface (the user machine) that sets cognitive constraints, and computer science is responsible for the design and creation of the software (the soft machine) which sets and implements technical constraints for the overall design of computer systems.

The fact that computer science and cognitive ergonomics are clients of each other does not express anything about their relative importance. Although there is a growing awareness that computer system design should include usability aspects, at present, especially in software engineering practice, the technical constraints are still predominant.
Cognitive ergonomics and ergonomics in general are still regarded as additional to technical programming skills, rather than the opposite. Later on, it will be argued that, from a human task performance point of view, a far more important, and perhaps a leading role, should be assigned to cognitive ergonomics.

At the start of this section, cognitive ergonomics was defined as the study to improve the use of cognitive tools in terms of efficiency, errors and accidents, and well-being. It may be possible to be more precise about the field and purpose of cognitive ergonomics. For example, humanitarians might insist that well-being refers to all human beings involved and exclude weapons, technocrats might want to focus on work systems and exclude enabling tools, and rationalists might want to restrict attention to purposive tools and exclude toys and pleasure. These statements may be caricatures but they do exemplify the risk of losing generality by playing the language game to strictly.
Despite that cognitive ergonomics is not served by a priori excluding application areas and a more precise definition will not be provided, the work that is discussed in this thesis is restricted to the purposive usage of tools by means of discrete task performance. Computer systems are most commonly used to support work systems which makes it an obvious choice in HCI to focus on users who perform tasks for the purpose of acquiring specific goals and, because of the way to issue commands to computer systems, to focus on performing discrete tasks.

Discrete tasks put specific requirements on cognitive processes such as perception, memory and attention which may differ considerably from the requirements of the tasks in different areas of cognitive ergonomics such as steering vehicles or controlling industrial processes. As a consequence, tasks have a rather universal status within HCI but in other areas of cognitive ergonomics aspects like the structure and presentation of information, and learning and skilled task performance may be more important.

The remainder of this chapter discusses the design of usable computer systems as one of the main subjects of cognitive ergonomics and what should be done in order to develop cognitive ergonomics into the science and the engineering practice of user interface design. Arguing that user interfaces should primarily be seen as the user's means to perform tasks, the user interface is defined as the knowledge that users need to successfully perform tasks with a computer system (Moran, 1981; van der Veer, 1990). In combination with the argument that cognitive ergonomics should focus on the development of theories to capture its scientific knowledge and methods to use this knowledge to solve practical design problems, the two main challenges of this thesis are formulated as:

- the selection and development of a good representation for user interfaces
- the development of a user interface design method based on this representation

1.2 Usable Computer Systems

Designing usable computer systems is a major concern for cognitive ergonomics. In this paragraph it will be argued that regardless of the general difficulties to design and to design computer systems, designing usable computer systems is a more difficult task due to the combination of the technology which, at least, in principle does not impose restrictions on the design and the psychology of the human task performance which is largely unknown and hidden from direct inspection.

Being concerned with the design of usable artefacts is not specific to cognitive ergonomics. For example, for centuries people have been designing usable bridges, at first, using trial and error, and gradually shifting to genuine design methods that carefully consider all known relevant variables (e.g. Petrosky, 1996). What is new in HCI design is that computers are general purpose information processors and able to support human task performance in many different ways, which may not be obvious, or even compatible with each other. As such, the number of variables to consider is large, and when their interactions are included, the number becomes huge, and when considering that there are no such 'obvious' criteria for evaluating HCI designs as there are for designing bridges, the number becomes endless.
In mathematical terms, for each problem that can be solved computationally there is an infinite number of possible solutions and, generally, a large number of feasible ones. In terms of computational support for human tasks, concern is not with solving single, isolated problems, but with interconnected problems. As such, when a software solution is found to solve a particular task performance problem, it may create new problems, and make yet other problems easier or harder to solve. In addition, when working on a problem, it may remain unknown for some time how particular solutions will influence solving other problems.

For example, an undo facility is a means to solve the problem that users may perform unintended actions, but this solution may create synchronisation problems when data is shared among users, and it may create the problem of what to do with accidental use of the undo facility. In this respect, design is largely a matter of solving trade-offs between requirements, between solutions and between requirements and solutions (Norman, 1986).
The individual problem solving aspects in supporting human task performance are not unique to HCI. Most trades that involve designing things for human use have to deal with discovering the unknown and trading-off requirements, partial solutions and interactions between them, and have to deal with human abilities, limitations, habits and preferences, and how these change over time. Unique to HCI is that both the problem, the material as well as the requirements and constraints are not well known.

The problem is clearly: how to support human cognitive task performance. Intelligent cognitive behaviour, abstract information processing and dealing with technology are very young branches of human conduct. Abstract mathematical information processing is only a few thousand years old and it is only since the great wars that technology has become a household product. It should not come as a surprise that performing cognitive tasks is difficult and that little is known about how to support it. The problem has become particularly relevant during the last decennium now that the rapidly expanding use of digital technology is exponentially multiplying the number of cognitive tasks that the humble user has to cope with.
The same lack of knowledge exists with respect to the material used in performing cognitive tasks: information. Little is known about what exactly information is, and how much information of what kind constitutes too little, optimal and too much information. Something is known about the difficulty of dealing with abstract and symbolic information in comparison to real life information. A main difference between computer and pre-computer information processing is that the symbolic information of the former is less, less-well and less-directly related with physical reality. In terms of Norman (1988), symbolic information processing lacks the so-called affordances that are abundantly present in information processing by physical means. A genuine hammer or bicycle provide many more clues about how to use them, in which circumstances, and for which purposes than any pictorial, auditory, textual or virtually real representation of them on a computer. The relation between computers and software, and empirical reality is completely accidental and depends much more on learning and memory.
Finally, to establish and determine requirements and constraints for tasks with empirical or physical components is much easier to do than it is for information processing tasks. Requirements for tasks with a physical component such as building a bicycle can be derived from the empirically measurable properties of the task, the user and the context, such as the distance to travel, body strength, and the state of the road. Information processing tasks depend on cognitive psychological properties, which lack directly measurable, empirical equivalents, and since cognitive psychology deals with human information processing, it is a very young branch of scientific conduct that is difficult and knows little.

1.3 Essentials of Cognitive Ergonomics

In the previous paragraph it was argued that the combination of human cognitive task performance, the computer as a general purpose information processor, and the lack of knowledge about human cognitive behaviour and how it should be supported, is unique to HCI. This paragraph states what cognitive engineering is, or should be, in practical terms.

To address the specific problems and aims of HCI, cognitive ergonomics might be defined more precisely as: the science and engineering practice of user interface design. Although it is a short definition, it mentions several important and interesting aspects:

- cognitive ergonomics is a science and engineering practice.
- cognitive ergonomics is about user interfaces.
- cognitive ergonomics is a design trade.

Stating that cognitive ergonomics is a *science and engineering practice* means that it is not black magic or art that depends on individual characteristics such as experience, talent, wisdom and skill, but that it is an endeavour that yields, and should yield systematic knowledge and methods. As a result, cognitive ergonomics can be taught and, as a prerequisite, research should focus on establishing a firm knowledge and methodological foundation before moving on to other interesting questions or applications.

Cognitive ergonomics is *about user interfaces* because user interfaces provide human beings with the means to use cognitive tools and devices. In principle, cognitive ergonomics is not restricted to the use of computer systems just as user interfaces are not restricted to computer systems but since this study is situated in the area of human-computer interaction, cognitive ergonomics is taken to be about user interfaces as the means to create usable computer systems. Usability is not a characteristic of the functionality or the software of a computer system, but a characteristic of how users interact with computer systems to perform tasks in the user's work domain. This means that the term "user interface" in cognitive ergonomics is, and should be, extended beyond the most common definition as a piece of software. More specifically, user interfaces should be defined as: all the knowledge users need to perform tasks with a computer system (Moran, 1981; van der Veer, 1990).

Cognitive ergonomics is a *design trade* since its aim is to improve, rather than gather knowledge about, human-computer interaction. Design methods should be the main concern of cognitive ergonomics because only design creates user interfaces, and because design methods, used implicitly or explicitly, are what all design processes have in common. Analysis, prediction, and coding are techniques to supplement design methods but do not create user interfaces by themselves. Since design methods are the only systematic way to improve user interfaces, they are an obvious candidate to provide structure to the field of cognitive ergonomics.

1.3.1 Cognitive Ergonomics is Science and Engineering

This paragraph discusses the requirement that cognitive engineering be a science and an engineering practice. With respect to their development it draws parallels between the development of software engineering and cognitive ergonomics, and assessing cognitive

ergonomics in terms of the minimum requirements for being a science and engineering discipline it is argued that too much attention is spent on fashionable research subjects at the cost of research that aims to create the scientific knowledge base as an organised set of facts, tools and methods. As such, this thesis is about what is necessary and not about what is most popular.

First, cognitive ergonomics is a science and engineering practice. The question is not whether cognitive ergonomics is an academic subject, or to what extent it is a pure, empirical or applied science but, rather, whether there is something necessarily artful, mystic or creative about user interface design that precludes it from being approached in a predominantly methodological way.

On the one hand, questioning the scientific nature of cognitive ergonomics is only of academic interest, and makes little sense because it won't help solve any practical problems. On the other hand, asking to what extent cognitive ergonomics is an art that depends on creativity does make sense because answering this question determines to what extent problems can be solved methodologically and without the need to resort to individual insights, experience, and expertise.

When computers are regarded as theatre and user interface design is regarded as an art (Laurel, 1990, 1991) then the quality of user interface design is implicitly accepted as depending on the intuition, the skill and the talent of the individual artist. When, on the other hand, user interface design is regarded as cognitive engineering (Norman, 1986; Rassmussen, 1987), it is accepted as something that may be abstracted from the individual applicant, and explicitly passed on and improved, for instance by means of education. A similar reasoning applies to the field of computer programming, which may be regarded as an art (Knuth, 1968, etc.) but also as a type of engineering, as it is most commonly regarded and, even more stringent, as a discipline (Dijkstra, 1976).

The developments in cognitive ergonomics seem to closely resemble the early developments in software engineering at the time when computers were commonly regarded as mysterious devices and the exclusive domain of computer specialists. In order not to waste valuable hardware resources and to increase programming productivity, attention went to the selection of talented programmers (e.g. Perry and Cannon, 1966) and individual differences (e.g. Curtis, 1988a). With respect to the task of operating computers, research focussed on programming languages and language constructs (e.g. Sime et al., 1977; Green et al., 1980), program style guides (e.g. Kernighan and Plauger, 1974) and guidelines (e.g. Shneiderman, 1980).

The implicit rejection of programming as an art or individual craft made it finally possible to shift the focus of research towards the psychological aspects of programming and program understanding (e.g. Weinberg, 1971; Brooks, 1977; Curtis, 1988b) and creative problem solving (e.g. Guindon and Curtis, 1988).

Currently, facilitated by the consensus around object-orientation the field seems underway towards the notion of the "software factory" with standard reusable system architectures (CORBA, 1999; DCOM, 1998), domain- and software representations (UML, 1999), and methods for process control and maintenance (CCTA, 1998; Roa et al., 1996; Paulk et al., 1993).

A similar trend may be discerned with respect to user interfaces. Initial research aimed at creating recognition for user-friendly software, and asked questions about individual aspects of user interfaces, such as: response times, standards, guidelines and interaction styles (e.g. Shneiderman 1980; den Buurman et al., 1985; Cakir et al., 1980). Once 'usability' became recognised as a potential market force, the focus of research shifted to methods for interface- and interaction analysis and representation (Gaines and Shaw, 1986; Shackel, 1986), and the development of standard tools for creating user interfaces (Hartson and Hix, 1989; Myers, 1994; Sastry, 1994; Guthrie, 1995) which reflect a growing consensus about the importance of the user interface.

The developments in cognitive ergonomics are less uniform than those in software engineering. A less favourable trend is formed by the rapid succession of new and fashionable research areas and technologies (Newman, 1993). These include, among other, CSCW, multimedia, web-based interaction, persuasive computing and virtual reality, and newly discovered theories and theoretical approaches such as, among other, Situated Action (Suchman, 1987), Contextual Design (Wixon et al., 1990), Activity Theory, (Nardi, 1996), and Ethnography (Blomberg, 1995). In principle, there is nothing is wrong with such research, provided it leaves more than guidelines to spend attention to XYZ. In this thesis, a less readily discernible development is considered more favourable because it aims to create engineering solutions to the problem of cognitive ergonomic design: the development of integrated and structured methods for user interface design (this thesis; Hix and Hartson, 1993; Wilson et al., 1993; Lim and Long, 1994).

With respect to cognitive ergonomics our experience indicates the value of developing the field away from individual facts, ideas, issues, and uses of computer systems towards a more scientific and engineering kind of approach. The need for a scientifically valid cognitive ergonomics methodology is more important than a continuous search for yet other things that matter. The commonalities between introductory texts to HCI (e.g. Shneiderman, 1986; Johnson, 1992; Preece et al., 1994) show that there is a set of common facts, concerns, exemplars and standard solutions, but also that, presently, there is no general or standard method to design user interfaces that would unify all these elements in a paradigmatic way.

In other words, there is a huge collection of individual and isolated bits and pieces related to user interface design but there is no organisation among them. What is needed is a sufficiently large collection of basic knowledge, and applicable tools and techniques to solve the practical problems of user interface design in a structured, methodological way. Such a science base would not only constitute a thing that makes us smart but it is also a prerequisite for creating quality user interfaces, as well as a requirement to prove that cognitive ergonomics is a worthy trade partner. Cognitive ergonomics may be a respectable academic subject, but in general, it is not yet a respectable partner of software engineering nor a respectable resource for system design projects in business. In order to gain acceptance for the purpose of cognitive ergonomics as well as for the opportunity to improve things in the real world, there is a need for cognitive ergonomics as an engineering trade.

In addition to direct advantages associated with structured methodologies there are indirect advantages by making cognitive ergonomic knowledge less dependent upon personal experience and expertise. Regarding the application of knowledge, one can hardly expect to sell a product successfully when the package necessarily includes hiring the seller, and

ergonomic knowledge may be much more efficiently passed on in teaching and training when more and better organised knowledge, tools and methods are available.

Consequently, this thesis is not about the most recent technologically induced fashion: it is not about intelligent agents, not about networks, CSCW, virtual reality, and not even about graphical user interfaces or direct manipulation. Instead it is about the basic problem of cognitive ergonomics: user interface design and, more specifically, the development of a (formal) specification language for user interface design, and the use of this specification language within a general purpose method for designing user interfaces. In a sense, the subject material of this treatise will reflect today's, or perhaps, yesterday's state of the art in interface technology. However, with respect to the rather disorganised state of cognitive ergonomics as a user interface design science, it may be called fundamental research since it attempts to bring about some lasting structure among the field's floating facts.

1.3.2 Cognitive Ergonomics is about User Interfaces

This paragraph is about *user interface* as a concept. “The *user interface* of a system consists of those aspects of the system that the user comes into contact with – physically, perceptually, or conceptually. Those aspects of the system that are hidden from the user are often thought of as its *implementation*. Unfortunately, systems are almost never partitioned this cleanly, for the designer is usually not aware of what will show through to the user.” (Moran, 1981, pg. 4, original italics).
It is necessary to further refine Moran’s definition. According to Moran, user interfaces consist of different aspects and, as such, there may be different, and possibly incompatible, definitions of a user interface. Instead, it is argued that regardless of whether the user interface is viewed as a physical, perceptual, or conceptual entity, a common characteristic of these aspects is that users have to know these aspects to work with a system. Note that this also applies to any implementation aspects which “show through to the user”. Consequently, this thesis follows van der Veer (1990) in defining the user interface in terms of the knowledge that users may and should have to successfully use the system.
Given that people use computers to perform tasks, hence: engage in psychological behaviour, such a definition rejects the software engineering view of a user interface as a piece of software and replaces it by a definition that the user interface corresponds to what a user should know to successfully perform tasks with a particular computer system. Defining user interfaces in terms of competence knowledge has three main advantages: it helps to keep the focus on what user interfaces are built for, it allows for direct measurement of cognitive complexity, and, in comparison to defining user interfaces in terms of performance knowledge, it does not require a complete cognitive psychology.

At first sight, stating that cognitive ergonomics is about user interfaces may seem a rather restricted goal, since cognitive ergonomics and HCI are concerned with much more than interfacing computer systems and their users. The source of this misconception is a very restricted view on what there is between the user and the computer; not only is HCI concerned with much more than the user interface as a piece of software, but the same applies to the concept of "user interface". The user interface, from a software point of view, is indeed a

mere program, but from the user's point of view, or from the point of view of cognitive ergonomics, the user interface involves everything that allows users to perform their tasks.
Earlier, it was pointed out that, although interacting with a computer takes place by physical means, in terms of goal acquisition it is primarily a cognitive matter, and it is for this reason only that cognition is the basic premise of cognitive ergonomics. Observable physical behaviour and covert cognitive behaviour must be understood as two extremes on the same dimension. Between cognitive and physical behaviour, information processing takes place along the same levels that are used in, for example, language production and understanding, such as lexical, syntactical and semantic processing (see de Haan, van der Veer and van Vliet, 1991). Human information processing along these levels of abstraction is a two-way process; necessary for both physically issuing commands to the computer system, and for understanding the computer system's physical responses (Norman, 1984).

The existence of multiple stages and levels of information processing is also applicable to the software engineering side of user interface design, where it has been recognised in, among other, the Seeheim model of interactive systems (Pfaff, 1985). However, in software engineering the user interface has primarily been regarded in terms of the flow of control within computer programs and, consequently, software engineering has only taken up the responsibility for creating the user interface as a particular software module. The view of user interfaces as software modules is overly restricted and incorrect. It is incorrect because every piece of software is a reflection of the programmer's or designer's understanding of, and concern with, how users do their work. It is restricted because it makes concern with the user's information processing capabilities and restrictions, or the usability of a computer system, a matter of importance secondary to functional requirements.
Functional requirements at the level of a single computer program translate into the commands that the program offers to its users, but when looked at from the perspective of a user or an organisation in performing certain tasks, functional and usability requirements are one and the same thing.

There is also a major difference in the perspective on the levels of abstraction in user interface design between software engineering and cognitive ergonomics. In software engineering it is common to distinguish different levels of abstraction in program and system specifications. These levels, however, structure programs in terms of more and less abstract functional units, and not in terms of e.g. the lexical, syntactic and semantic entities and rules of the problem domain from the user's point of view. Software engineering specifications distinguish levels of abstraction to attain engineering goals, such as program maintainability, correctness, etc. (Sommerville, 1985; van Vliet, 1994), which is perfectly suitable for a functional use of computer systems but less suitable for a meaningful human-computer co-operation (Oberquelle, 1984).
The focus on functionality is especially clear with respect to so-called (interactive) user interface builders, which are rather like tools to draw display screens and screen elements. Few, if any, interface builders identify user tasks (goals and semantics) and dialogues (syntax), and they leave it to the designer to implement these in terms of the behaviour of subroutines, call-back functions and various widgets. With the introduction of object-oriented programming, analysis and design methods, some opportunities have evolved to (so to speak) free software from the Von Neumann style of engineering. Object-orientation stimulates

engineers to describe problems and solutions instead of requiring that problems are reformulated in terms of the functional procedures that solve them.

Above, cognitive ergonomics was defined in terms of user interface design, with the addition that the cognitive ergonomic notion of a user interface goes far beyond how user interfaces are understood in software engineering. It may be argued that cognitive ergonomics should be defined without referring to user interfaces at all, because user interfaces only exist because of accidental limitations in computer technology. As such, cognitive ergonomics should be defined in terms of what people could do with computer technology and how technology should be adapted to such uses.
In this thesis, the argument is followed that a clever combination of existing technologies is much more important to innovation in HCI in general than the influence of new technologies (e.g. Johnson et al., 1989), if only, because of the need for adaptation by a mass market that lags behind (Norman, 1998). The Star user interface, for example, although counting as one of the best known innovations in HCI was not created from new technology but rather from technology that became affordable at the time (Smith et al., 1982; Bewley et al., 1983). The same applies to the concept of Direct Manipulation (Shneiderman, 1982a) which changed the form of task performance (but not the principles). Direct Manipulation may only count as an innovation after taking into account the de-innovation of introducing indirect manipulation in computer technology in the first place.
Regardless that much of the published work in HCI focuses on the concept of novelty (Newman, 1993), in conclusion, it seems safe to state that the influence of technological innovation on cognitive ergonomics is rather limited and that developing design methodology is relatively future-proof. Because of the long-term benefits of methodology over collecting facts, developing design methodology is preferable over investigating new technology.

Earlier, it was argued that the user interface is not the mere program that addresses the interaction between user and computer system, and that it should, instead, include everything that is relevant to the eventual use of the computer system being designed. It was also argued that cognitive ergonomics, hence the user interface, is about cognitive behaviour in the form of human information processing along various levels of knowledge. Below, these notions are extended and related to human work, the purposive use of computer systems, and the notion of computers as general purpose symbolic processors, respectively.

Computer systems allow for an almost unlimited number of solutions to support human goal acquisition. Stated differently, to support a certain task many different user interfaces are possible. Each user interface, either as a program or in the cognitive sense, is a particular machine to support the user, in the same sense as a Pascal machine is a computer system (a Von Neumann machine) that supports usage of the programming language Pascal, irrespective of the underlying physical workings of the system. A user interface in this sense is what Oberquelle (1984) calls a Virtual Machine. "Virtual machines describe the functionality of a system in terms of abstract functional units and their behaviour. Implementation details and details of the underlying hardware are suppressed" (Oberquelle, 1984, pg. 30).

Oberquelle's view is still too limited because it focuses on the functionality of systems, and does not explicitly specify what the abstract functional units operate upon. To the human user these abstract data elements are rather real; they are the logical elements, concepts and objects people work with, such as files, forms, paragraphs, sentences and revenues. Consequently, user interfaces are virtual machines which should include the abstract data elements, to be employed by users to perform their tasks.

User interfaces are virtual machines because there need not be a one-to-one mapping between the behaviour of the machine and its realisation in hard- and software. Users working with the same realisation or implementation of a user interface may have a different understanding of the workings of the machine, irrespective of the idiosyncrasies in the individual user's knowledge. Likewise, people with a similar understanding of how to perform their tasks may actually be using different interfaces.

In order to allow for differences in understanding the workings of a computer system, whilst avoiding to define user interfaces in terms of individual user knowledge, user interfaces are defined in this thesis in terms of user competence knowledge: the knowledge a user should have to successfully perform tasks with a particular user interface (van der Veer, 1990; Tauber, 1988).

A major advantage of defining user interfaces in terms of user knowledge is to help keep focus on what computer systems are meant for; namely, to help the user to perform tasks, and to keep focus away from (or perhaps rather: within proper boundaries of) the software engineering problem of how to create the software implementation. It would be a most peculiar world indeed to have the ways of sleeping, drinking coffee and fighting wars determined by the ease of manufacturing beds, mugs and guns.

Focusing on computer systems as tools to enable users to perform their tasks does not preclude that attention is spent on implementation questions. Defining a user interface in terms of the user's tasks only says which tasks and (user) data elements should be made available. It does not say anything about how these are implemented. User task considerations have been neglected to such an extent that the term "too little too late" (Reisner, 1983, 1984; Lim and Long, 1994) became common to indicate that, usually, there is little concern with the user's tasks in software design, and if there is, it is usually introduced only after the main design decisions have been taken, and little room for improvements is left. From this perspective, it is a plain necessity to focus on user task performance.

A second main advantage of defining user interfaces in terms of user tasks is the ability to measure and compare cognitive aspects of user interfaces. Given that there are currently no satisfying measures to determine even the complexity of the computer software itself (e.g. to predict the ease of understanding a program) it should not be difficult to see that predicting the complexity of user interfaces to the user on the basis of program listings is even much harder. It is argued that specifying user interfaces in terms of user task concepts is identical to specifying user interfaces in terms of what a user has to know about the interface in order to use it.

Apart from the theoretical advantage that using knowledge specifications helps to establish links between the practical work of cognitive ergonomics and the theoretical research in

cognitive psychology, knowledge specifications have practical advantages. To determine the ease of learning a particular user interface is little more than to measure the size and complexity of its knowledge specification. Similar methods may be used to measure other cognitive aspects of user interfaces, such as ease of use, cognitive load, etc. To help solve the "too little too late" problem, such methods can be used as soon as a user interface has been (partially) specified, without the need to have an actually working system available.

Finally, the reason to propose defining user interfaces in terms of the user's competence knowledge is meant to keep things as simple as possible. The term competence knowledge is proposed by Chomsky (1965) to distinguish between the knowledge people need to understand language (linguistic competence), and the knowledge people use to write or speak (linguistic performance). To describe what it takes to understand a language (the 'grammar' of the language) is much easier than to describe what people know and how they use this knowledge to produce linguistic utterances, if only because of the amount of planning involved.

To describe the linguistic performance requires a fairly complete theory of human behaviour. In a similar vein, it is much easier to describe user interfaces in terms of user competence knowledge than it is to create performance knowledge descriptions of user interfaces, apart from the fact that a reasonably complete performance theory would at first require one to solve the problems of cognitive psychology (Green, 1990). Apart from the sheer lack of time to do that, for the 'limited' purpose of specifying user interfaces it will not be parsimonious either. A competence description completely specifies a user interface in terms of what a user has to know about the interface. A performance description, such as a Programmable User Model (PUMS; Young, Green and Simon, 1989) also specifies user choices, errors, lapses of memory and attention, etc. It remains to be seen whether knowing what real users will do, compared to what they need to know, can justify the additional work that is required, especially with respect to design purposes.

This thesis describes a notation and the development of the notation to represent user interfaces from a user-task point of view. This notation is used to create complete competence models of the knowledge users need to perform their tasks, which includes the "how-it-works" knowledge (the conceptual entities and the user's virtual machine) as well as the "how-to-do-it" knowledge (the task-command mapping language). It will be shown how these competence models can be used for purposes other than describing user knowledge, but no attempt will be made to describe how real users will employ this knowledge.

1.3.3 Cognitive Ergonomics is about Design

In this paragraph it is argued that cognitive ergonomics as a science is not sufficient but that it also has to prove its usefulness in actual design. Since real practitioners don't eat theory it is necessary to develop methods, tools and techniques as means to translate the theoretical knowledge of cognitive ergonomics to practical ends. For this reason, the thesis seeks to develop, not only a notation to describe the knowledge that is required to use a particular user interface, but also a method to design user interfaces that is based on the notation.

Cognitive ergonomics is not ready for the practice of design by taking a scientific and engineering stance towards studying user interfaces. In pure sciences, like psychology or physics, it makes sense to try to understand how things work, while keeping the application of such knowledge an implicit aim. Cognitive ergonomics is not only by name an applied science; it would be fairly ridiculous to study practical questions, and especially questions that relate to the state of technology, without practical purposes. As an applied science, an engineering trade, or whatever label is stuck on it, cognitive ergonomics needs a vehicle to bring about the improvements it is meant to produce.

In a competitive world, at least with respect to resources, a political vehicle or necessity is to gain acceptance that it pays off to consider human aspects in devising computer systems. Assuming that decision makers are well-informed rational beings, this is not necessary anymore after reports by, for instance, Gaines and Shaw (1986), Eason (1988), and Bias and Mayhew (1994). A more direct, but still indirect vehicle for improving human-computer interaction is to spread cognitive ergonomic knowledge among designers by training and education, as exemplified by the ACM SIGCHI report on Curricula for HCI (Hewett et al., 1992). Spreading the gospel and the teachings, however, does not guarantee that more usable computer systems will evolve.

In order to directly influence the usability of computer systems, cognitive ergonomics should be able to offer tools, techniques and methods to those responsible for system design. Practitioners, contrary to academic researchers, have no particular use for theories. Theories, including the practical theories of cognitive ergonomics, are concerned with truth rather than with creating things and, therefore, cognitive ergonomics should propose methods, techniques and tools.

Methods are understood as systematic sets of abstract procedures to bring about certain goals. Methods are evaluated by their ability to acquire the particular goal, such as the goal to create usable systems. Methods are closely linked to theories which explain why the procedures are used and why they are used in a particular order and combination. Techniques, for example, the use of pencil and paper prototypes for early evaluation of design options fall somewhere between tools and methods. Techniques do not depend on theories but to apply them still requires interpretation of the problem and translation into activities. Tools, such as using particular questionnaires or software for analysis, are generally smaller in scope, leave less room for interpretation, and are more restrictive than the other two.
In practise, the difference between methods, techniques and tools is used more gradually. Methods, such as the ones used to structure design processes or perform evaluations, are used as tools that happen to require more expertise, to take longer, and to be concerned with answering more general and higher level questions than genuine tools do.

What is important is that tools, techniques and methods and, to some extent, even theories, are not very valuable when used in isolation. In order to provide a context to use and interpret them, and to relate them to higher-level concepts, such as usability, they require organising principles and, more specifically, a design approach.
The relevance to this thesis is that the original aim was to create and document a method or notation for user interface design representation: which formal notation is most suitable to

represent user interfaces during the design process from a user's point of view. This question is too limited with respect to organising user interface design. Good user interface design, and good user interfaces, do not result from piling up more building blocks, and creating 'yet another' formalism for user interface design representation would only add to the confusion.

What should ideally be done is to create a building block, in this case a notation for design representation, such that it will fit together with other parts of the user interface design process. This is not an easy undertaking because most parts of design, like task analysis, creative design and the various software design elements, do not have a well defined interface. A notation for design representation may fit well with one method for task analysis but not with another.

What has been done, instead, was to take a dual approach. First, given that it is not clear what kind of notation is required for a good integration with other parts of the design process, focus on what is required to make a good notation for design representation. Here, a good design representation is taken as one which captures, as completely as possible, all the characteristics of a user interface that are relevant for its usability. Instead of selecting a bolt to fit a certain nut, finding out what a good bolt is makes sense, especially since that should be done anyhow.

Second, given that the overall design method provides value or meaning to the building blocks for user interface design, once a good notation for design representation was established, efforts focussed on creating a design method for usable computer systems, using the design representation as a basis. Using the nut and bolt metaphor, one might say that once you know what a good bolt looks like, you create a fitting nut.

Following the dual approach has a number of advantages. It gives context or meaning to the notation for design representation: instead of a mere notation, the method shows how it is used and, thereby, why it is structured as it is. Also, instead of ending up with only a building block, with "just a little" extra work, the result is a full-fledged design method that enforces, or at least stimulates, designing usable computer systems. Finally, the design method provides a context for using the representation, by showing when and how the representation might be used within different design methods.

On a higher level of abstraction, with respect to the status and purposes of cognitive ergonomics, it is also important, if not essential, that the result should be a design method, rather than a tool for design representation. In comparison to methods and theories, tools have a limited scope: like a tool may be said to operationalize particular aspects of a method, a method may be seen as a procedural prescription, just what it is, but, more in general, also as a summary of a number of facts, things to consider, and things to do. ETAG-based design (chapter 4) is a general procedure to represent user interface designs but it is also a means to capture, to represent, and to allow derivation of a great many facts and procedures about user interface design in general.

In comparison to techniques and tools, methods are more powerful means to convey the knowledge and purposes of cognitive ergonomics. This applies even stronger to theories, provided that they are known, that they are considered by designers, and that they are generally taken to be true. Individual tools and techniques for e.g. analysis and prediction in

user interface design do not tell as much about design as methods do, especially concerning the reasons for doing and not-doing certain things.
With respect to the relation between cognitive ergonomics and software engineering, it may be added that proposing an alternative method provides a stronger argument to promote a prime focus on usability than adapting techniques to software engineering methods or continuing to complain that software engineering, at least in practice, spends too much attention to the technical aspects of user interface design.

Regarding the development of cognitive ergonomics, focussing on a method for user interface design also seems to be a healthy thing to do. Cognitive ergonomic research has produced a large number of facts and ad-hoc tools and techniques. In order to structure or "summarise" this material, there is a clear need for higher level knowledge, such as methods and theories.
Earlier, a disappointment was mentioned about the way in which theories in cognitive ergonomics are subject to fashion. Theories are proposed, exposed, and forgotten about, even when they rest on firm, empirical investigations, like most of the formal modelling techniques discussed in the next chapter. When theories get lost in fashion, and tools in their own multitude, it seems best to put the prime focus on practical validity, and let the customers of cognitive ergonomics have the last word, like they should.

It is for these reasons that this thesis will not stop at a notation for user interface representation based on theoretical considerations. Instead, it proposes to proceed with developing a design method for user interfaces that is almost exclusively based on the consideration that the primary aim of design is to create usable computer systems, rather than to aim at the secondary goal, to make the design technically feasible and manageable.

1.3.4 The Main Research Questions and the Structure of the Thesis

The previous paragraph argued that in order to further the development of cognitive ergonomics as a science and engineering practice that is concerned with the design of usable human-computer systems the choice was made to follow a dual approach and to answer two questions:

- What is a good representation to model usable human-computer designs for design purposes?
- How to create a design method based on a good representation of HCI design?

The first question, about a good representation for design purposes, is answered in two stages. First, the criteria for good design representations are developed mainly using theoretical considerations and from the (formal) models that are described in the literature a most promising candidate is chosen. This stage concludes with a description of the content, characteristics and application of the selected model: Extended Task-Action Grammar (ETAG). Three key issues in this stage are that formal models are preferable over non- or semi-formal models, that the models themselves should contain or carry measures of the usability of the design, and that models which provide a conceptual specification of the design are preferable over models that do not.

In the second stage, and mainly using practical considerations, a number of application studies evaluate the use of ETAG to answer different design questions, and how it may be improved further. The result of this stage is a number of recommendations concerning the further development of the ETAG representation. A key issue in this stage is lack of a specification of the perceptual aspects of designs.

In summary, in order to find a good representation for HCI design, four subquestions are asked:

1. What criteria should be used to assess and evaluate models for design representation? (chapter 2.1 - 2.3, 2.5)
2. Which models are available that are or may be developed into models for design representation? (chapter 2.4)
3. Which of the models is most suitable or most promising with respect to the assessment and evaluation criteria? (chapter 2.4 - 2.6, chapter 3)
4. How well does the selected model fare to answer different design questions and how can it be improved? (chapter 5 - 9)

To answer the second question, about the creation of a design method, the starting point is that in ETAG-based design the ETAG representation is used throughout the design process as a red thread to represent the evolving state of the design product, and where the model fails to capture the information that is required at a particular stage, other representations will be used. The first step is to identify the greatest common denominator (the 'received view') among the available methods for user interface design and to determine how it might explain the failure to create usable human-computer systems. A key issue here is the conclusion that the usability problems rather seem to follow from taking the wrong perspective on the user interface than from using the common method.
The second step is to ask how the user interface should be viewed from the perspective that it provides a virtual task world that users must know to perform their tasks by means of the computer system. As a preliminary to this question, the thesis first discusses a number of implicit and explicit definitions of the user interface as concept and reviews how appropriate each of these definitions is. A key issue here is that the user interface should not be seen as a software module or a collection of functions but as a complete definition of the environment (or world) in which a user has to perform his or her tasks.
In the third step we create and develop a framework to structure user interface design and use that to provide an overview as well as a detailed description of ETAG-based design. Key issues in ETAG-based design are to first lay down a complete task world before specifying the details of functionality and task-actions, and to attempt to avoid global iteration cycles by means of iterations within design steps.
The fourth and last step consists of validating ETAG-based design. Rather than to evaluate the design method as a whole and apply it in different circumstances to create actual user interface designs, the ETAG specification language was used in different design stages to analyse for which purposes the model could or could not be used as the sole representation. As such, there is only a small perspective difference with the evaluation questions concerning the first main question.

In summary, to answer the question of how to create ETAG-based design, four questions are asked:

5. Is there a general or 'standard' method for user interface design and how does it fare? (chapter 4.2)
6. What is the appropriate view or definition of the user interface as a concept? (chapter 4.3)
7. What does ETAG-based design look like? (chapter 4.4 - 4.5)
8. How can ETAG be used in the different design stages and how should it be improved or complemented? (chapter 5 - 9)

The main method that is used to answer the questions is analytic and theoretical in nature. Rather than to seek answers that apply in particular empirically investigated area's, the thesis aims to establish general and principled answers. From the idea that theories summarise empirical research, theoretical requirements and criteria are used to evaluate the applicability of formal models in general and ETAG in particular. Whenever it is necessary, a notation is first applied to design questions to create a description or a demonstration system for applying the theoretical requirements and criteria. The correspondence between the results of applying the notation and the criteria and requirements is subsequently used to arrive at principled evaluations of the notation and to suggest improvements.
The questions asked above are not answered in the order given; instead, the structure of this thesis has a logical order that consists of an introduction to modelling approaches in HCI, a description of ETAG as a particular model and ETAG-based design, and it continues with a treatment of the role of ETAG in each design stage, followed by a general evaluation, as follows:

Chapter 2 discusses various formal modelling tools in HCI and selects one model (Extended Task-Action Grammar or ETAG) that fulfils most of the requirements for specifying user interfaces for design purposes.

Chapter 3 contains an in-depth discussion of the background and the constituents of ETAG. It discusses the psychological background and validity of ETAG on the basis of theories and evidence from psychology, linguistics and logic.

Chapter 4 discusses how user interfaces are looked at from the perspective of ETAG, and it outlines ETAG-based design, as a user interface design method that is meant to enforce, or at least stimulate, the design of usable computer systems.

Chapter 5 starts with a discussion of task analysis directed at user interface design, and discusses the elements that should be part of any method for task analysis. The chapter further exemplifies and evaluates using the ETAG notation in the task analysis process.

Chapter 6 describes the general procedure to create an ETAG representation and it provides assisting guidelines and lesson-learned. It also provides a detailed a step-by-step example of creating and using an ETAG representation in a design project.

Chapter 7 focuses on one way to use of the ETAG notation for a purpose other than design specification, namely the use of the ETAG representation as a means to analyse user interfaces on usability characteristics. Applying ETAG for this purpose is also used to validate the notation itself.

Chapter 8 is dedicated to another way of using ETAG for another purpose than as a notation for design specification, namely as a source of information about an interface that may be used for generation of help information and prototyping purposes.

Chapter 9 contains a general discussion of the findings. It consists of a summary of the findings, in which strong and weak sides of ETAG are discussed. On the basis of the summary of findings a number of recommendations are made about the development of ETAG and, more in general, the development of user interface design methodology.

Chapter 2:

Formal Modelling Techniques in Human-Computer Interaction

When I think of formal scientific method an image sometimes comes to mind of an enourmous juggernaut, a huge bulldozer -slow, tedious, lumbering, laborious, but invincible. It takes twice as long, five times as long, maybe a dozen times as long as informal mechanic's techniques, but you know in the end you're going to get *it.*

Pirsig, R. (1974).

Abstract

This chapter is a theoretical contribution, elaborating the concept of models as used in cognitive ergonomics. A number of formal modelling techniques in human-computer interaction will be reviewed and discussed. The analysis focuses on different related concepts of formal modelling techniques in human-computer interaction. The label "model" is used in various ways to represent the knowledge users need to operate interactive computer systems, to represent user relevant aspects in the design of interactive systems, and to refer to methods that generate evaluative and predictive statements about usability aspects of such systems. The reasons underlying the use of formal models will be discussed. A review is presented of the most important modelling approaches, which include External-Internal Task Mapping Analysis; Action Language; Task-Action Grammar; the Goals, Operators, Methods and Selection model; Command Language Grammar and Extended Task-Action Grammar. The problems associated with applying the HCI formal modelling techniques are reviewed, and possibilities to solve these problems are presented. We discuss the future work that needs to be done, i.e., the development of a general design approach for usable systems, and the need to focus attention on the practice of applying formal modelling techniques in design. Finally, to attain those purposes and on the basis of the review conclusions, we will select ETAG as the modelling technique that will form the starting point for further investigation into user interface design on the basis of formal specification.

2.1 Introduction

The use of computers by non computer experts has sharply increased during the last decades. For this group of users, the computer is only a tool to get their work done, comparable to a pencil or a notebook. As such, to learn and use the computer system is an additional task derived from the application of this specific tool in the course of performing the primary task: the work being done (van der Veer, 1990), and therefore the effort of having to learn and use a computer system should be minimal. To reduce the effort it is important to adapt the operation of the computer system to the characteristics of the users, of the tasks and of the task environment. Traditionally, the design of computer systems has been the subject matter of computer science; users and people in general were studied by (cognitive) psychology; and tasks and task environments belonged to the fields of organisational psychology and ergonomics. Regarding usage, computer system design can not be adequately covered by any of these individual fields of study and therefore a new field of study has been developed with contributions from all these disciplines: cognitive ergonomics.

In cognitive ergonomics (the study of human-computer interaction) the subject of study is the usability of computer systems to perform the user's tasks. In a sense, cognitive ergonomics is not very interested in computer systems at all, i.e., most characteristics of the hardware of the

machine are of minor concern, such as its speed and the amount of memory. Instead, the main focus of cognitive ergonomics is on operating the system to accomplish users' tasks and this is primarily a matter of mapping system characteristics to human skill, knowledge, strategies, needs and preferences. From the viewpoint of the user, operating a computer system is not merely pressing buttons, but rather building an understanding of the system, as if it were a human partner (Oberquelle, 1984).

To analyse the computer system in this sense, a number of concepts have been introduced. Oberquelle has introduced the virtual machine, as the functionality of a system in terms of abstract (hence: virtual) functional units and their behaviour, without considering the details of the implementation and the hardware. In a similar way, the term user interface is used to include all perceptually and conceptually relevant elements and behaviour of a computer system that the user might know about, and should know to perform his tasks successfully. One might say that ideal or competent users, who have full knowledge of how to use a computer system to accomplish all possible tasks, know everything about the user interface in the Cognitive Ergonomic sense (van der Veer, 1990; Tauber, 1988).

Note that in software engineering the term user interface has a restricted and different meaning: the piece of software that controls the communication between users and the rest of the system. In the remainder of our contribution we will use the term computer system with the same meaning as, and interchangeable with, the terms "virtual machine" and user interface, to denote the system as far as it is relevant for a user, and hence may be either perceived directly, or may be conceptually conceived by interacting with the system.

In cognitive ergonomics, the usability of a system is assumed to depend on the organisational circumstances in which the computer is employed, on characteristics of the intended users of the computer system, on the tasks they have to perform, on the style of the dialogue with the computer system, and on the physical environment. Matters of concern are, among others, job and task analysis, task and computer experience, skill and problem solving, and measurements of office equipment. In cognitive ergonomics it is generally agreed upon that any system designed to be used by people should meet certain requirements. For example, Gould and Lewis (1985) state that a computer system should be:

- **functional**
- **easy to use**
- **learnable**
- **pleasant to use**

A computer system can be said to be usable, to the extent that it meets these requirements. Usable systems then, should provide the users with the functions they need to fulfil their tasks (functionality). The operation of the computer system should not require extensive mental or physical effort (easy to use). The operating procedures of the system should be easy to learn and easy to remember after periods of not using the system. And finally, using the computer system should be enjoyed by users. The joy of using computers will not be dealt with in this chapter, because it is difficult, if at all possible, to approach this question with formal techniques. The first three requirements, of functionality, ease of use and learnability are directly related to what users have to do to perform their tasks and what the computer system does to support these activities.

It is necessary to distinguish the general concept of usability of computer systems in the sense of fulfilling the first three (or even all four) of the above mentioned requirements, and the separate requirements. A computer system that is easy to use is not necessarily a usable system since it may be lacking in functionality or it may be very difficult to learn. We will reserve "usability" and "usable", to refer to the quality of the system as a whole, and reserve "easy to use" and "ease of use", to refer to the narrow sense of demanding little mental or physical effort,

Formal modelling techniques or approaches can be used to represent the knowledge the user needs and/or the actions the user should perform to delegate his tasks to the computer system. We will not distinguish between the terms technique and approach and use these terms interchangeably. What is important is that applying formal modelling techniques results in models of knowledge and behaviour, which can be analysed to investigate the extent computer systems fulfil the three requirements for usable systems.

2.2 Models and Levels Of Abstraction in HCI

In order to develop a better understanding of what is involved in designing usable systems, it is necessary to take a closer look at the role of the user in operating a complicated device such as a computer, and introduce the notion of a user's mental model. Norman (1983) distinguished between three types of model: the user's mental model, the system image and the conceptual model of a computer system (see Figure 1).

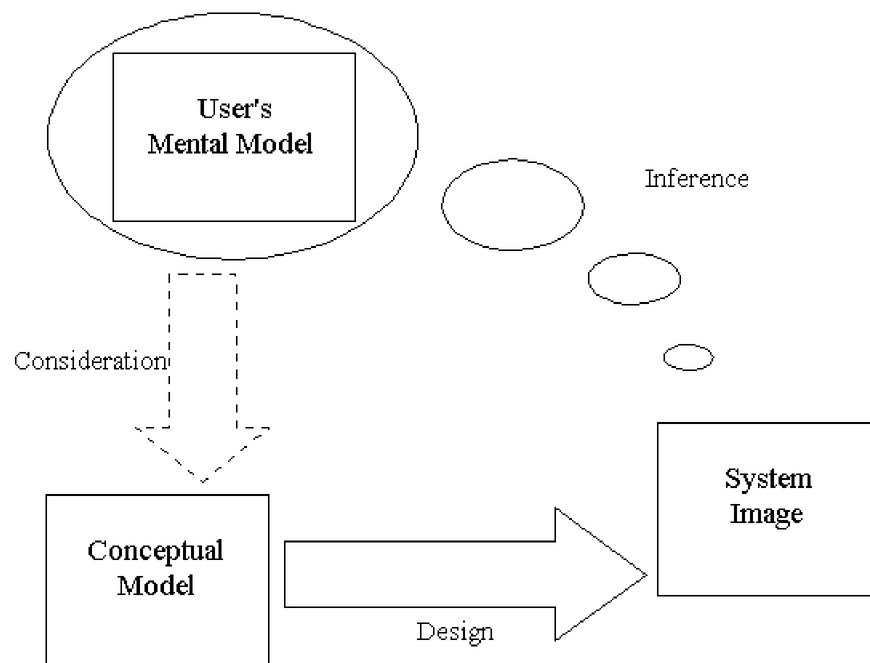

Figure 1. Models in HCI (Norman, 1983).

(a) The *user's mental model* is a model of the machine that users create or, according to Norman, which naturally evolves when learning and using a computer. This type of model does not have to be, and usually is not, accurate in technical terms. Instead, it may contain mis-conceptualisations, omissions and it does not have to be stable over time. However, a user's mental model is indispensable for the user to plan and execute interaction with the system, to predict, evaluate, and explain the behaviour of the computer, and to reduce the mental effort involved.

(b) The user's mental model is based on *the system image*, which includes all the elements of the computer system the user comes into contact with. As such, the system image includes all the aspects ranging from the physical outlook of the computer and connected devices to the style of interaction and the form and content of the information exchange. Although Norman (1983) excludes teaching materials and manuals from the definition of the system image, these could be included as well because they also shape the image of the system.

(c) The third type of model that Norman distinguishes is *the conceptual model* of the target system. This is the technically accurate model of the computer system created by designers, teachers and researchers for their specific purposes. As such, this type of model is an accurate, consistent and complete representation of the system, as far as user relevant characteristics are involved.

The important point of Norman's distinction between three types of models is that in well designed systems the conceptual model of the designers forms the basis for the development of the system image, which in turn is the basis for the evolvement of the user's mental model. A good design starts with a conceptual model derived from an analysis of the intended users and the users' tasks. The conceptual model should result in a system image and training materials which are consistent with the conceptual model. This, in turn, should be designed in order to induce adequate users' mental models.

In the introduction we remarked that for many if not most computer users working with and getting to know a computer system is not a goal in itself, but instead the computer is used as a mere tool to fulfil task goals. Except for courses in computer literacy and playing computerised games of skill, learning how to handle a computer system is of secondary importance, whereas task performance is of primary importance. As such, the effort required in learning and using the computer should be minimised. From this starting point, the three requirements of a good computer system can be derived. In the first place, the computer system should be designed in such a way that it optimally supports the tasks of the users (i.e. the requirement of functionality). In the second place, operating the computer should require a minimal amount of effort and attention (i.e. the requirement of ease of use). And finally, learning and remembering how to use the computer system should be as easy as possible (the requirement of learnability).

Formal modelling techniques in human computer interaction are used to represent the knowledge users need about the operation of a proposed computer system, and to describe the actions users have to perform to delegate tasks in order to attain their task goals. By analysing these representations, something can be said about whether the design of a computer system meets the required functionality, and to what extent the design will be easy to use and

learnable. Formal representations show the complexity of the knowledge a user needs to acquire in relation to the tasks to be delegated.

Users need knowledge about a computer system for "translation" in two directions. First, users come to the computer system with a set of tasks, and they will have to know how to translate and rephrase task delegation into the operating procedures and commands provided by the computer system. Secondly, after a command has been supplied to the system, the user must know how to interpret the behaviour of the system, and how to determine the success or failure of the task attempts. That is, users have to know how to translate their highly abstract task goals into the physical actions towards the computer system, and know how to relate the physical responses of the system to task goals and to task knowledge. In human-computer interaction it is common to distinguish several levels of abstraction in this specification/interpretation cycle (e.g. Moran, 1981; Nielsen, 1986; Fröhlich and Luff, 1989). Nielsen proposed the following levels of abstraction in the knowledge of computer users:

- **Task Level**
- **Goal Level**
- **Semantic Level**
- **Syntax Level**
- **Lexical Level**
- **Physical Level**

The essence of the notion of levels is that a user and a computer system communicate via certain types of languages that are different from "natural" language. The correspondence between the meaning in the user's mind and the physical exchange of tokens between user and machine is far from trivial. Therefore, on the one hand, when a computer is used to get certain tasks done, the user has to rephrase his or her intentions in a top-down manner to provide the system with the commands in its own language. On the other hand, the user has to understand the responses of the computer in order to be able to evaluate the state of affairs of his tasks, and for this part the user has to cross the levels in the opposite direction.

For example, a secretary who receives the instruction (task) to make a copy of a letter sent to a customer must know that to copy a letter means to reproduce it on paper (goal level), which, in terms of a computer system, means to send a particular text file to a printer device (semantic level). This user has to know that commands must be submitted to the computer by specifying the operation (e.g. printing), a delimiter, an ordered list of arguments (e.g. printer destination first, letter identification second) and an end-of-command indicator (syntax level). Further, this user has to know that the letter is called 'smith.txt', that the command to send something to a printer is called 'print', and the name of the particular printer (lexical level). Finally, to submit the command the appropriate keys have to pressed (physical level).
Now, suppose that a message appears: "device RP not available - cannot print smith.txt". From this message, the user will have to infer that the command was correctly parsed, the file was correctly referred to, but that the system could not identify the intended printer. The error message indicates that, syntactically and lexically the command was correctly understood, but that for the system, something is wrong at the semantic level. In this example, the source of the error might actually have been a lexical error or even physical slip, such as typing 'RP',

where 'PR' was meant, or a genuine semantic mistake, when experience with another system caused the user to expect that there would be a printer called 'RP'.

Especially where it concerns larger and more complicated computer systems, it will be more difficult for a user to have a complete and flawless knowledge of each of these levels. Only expert users will have a mental model which will be consistent with the conceptual model of the system. In terms of Norman (1983), the user's mental model will usually differ from the conceptual model of the system. However, when a user interface is consistently structured and allows for a clear and straightforward mapping between the levels of abstraction then it will be easier for a user to develop an adequate mental model of the system. The key question is then: how to design user interfaces in such a way that the development of an adequate user's mental model is stimulated.

2.3 Methods to Enhance the Usability of Computer Systems

Once a system is designed and implemented, measuring how usable it is, is relatively straightforward. When a computer system is successfully used, one may assume that the system did indeed stimulate the development of adequate mental models. For design purposes, empirical testing of the final product alone is not very practical; empirical testing is difficult and costly, it requires extensive effort, and its results come very late in the development cycle, often even too late to influence the design (Reisner, 1983). A number of methods and tools have been developed to enable predictions regarding the end product's usability at an early stage of the design process.

One approach is to use design methodologies which involve active user co-operation during the whole process of design, to avoid that the design differs very much from what it was meant to be. Proposals for design methodologies like this frequently involve iterations to account for changing system requirements which may arise when users get familiar with using the new system.

Evaluating prototypes can also be used to determine if a system, or parts of it, are usable. Prototyping refers to building parts or particular aspects of systems in advance to enable thinking about or evaluating design options. Prototyping generally refers to "high fidelity prototyping" (Diaper, 1989a). A high fidelity prototype is a computer system that simulates the behaviour of the proposed computer system rather accurately. One may also apply low fidelity prototypes, for instance, to mimic the presentation of screens of the intended system by systematically turning over the leaves of a book with pictures of example screens. In the Wizard of Oz technique (Diaper, 1989a) the reactions of a computer system are not generated by the system but, instead, by a human operator.

Another approach is to use analogies and metaphors, which means that the design is built around an image borrowed from everyday life, like the furniture and tools in a clerk's office. Using a familiar image aims at letting users build on established knowledge, without having to start from scratch to build a mental representation of a system. A well-known example is

the desk-top metaphor: to present the computer system to the user as if it was organised according to the familiar desk and the objects and functions related to it.

A technique that designers expect to be really practical for improving the quality of design is to use guidelines and standards, or simply to look at the products of successful competitors. Applying guidelines like "never use more than six colours" may save a large amount of time and effort, otherwise spent on user testing. The results will, however, heavily depend on the validity of the extrapolation from one situation to another, and on the particular combination of features. In this respect, it is often mentioned that existing guidelines are too general for a specific design question, that just the opposite applies, or that guidelines may contradict each other. An extensive and well known collection of guidelines can be found in Smith and Mosier (1984).

Finally, one may use modelling techniques to capture and analyse the knowledge the user needs to delegate tasks on a proposed computer system. Using these techniques, the analyst makes a more or less formal representation of the relations between the task-goals of the user and the operations needed to reach these goals. By analytic methods that estimate the complexity and consistency of the formal model, e.g. counting the number of rules or calculating the average number of parameters of each rule, something can be said about the functionality, ease of use and learnability of the proposed system.

Above, the various methods to ensure and improve the quality of designs have been presented separately. In actual design, however, combinations of methods need to be used, because no method can resolve all design questions, whereas combinations might produce better results. Which particular method to use depends on the specific question and the ability of a method to solve it, and on how much time and effort is needed to apply the method.

In the following, we will concentrate on the use of formal modelling techniques to model user knowledge. We expect that formal modelling techniques have a number of advantages the other methods do not have, even though their use is not as well established as some other methods. The main advantage of formal models is the possibility to specify a design very precisely, without ambiguity. Using a formal notation also creates the possibility to automate parts of the implementation of user interfaces, and provides more possibilities for rapid prototyping and user testing.
Moreover, analytic methods applied to formal models (aiming at establishing indexes of ease of use, learnability, and functionality) do not require a working system, as opposed to using empirical measurement techniques. Because only an initial specification of the user interface is required, formal modelling techniques may be applied very early in design to predict some of the usability aspects of the system. Also, since neither users nor working systems are involved, formal modelling can be applied at relatively low cost in comparison with techniques that require more than just a specification. We do not mean to say that empirical testing is not needed at all. Rather, we would like to suggest that applying formal models could enable the answer to certain questions in design earlier and with less effort than empirical testing, thereby leaving more resources for other methods to improve the quality of the design.

A further advantage, which formal techniques share with using analogies and metaphors, is that they closely adapt to Norman's notion of mental models. Formal models and metaphors both describe aspects of the conceptual model of the system and aspects of the user's mental model, but metaphors give an informal account of the conceptual model, and formal models represent aspects of it (e.g. the "how-to-do-it" knowledge of a competent user) formally. Whereas metaphors refer to the conceptual meaningfulness of mental models, formal representations present the structure and consistency of an adequate mental model (i.e., a mental representation that is compatible with the conceptual model). The other advantages of formal models combined with their close connection with the notion of users' mental models makes them suitable candidates to serve as the conceptual models to base design upon.

2.4 An Overview of Formal Modelling Techniques

In this overview, the most important or well known of the formal modelling techniques to represent user knowledge will be treated. Applying formal models for user interface design includes some variant of a general procedure:

(a) the analyst makes a list containing what the users' goals can be and what users have to do in order to reach these goals;
(b) a model is built from the goal-operation sequences, using a more or less formal representation language e.g. Backus-Naur Form;
(c) the model is restructured to represent the knowledge of a target group of users, such as "novices", "competent users", "occasional users". Many techniques model the knowledge of the "ideal user", who is assumed to have full knowledge of operating the system and does not make any errors;
(d) the model is analysed using some metric, e.g., providing indications of complexity of the transcription rule system, of the discriminability of sets of tasks or sets of objects, of the consistency of representations at different levels of abstraction.

The differences between the various kinds of models include characteristics of the formal language (conceptual basis for representation), levels of abstraction used in representing the communication between the user and the computer, methods available for the analysis of ease of use, learnability, and functionality. The most important reason behind these differences relates to the main goal the model is constructed for. Various categorisations of formal modelling approaches have been proposed (e.g. Green, Schiele and Payne, 1988; Murray, 1988; Oberquelle, 1984; Rohr and Tauber, 1984; Simon, 1988; and Whitefield, 1987).

The main problem in comparing these different categorisations of approaches to formal modelling is that they distinguish the modelling techniques to a large extent on scientific dimensions derived from the field of research, instead of practical considerations. For example, Green, Schiele and Payne (1988) ask whether a model describes performance or competence aspects of behaviour. Murray (1988) distinguishes prescriptive and descriptive models. In a similar vain, Simon (1988) uses the degree of idealisation to distinguish between modelling ideal behaviour and real behaviour. Finally, Nielsen (1990) and Whitefield (1987)

distinguish models, based on whether the model is owned or created by the user, the designer, the computer system, or the researcher, and what or who is being modelled.

Our intention is to analyse modelling approaches in relation to their merits for design from the point of view of usability. As systems are designed in order to enable task delegation by users to systems, we will base our categorisation on this phenomenon. In delegating tasks to a computer four important aspects are of importance.

(a) *External tasks*: Users have to perform tasks existing in a task domain outside the computer, which have to be rephrased in terms of the tasks that can be delegated to the computer.
(b) *User knowledge*: In delegating tasks, users need knowledge about the computer system, about the objects and operations the system knows about, and how to operate these in terms of physical actions.
(c) *User performance*: User performance is concerned with the users' behaviour in delegating tasks to the computer system. Users must perform certain actions, both mental, perceptual and physical actions.
(d) *The computer system*: The system is the actual tool for task delegation, and, as a side effect, a main source for the user's knowledge of the interaction.

In accordance with these four aspects of task delegation, formal modelling techniques will be distinguished into four categories, each with its own specific purpose: task environment analysis, user knowledge analysis, user performance prediction, and representation for design purposes. The assignment of models to particular categories is not mutually exclusive. A formal representation can (and sometimes will) be used for different purposes, but assigning models to the category for which they were primarily developed will enable a fair judgement of their advantages as well as their restrictions.

2.4.1 Models for Task Environment Analysis

With task environment analysis we apply a modelling technique that focuses on the characteristics of how to execute tasks in a certain task domain, and related knowledge of this task domain. The single example we show in this category is "External Internal Task Mapping" (ETIT, Moran, 1983).

2.4.1.1 ETIT

Moran's External Internal Task Mapping Analysis is meant to analyse the relations between the external task domain (which refers to the tasks a user sets himself, or are set for a user, in relation to everyday reality) and the internal task domain (representing the delegation of suitable tasks to a computer system designed for application in the external task domain). Figure 2 contains an example of text manipulation as a user's task, related to text manipulation using a simple editor on a computer (for a description of the editor, see Moran, 1983).

EXTERNAL TASK SPACE:
Terms: Character, Word, Sentence, Line, Paragraph (Text)
Tasks: Add, Remove, Transpose, Move, Copy, Split, Join
Excuded: Copy, Split and Join Characters

INTERNAL TASK SPACE:
Terms: String
Tasks: Insert, Cut, Paste

MAPPING RULES:
1. Split, Join Sentences -> Change String
2. Text -> String
3. Add -> Insert
4. Remove -> Cut
5. Transpose -> Move
6. Split -> Insert
7. Join -> Cut
8. Change String -> Cut String + Insert String
9. Move String -> Cut String + Paste String
10. Copy String -> Cut String + Paste String + Paste String

Figure 2. External-Internal Task Mapping Analysis.

The example shows the entities and the operations or tasks involved within the two contexts. In the external task space, the 'known world', there are object types, like characters and lines, and there are tasks, like adding, moving, removing. The example editor, however, only knows a single object type "string" as an entity that can be "inserted", "cut", or "pasted". In the analysis, several task-object combinations are excluded, because they do not make sense, such as an operation to split characters. Exempting these irrelevant operations, a number of mapping rules can be determined which state how to translate a particular task from one environment to the other. The task "copy a sentence" in the external world can be mapped on a task delegated to the editor, "copy a string", which in its turn, must be rewritten as a combination of the actions "cutting a string", "inserting the string back in its original location", and "inserting it elsewhere". According to Moran, establishing the mapping between the objects and operations of the external and the internal tasks will make it possible to make inferences about the functionality, learnability and consistency of the user interface. ETIT should also be applicable in assessing the extent in which transfer of knowledge will occur between different user interfaces. Although ETIT is mentioned in the literature many times, we presently do not know whether ETIT has ever been applied to real systems.

2.4.2 Models to Analyse User Knowledge

The modelling techniques in this category employ a formal grammar to analyse and represent the knowledge the user needs to operate a user interface. This type of model may be used to compare the usability of different interfaces or different design options, and to predict differences in learnability.

More specifically, these techniques describe and analyse the knowledge the user must have in order to translate his tasks (originally represented by a user at semantic or conceptual level) into the appropriate physical actions required to operate the system. We mention two modelling techniques in this category, Reisner's Action Language (Reisner, 1981, 1983, 1984) and Task-Action Grammar (TAG, Payne, 1984; Payne and Green, 1986). Both techniques use a formal grammar to describe the task-action mappings, and both assess usability aspects by

counting the number of rules, the depth of the derivation of rules and the number of exceptional rules. They differ with regard to the formal grammar they use.

2.4.2.1 Action Language

Reisner's Action Language represents the task-action mappings in a notation called Backus-Naur Form (BNF), named after two of its authors (Backus et al., 1964). BNF is a formal notation to describe phrase structure grammars by means of a number of hierarchically organised rules. BNF is well known in computer science, where it is used, among other purposes, to describe what the legal or grammatically correct expressions are in programming languages like Algol (Backus et al., 1964) and Pascal (Jensen and Wirth, 1974). In BNF, each rule specifies the relation between the more abstract term on the left-hand side and the more specific terms on the right-hand side by means of the "is-defined-as" operator (::=). Alternatives are indicated by the "or" symbol (|), and succession by the "sequence" symbol (+). To reduce the size and increase the clarity of grammars, it is common practise to use various extensions of BNF, in which options are enclosed in square brackets ([...]) and repetition is indicated by braces ({...}). BNF is a notation for context-free grammars which means that terms on the left-hand side are uniformly rewritten on the right-hand side independent of other terms and rules. As such it is not possible to indicate that, for instance, the form of a verb in English depends on whether the subject of the sentence is singular or plural.

In cognitive ergonomics BNF can be used to describe the legal sentences in the communication language the user has to use to delegate tasks to the computer system. In this way it models what a user has to know. Reisner (1983) extended BNF to include cognitive actions, written in angle brackets (< >), and physically observable actions, written in capital characters. Figure 3 shows a fragment of Reisner's Action Language or psychological BNF.
The first line in the example shows that the issuing of the command "Dn" (to delete n lines) consists of a cognitive action (to retrieve the correct syntax of the command), followed by a plain nonterminal (referring to how the syntax information is used and which keystrokes are involved). Although retrieving the needed information is a cognitive activity, and using it a physical activity, both parts are rewritten in the same way.

```
employ Dn                       ::= <retrieve info. on Dn syntax>
                                    + use Dn

<retrieve info. on Dn syntax>   ::= <retrieve from memory>
                                    | <retrieve from external source>
<retrieve from human memory>    ::= <RETRIEVE FROM LONG TERM MEMORY>
                                    | <RETRIEVE FROM SHORT TERM MEMORY>
                                    | <USE MUSCLE MEMORY>

retrieve from external source   ::= RETRIEVE FROM BOOK
                                    | ASK SOMEONE
                                    | EXPERIMENT
                                    | USE ON-LINE HELP

use Dn                          ::= identify first line
                                    + enter Dn command
                                    + PRESS ENTER
```

identify first line ::= ...

enter Dn command ::= TYPE D + type n
...

Figure 3. Action Language (Reisner, 1983).

Reisner's action language has not been extensively used; Richards et al. (1986) used it to specify a graphical operating system shell (MINICON). The only other application we know of stems from Reisner's own work on the precognitive action language (Reisner, 1981), in which two versions of a drawing program are compared, one that does, and one that does not treat all the data objects in a uniform way. In this study, the non-uniform interface was characterised by the presence of additional rules to describe the exceptions, and as it was predicted, this interface turned out to be more difficult to learn and use than the interface which needed fewer rules to be completely described.

Reisner's work has indicated that BNF can be used to describe the knowledge the user needs to operate a computer system. However, in terms of the strength of expression, more powerful grammars can be, and are, used. Shneiderman (1982b) introduced the idea of using a "multi-party BNF" for representing the interaction decomposition regarding both "partners" in human-computer interaction (see Innocent et al., 1988, for an elaboration of this concept). These formalisms, again, have not yet been elaborated for real life situations. But BNF-like grammars of this type are still restricted to representation of sets of single rules.

A further development of BNF, van Wijngaarden grammars (van Wijngaarden, Mailloux, Peck, and Koster, 1969), provides a formal representation technique for structured grammars that include the use of two levels of production rules. Payne and Green (1983) show that set grammars (related to van Wijngaarden two-level grammars) enable the representation of "family resemblances" among rules. Only this new type of representation could account for the perception of consistency and inconsistency in syntax constructions, and, hence, could be used as a better model of a user's perception of an interaction language. This analysis lead to the development of TAG.

2.4.2.2 TAG

Task-Action Grammar (Payne, 1984; Payne and Green, 1986) employs a more sophisticated semantic feature grammar than Reisner's BNF. "Simple tasks" in TAG are represented by rules which can be rewritten in the same way as BNF rules, but in addition, TAG contains features which make it possible to describe tasks in terms of the meaning they may have for the user. In technical terms this means that it is possible to have rules describing the structure of sets of rules, which is not possible in the original versions of BNF. In terms of the user this means that tasks such as "moving the cursor to the left" or "moving the cursor to the right" are identical except for the indication of the direction. In figure 4 cursor movement is used to illustrate Task-Action Grammar. The commands are listed in the "Dictionary of simple tasks", from which a simple "Rule schema" is derived that illustrates the consistency of the syntax of the example. The user needs only knowledge of one general rule and of the "features" Direction and Unit.

Green et al. (1988) have applied Task-Action Grammar to describe and explain the results of various experiments on command languages. In one experiment, subjects had to learn and use three applications, with command languages that were grammatically similar or different between applications. Learnability predictions were established for various formal modelling techniques, and for several design guidelines. A comparison between the predicted and the actual results showed TAG's predictions to be most accurate. Finally, Green et al. (1988) also applied TAG to describe several commercially available software packages, from which the general conclusion is drawn that extensions are needed when TAG is used for other purposes than the analysis of command language consistency.

List of commands

move cursor one character forward ctrl-C
move cursor one character backward meta-C
move cursor one word forward ctrl-W
move cursor one word backward meta-W

List of features	*Possible Values*
Direction	forward, backward
Unit	character, word

Dictionary of simple tasks

move cursor one character forward
{ Direction = forward, Unit = char }
move cursor one character backward
{ Direction = backward, Unit = char }
move cursor one word forward
{ Direction = forward, Unit = word }
move cursor one word backward
{ Direction = backward, Unit = word }

Rule Schemas

Task [Direction, Unit] -> symbol [Direction] + letter [Unit]
symbol [Direction = forward] -> "ctrl"
symbol [Direction = backward] -> "meta"
letter [Unit = character] -> "C"
letter [Unit = word] -> "W"

Figure 4. Task-Action Grammar (Green, Schiele, and Payne, 1988).

2.4.3 Models of User Performance

Methods for user performance predictions are the modelling techniques primarily targeted at analysing, describing, and predicting user behaviour and time needed to get tasks done while using a particular computer system. Two often cited modelling approaches in this category are the GOMS model (Goals, Operators, Methods and Selection Rules) of Card, Moran and Newell (1983) and the Cognitive Complexity Theory (CCT) of Kieras and Polson (1985).

Internally, the models used in this category are not very different from the models used to analyse the knowledge of the user, except for the fact that these models have a formal production-rule (if ... then ...) representation instead of a formal grammar. They do, however, differ with respect to the purpose of application. Whereas modelling techniques to analyse

user knowledge describe and analyse what a user should or must know (without specifying how a user should apply this knowledge), the techniques to predict user performance describe and analyse what a user should know and, additionally, what a user should actually do, in order to attain task goals.
As such, the GOMS and the CCT models are performance models, whereas the Action Language and the TAG representation are competence models. This difference may be illustrated for the case when the user may choose from alternative methods. In Reisner's Action Language and TAG the choice from alternatives is just described and specified by the "or" (|) symbol, as a complete list of different possibilities, without indicating any conditions for actual choices. In GOMS and CCT, however, the goal is to predict user performance, and consequently the conditions for a user to choose an option must be specified in advance, e.g., by inferring individual users' strategies from observation.

The most serious implication from this is that GOMS and CCT require a complete specification of the task goal hierarchy of the user. Another consequence of this choice is, that GOMS and CCT implicitly claim that they can formally represent much more than Action Language and TAG claim, namely actual behaviour, instead of only knowledge as a basis of behaviour. The task goal hierarchy which is needed for a GOMS or CCT analysis is in both cases a GOMS representation, or a hierarchical specification of the users' goals, operators, methods and selection rules.

2.4.3.1 GOMS
Figure 5 presents an example of a GOMS representation of part of a text editing task.

```
GOAL: Edit-Manuscript
. GOAL: Edit-Unit-Task-until no more unit tasks
. . GOAL: Acquire-Unit-Task
. . . Get-Next-Page-if at end of page
. . . Get-Next-Task
. . GOAL: Execute-Unit-Task
. . . GOAL: Locate-Line
. . . . [select:     Use-String-Search-Method
. . . .              Use-Linefeed-Method]
. . . GOAL: Modify-Text
. . . . [select:     Use-Delete-Word-1-Method
. . . .              Use-Delete-Word-2-Method
. . . .              ...]
. . . . Verify-Edit
```

Figure 5. GOMS top level of an editing task (Card, Moran and Newell, 1983).

As can be seen in the example, goals exist at several different levels of a task. A "general goal" like editing a manuscript is initially subdivided into "unit task goals", which correspond to the tasks the user knows how to perform. In general, "unit tasks" correspond to the commands of a computer system, such as deleting a word, transposing two words, etc. in case of an editor. Unit task goals are further subdivided into a number of levels of "subgoals", until they can be resolved by applying "operators" or the "elementary perceptual, motor, or cognitive acts", such as pressing a key, inspecting the screen or acquiring the next unit task.

As a matter of fact, GOMS forms a family of models, because the level at which operators are defined may vary, and this level defines the granularity of a GOMS model.

Methods like using a string search or repeatedly pressing "linefeed" to get the cursor in position are collections of operators. If there is more than one method to reach a goal, then selection rules determine which method will be used. For example, if the target position of the cursor is on the screen, the linefeed method is used, otherwise, the string search method is used. The time predictions GOMS generates depend on the level at which the operators are defined. In general, the predictions are based on the summation of the times needed to execute the elementary actions of the model, which include physical acts (pressing a key), perceptual acts (locating the cursor), and cognitive acts (making a selection).

Operator	Description and remarks	Time (sec)
K	PRESS KEY OR BUTTON Pressing the SHIFT or CONTROL key counts as a separate K operation. Time varies with the typing skill of the user; the following shows the range of typical values:	
	Best typist (135 wpm)	0.08
	Good typist (90 wpm)	0.12
	Average skilled typist (55 wpm)	0.20
	Average non-secretary typist (40 wpm)	0.28
	Typing random letters	0.50
	Typing complex codes	0.75
	Worst typist (unfamiliar with keyboard)	1.20
P	POINT WITH MOUSE TO TARGET ON DISPLAY The time to point varies with distance and target size according to Fitt's Law, ranging from .8 to 1.5 sec, with 1.1 being an average. This operator does not include the (0.2 sec) button press that often follows. Mouse pointing time is also a good estimate for other efficient analogue pointing devices, such as joysticks.	1.10
H	HOME HAND(S) ON KEYBOARD ON OTHER DEVICE	0.40
$D(n_D, l_D)$	DRAW n_D STRAIGHT-LINE SEGMENTS OF TOTAL LENGTH l_D CM. This is a very restricted operator; it is assumed that drawing is done with a mouse on a system that constraints all lines to fall in a square .56 grid. Users vary in their drawing skill; the time given is an average value.	$0.9n_D+0.1l_D$
M	MENTALLY PREPARE	1.35
R(t)	RESPONSE BY SYSTEM Different commands require different response times. The response time is counted only if it causes the user to wait.	t

Method for Task T1-BRAVO:

Reach for mouse	H[mouse]
Point to word	P[word]
Select word	K[yellow]
Home on keyboard	H[keyboard]
Issue Replace command	MK[R]

Type new 5-letter word	5K[word]
Terminate type-in	MK[esc]

$T_{execute} = 2t_M + 8t_K = 2t_H + t_P = 6.2$ sec.

Figure 5. Keystroke model (Card, Moran and Newell, 1983).

A well known member of the GOMS family of models is the Keystroke Level model. This model, however, lacks the analysis purpose of GOMS itself and is purely meant for predicting error free, expert performance times. In the Keystroke Level model the user's tasks are analysed at the level of unit tasks. The time to perform each unit task is estimated by adding the time to acquire the unit tasks, the time to execute the keystrokes in the associated commands, and the time needed for mental operators, which are inserted into the command sequences according to sets of heuristic rules. Figure 6 shows an example of keystroke level analysis. General time parameters are estimated for different actions like "press key or button", "point with mouse", and "mentally prepare". The unit task illustrated requires the performance of a sequence of actions ("reach for mouse", "point to word", "select word" etc.), for which the corresponding time parameters are added to estimate the total execution time.
GOMS is probably the most cited formal model in human-computer interaction, even though GOMS is meant to be applied under rather restrictive conditions and for a rather limited purpose.

(a) A performance model, GOMS is restricted to predicting error free performance. In GOMS the cognitive load of a user is assessed by counting the number of active goals in memory. Lerch, Mantei and Olson (1989) used GOMS based estimates of mental overload to predict error behaviour, which they showed to be valid for simple ("overload") errors, although GOMS still is unable to predict conceptual errors. This is a serious restriction. Roberts and Moran (1983) report that experts spend between 4 and 22 per cent of their time in correcting (only) serious errors (non-experts would struggle with errors even more often). Although Card et al. (1983) mention that it should be possible to apply GOMS to include error repair and non expert performance, this has, to our knowledge, not been investigated in their studies.

(b) The model generates reasonably good predictions under rather specific conditions only. Card, Moran and Newell (1983) base their predictions on GOMS analyses adapted to individual subjects, for instance, to account for differences in the criteria of selection rules. In the validation studies of the GOMS model, the subjects had to make changes to manuscripts from annotations, which is a rather limited task domain. Lerch et al. (1989) applied their GOMS analysis on a restricted set of tasks (financial calculations) for two commercially available spreadsheet systems.

(c) The GOMS analysis depends very much on the definition of unit tasks, but "task the user knows how to perform" is not a precise definition, so that the analist may have to rely on his own intuition in dividing the task into unit tasks (Wilson et al., 1988). In order to resolve this problem, van der Veer (1990) defines a unit task as "elementary primary task that may not be decomposed into other primary tasks", where "a primary task", in turn, is defined as "a task the user wishes to perform, independent of the specific characteristics of the tools he will use".

Although there are serious criticisms about how GOMS is applied to predict user behaviour, it is one of the most widely investigated and extended models (John and Kieras, 1994), and the value of GOMS as a heuristic method to gain insight in the users' tasks must not be underestimated. A GOMS representation is a very useful and systematic tool to describe the structure of decomposing the user's tasks in smaller elements. For structural description, a GOMS representation is much more useful than models of user knowledge or models for task environment analysis.

2.4.3.2 CCT

Another model to predict user performance is Cognitive Complexity Theory from Kieras and Polson (1985). CCT is primarily an implementation of the GOMS model in terms of an explicit production system; that is, Kieras (1988) has published a set of rules to rewrite the implicit if-goal then-action notation of a GOMS model into a real production system notation. Such a system is called a " job-task analysis" which is regarded as a description of the process going on in the users' working memory. In this representation multiple conditions can be taken together by the AND operator, while the action part may consist of more than one action, including cognitive actions, such as the creation of (new) subgoals. An example of a fragment of a CCT analysis is shown in figure 7, which represents a particular way to delete words, when using IBM's Displaywriter (Kieras and Polson, 1985, pp. 374). The first production shows that the condition (goal is to delete a word, the goal position of cursor movement is not identical with the current position of the cursor) leads to the addition of the subgoal to move the cursor to the goal position. The second production shows that the condition (goal is to delete a word, cursor is at the goal position) leads to the sequence of actual actions that imply deletion and to the removal of the goal once this is fulfilled.

Method to delete a single word

```
(PDELW1
        IF (AND (TEST-GOAL delete word)
                (NOT (TEST-GOAL move cursor to %UT-HP %UT-VP))
                (NOT (TEST-CURSOR %UT-HP %UT-VP)))
        THEN (  (ADD-GOAL move cursor to %UT-HP %UT-VP)))

(PDELW2
        IF (AND (TEST-GOAL delete-word)
                (TEST-CURSOR %UT-HP %UT-VP))
        THEN (  (DO-KEYSTROKE DEL)
                (DO-KEYSTROKE SPACE)
                (DO-KEYSTROKE ENTER)
                (WAIT)
                (DELETE-GOAL delete-word)
                (UNBIND %UT-HP %UT-VP)))
```

Figure 7. Cognitive Complexity Theory (Kieras and Polson, 1985).

The aim of this representation is threefold:

(a) By estimating the time needed to execute productions, and, in particular, their operator parts, time predictions can be generated, based on the actual tasks and the task-job representation.

(b) Analysing what the user has to do to operate a certain system in terms of production rules provides a uniform way to compare computer systems. When two computer systems have comparable functionality, the system that requires most production rules will be more difficult to learn and use. Kieras and Polson have extended this point by stating that the ease of learning a new system will depend on the number of common rules between the new and the known system. Transfer of learning would only depend on the number of common rules, irrespective of, for example confusion created by seeming commonality.

(c) The production system representation can be used to analyse task-to-device mapping, representing the difference between the task-goal hierarchy of the user (how to do it) and the state transition of the system (how it works). Although, presumably no one will deny that performance is best when the user's expectations coincide with the behaviour of the system, Kieras and Polson have treated this subject too scantly to say anything conclusive about it.

CCT may be used to compare systems in more or less the same way as the GOMS model and the models to analyse user knowledge or task environment. Because CCT uses the GOMS model, the same criticisms apply: the model has difficulty to cope with errors (Vossen, Sitter and Ziegler, 1987), so that it may only be applied to routine expert tasks. According to Knowles (1988), CCT, by virtue of its reliance on GOMS is restricted in application to tasks involving no problem solving, besides that CCT relies on the quantitative aspects of representing knowledge at the expense of qualitative aspects. At present, CCT has been used mainly to analyse transfer of training effects both successfully (Polson and Kieras, 1985; Foltz, Davies and Polson, 1988; Polson, 1988) and less successfully (Vossen, Sitter and Ziegler, 1987).

Critics state that CCT is not very clear about what actually constitutes a single production: "Production rules can be rewritten in many different forms, thereby affecting the apparent complexity in terms of number of rules, number of times each one is used, etc." (Green, Schiele and Payne, 1988).

2.4.4 Models of the User Interface

Models in this category are developed in relation to formal techniques for design specification. The models in this group aim at providing a complete and full representation of the "virtual machine" (the computer system as seen from the point of view of a fully competent user). These models represent aspects of a computer system that are relevant for both the potential user and the designer, at the different levels of abstraction of human-computer interaction.

This category is exemplified by the Command Language Grammar (CLG; Moran, 1981) and by Extended Task-Action Grammar (ETAG; Tauber, 1988, 1990). Both are methods to describe the hierarchical structure of a user interface at, except for a few minor details, the

same levels of abstraction mentioned before, in a related way. Apart from details such as naming, ETAG and CLG differ with respect to the formalism they use, and in some choices related to the organisation and the main points of the representation.

2.4.4.1 CLG

Starting with a formal description of the tasks and the associated task-entities, and finally ending with the specification of the physical actions, the user interface can be precisely described for design purposes in a top-down manner. Moran (1981) partitions the communication between man and machine into three components, each containing two levels. Each of the six levels is a complete description of the computer system at its level of abstraction:

Conceptual Component:	Task Level
	Semantic Level
Communication Component:	Syntactic Level
	Interaction Level
Physical Component:	Spatial Layout Level
	Device Level

Moran (1981) only discusses the first four levels and leaves the other two for future elaboration (and to "classical" ergonomics). The distinction into three components clearly indicates the major concern at each pair of levels (see van der Veer, 1990), but this is not strictly needed to understand CLG, and it will not be discussed here.

The distinction into six levels is directly related to considerations of good user interface design, based on the user's mental model. That is, the user comes to the computer system to get tasks done, and in order to do that, the tasks of the user have to be rephrased into the task language of the computer system and finally specified by physical actions of the user. The other way around, in order to understand the system, the user has to perceive the physical signals of the system, code them into meaningful symbol structures, and rephrase the responses of the system in terms of his primary tasks. Each of Moran's levels describes at a particular level of abstraction an aspect of this process, in terms of what has to be translated and how it is to be done. In this way, the output of each level is a further refinement of a previous level, or, in the opposite direction, an abstraction of the next level.

The purpose of the representation at the task level is to analyse the user's needs and to structure the task domain in such a way, that a computer system can play a part in it. The task level describes the structure of the tasks which can be delegated to the computer system. In order to use an interactive system, the user has to translate his tasks into operations the computer knows about.
The representation at the semantic level describes the set of objects, attributes, and operations, the system and the user can communicate about for the purpose of task delegation: for the system as data structures and procedures, and for the user as conceptual entities and operations on them.
The syntactic level describes which conceptual entities and operations may be referred to in a particular command context or system state, and how that is done, in terms of linguistic

aspects (references to commands and objects, including the lexicon) and lexicographic aspects (the order of referencing, display area's). At this level it is specified, for instance, that there is a window to position delegation commands, and that the command to delete is "delete" followed by the type of arguments to delete and a list of arguments. Ultimately, the communication between man and computer is a matter of physical actions, such as sequences of key presses, movements of the mouse, meaning signals like the "beep" etc.
The interaction level describes the translation of the reference names of commands and objects into the associated physical actions and the structure of the interaction, including typing rules and mouse manipulation conventions and the reactions and prompts from the system.
In Moran (1981), this is also the level where the treatment of CLG ends, but he adds that a full CLG analysis would also include a specification at the spatial layout level, and one at the device level. The former level describes the arrangements of the input and output devices, including display graphics, while the device level would describe all the remaining physical features. Figure 8 presents a fragment of an electronic mail tool, described at the four highest levels of interaction. At task level a description is shown of the task to read new messages (if any) and of the objects SEND-MESSAGE (which indicates a new message) and MESSAGE (which indicate any message, whether old or new). One of the constituents of the above mentioned task (check for new mail) is subsequently represented at semantic, syntax, and key-stroke level.

```
NEW-MAIL = (A TASK (* Check for new SEND-MESSAGEs,
                   if any, read them)
                 DO (SEQ:  (CHECK-FOR-NEW-MAIL)
                           (READ-NEW-MAIL)))

SEND-MESSAGE = (AN ENTITY
               NAME = "Send-message"
               (* comments ... ))

MESSAGE = (AN ENTITY
                REPRESENTS (A SEND MESSAGE)
                NAME = "Message"
                AGE = (ONE-OF: OLD NEW)
                (* comments ... ))

SEM-M2 = (A SEMANTIC-METHOD
               FOR CHECK-FOR-NEW-MAIL
               DO (SEQ: (START EG-SYSTEM)
                   (SHOW DIRECTORY)
                   (LOOK AT DIRECTORY FOR
                (A MESSAGE AGE = NEW))))

SYN-M2 = (A SYNTACTIC-METHOD
              FOR CHECK-FOR-NEW-MAIL
              DO (ENTER-EG-IF-NEW-MAIL))

IA-M2   =  (AN INTERACTION-METHOD
              FOR CHECK-FOR-NEW-MAIL
              DO (KEY: "EG/N" RETURN))
```

Figure 8. Command Language Grammar (Moran, 1981).

According to Moran (1981), three different points of view apply to CLG:

(a) The psychological view applies CLG as a model of an "ideal" user's knowledge that shows the different kinds of knowledge that users have about systems. Moran, however, does not comment on the psychological validity of CLG as a model in this respect.
(b) The linguistic view uses CLG as a description of the structure of command language systems, which may be used to generate all possible "command languages". It should be noted that at the time of publication of CLG there was no uniform nomenclature of interaction styles (and, indeed, some currently well known styles were not generally available), but CLG's claim is in principle valid for the description of all types of interaction mode.
(c) The design point of view applies CLG as a representation tool for specifying the system during the (top-down) design process to help the designer generate and evaluate alternative designs for the system.

However, only the third view of CLG as a description of the conceptual model is really worked out, and most prevalent. To this might be added that there are more powerful, or less cumbersome, grammars to describe the linguistic structure of an interaction language. Furthermore, Moran leaves us with only a number of suggestions about how a CLG representation might be analysed to predict or evaluate aspects of the systems' usability, such as performance times, memory load and learning. Sharratt (1987) presents some results of using CLG as a specification tool in a practical design exercise, in which he asked students to use CLG to specify a design for a transport time-table system. Sharratt concludes that CLG is useful for design specifications, but that it carries many of the drawbacks of a strictly top-down design process and leaves little room for design iterations. Furthermore, CLG cannot be used to describe the relation between the tasks and the information on the screen. CLG, however, seems to provide a valid framework to model (competent) users' knowledge - at least from the point of view of the system designer (van der Veer, 1990).

2.4.4.2 ETAG

ETAG or Extended Task-Action Grammar (Tauber, 1988, 1990) is in many ways comparable with Moran's CLG. Both are techniques to describe the human computer interface from the point of view of the user (the "virtual machine"), both employ the notion of levels of abstraction in the interaction, and both use formalisms to specify the contents of, and the mapping between these levels. Tauber (1988) used Task-Action Grammar (Payne, 1984) to describe how users have to rephrase their tasks in terms of lower level rules, until arriving at the physical actions submitted to the computer system. Tauber prefers to use the concept of "basic tasks" (different from Payne's "simple tasks"), defined as "tasks for which the system provides a single command or equivalent unit of delegation" (Tauber, 1988). The system's basic tasks should be distinguished from the user's "unit tasks", defined as "elementary primary tasks that may not be decomposed into other primary tasks" (van der Veer, 1990).

Whereas TAG only provides levels for purely notational reasons, ETAG uses CLG's well-chosen levels of abstraction, adding some refinements. This is done, because ETAG is

also aimed at formally specifying the "user's virtual machine", including the task related semantics of the computer system. The user's virtual machine (UVM) is defined by means of a canonical basis, an ontology borrowed from Psycho-linguistics (Jackendoff, 1983, 1985). Basically, the ontological or canonical basis describes the world in terms of concepts (such as: objects, places and states), attributes, relations between objects, functions of objects (such as: object being at places, e.g., on top of others), and events which change existence, functions, and relations (such as killing, moving and copying objects).

Part of a canonical basis for a UVM

```
[CONCEPT]     ::= [OBJECT] | [VALUE] | [PLACE] | [STATE] | [EVENT]

[PLACE]       ::= [place.IN ([OBJECT])] | [place.ON ([OBJECT])]
                | [place.ON-POS ([OBJECT])] | [place.ON-TOP ([OBJECT])]
                | [place.ON-TAIL ([OBJECT])]

[STATE]       ::= [state.IS-AT ([OBJECT], [PLACE])] |
                | [state.HAS-VAL ([OBJECT], <ATTRIBUTE>, [VALUE])]

[EVENT]       ::= [event.KILL-ON ([OBJECT], [PLACE])]
                | [event.MOVE-TO ([OBJECT], [PLACE])] |  ....

type[EVENT > event.MOVE-TO ([OBJECT: *o], [PLACE: *p]) ]
    precondition:      [state.IS-AT ([OBJECT: *o], [PLACE: *p0])] ;
    clears:            [state.IS-AT ([OBJECT: *o], [PLACE: *p0])] ;
    postcondition:     [state.IS-AT ([OBJECT: *o], [PLACE: *p])] ;
end [EVENT]
```

A conceptual object and a conceptual event of a UVM

```
type [OBJECT > MESSAGE]
    supertype:         [TEXT] ;
    themes:            [HEADER], [BODY] ;
    relations:         [place.ON-POS(1) ([MESSAGE])] for [HEADER],
                       [place.ON-POS(1) ([MESSAGE])] for [BODY],
                       [place.POSS-AT ([MESSAGE])] for [HEADER], [BODY] ;
    attributes:        <SENDER>, <SENDING_DATE>, <RECEIVING_DATE>,
                       <STATUS>, <DELETION_MARK> ;
end [MESSAGE]

type [EVENT > COPY_MESSAGES]
    description:       for {[MESSAGE: *x]}
                               [event.COPY-TO ([MESSAGE: *x],
                               [place.ON-TAIL ([MESSAGE_FILE: *y]): *p2])] ;
    precondition:      [state.IS-AT ([MESSAGE: *x],
                       [place.ON-POS.(i) ([MESSAGE_FILE: *z]): *p1])] ;
    comments:          "copy messages x from file z onto the end of file y" ;
end [COPY_MESSAGES].
```

A basic task from the dictionary

```
ENTRY 6:
        [TASK > COPY_MESSAGES],
        [EVENT > COPY_MESSAGES],
```

[MESSAGE_FILE: *z],
T6[EVENT > COPY_MESSAGES]
[OBJECT > MESSAGE: (*x)][OBJECT > MESSAGE_FILE: *y],
"copy messages from the current message file into a message file".

Figure 8. Extended Task-Action Grammar (Tauber, 1990).

The canonical basis indicates relevance for the user, and should be part of the user's virtual machine. In terms of Norman (1983) the UVM is the conceptual model of the target machine and, as such, equivalent to a competent user's mental model. In terms of Kieras and Polson (1985) the UVM describes the "how it works" knowledge the user needs. An example of a conceptual event in an electronic mail system is to "mark for deletion", which sets the attribute "deletion mark" for an object "message" that resides at a place in "message file".

The next level in ETAG consists of the dictionary of the basic tasks. This level lists which basic tasks are possible, and how they relate to the concepts of the UVM. Figure 8 gives an example of the higher levels of an interface specification in ETAG. Fragments of the UVM of an electronic mail system are illustrated including part of the "canonical basis (a concept hierarchy) and a description of an object (a message). An entry of the dictionary of basic tasks shows the formal description of the semantics of a basic task (copy a message).

The dictionary of basic tasks corresponds to the top level of the production rules. The production rules use the feature grammar of Task-Action Grammar to describe how to perform the basic tasks in terms of still lower levels, until the commands for the computer system are fully specified. ETAG employs a refinement of the levels of CLG to structure the process of derivation by introducing levels to specify the syntax, the referencing style (e.g. pointing versus naming), the lexicon and the keystrokes, respectively.

ETAG, although originally designed as a modelling tool for user interface design, has already been applied for modelling user performance and user knowledge (van der Veer, Yap, Broos, Donau, & Fokke, 1990).

2.5 An Evaluation of Formal Modelling Techniques

Formal modelling techniques as analysed in the previous section, are representation methods to specify aspects of human-computer interaction. Models represent what a user has to know or to do in order to accomplish tasks by means of a computer system. Models can be used to analyse the similarities and differences in the way tasks are to be done with and without a computer system, to evaluate usability characteristics of human-computer interfaces, to predict certain aspects of user behaviour, and to formalise the hierarchical design of user interfaces in terms of the knowledge of the user at multiple levels of abstraction.

At this point, one could ask to what extent the HCI formal modelling techniques can be used successfully for these purposes, and if not completely successful, what the main problem areas are in using these techniques. To answer this question, it is necessary first, to determine the special requirements the modelling techniques should fulfil to specify user knowledge for the aforementioned purposes. We explicitly mention "special" requirements, because we are

concerned with formal models for specification purposes, namely, of user knowledge for the purpose of task-environment analysis, knowledge analysis, performance prediction, and representation for design.

In the past, others have suggested various requirements, some of which have been adapted, either completely or partially, and some have been rejected. For example, from Green, Schiele and Payne (1988) we accepted the requirement that formal models should be usable for designers. However, when they write that "The model must contain a representation of the external semantics." (pp. 38), they notice an important problem area of the models that they (and we) discuss, but not an overall requirement. Also, we do not think that it is necessary that "The model must describe a reasonably complete psychology." (pp. 38), as long as the resulting inferences are valid and useful for the desired purpose.

What we consider to be important is that formal models are tools to represent user knowledge for the purpose of computer system design. In the introduction, Norman's (1983) notion of models in human-computer interaction was discussed to stress the point that the design of a system should be based on a conceptual model, derived from an analysis of the intended users, and the users' tasks, in order to attain mutual consistency between the system image, the users' mental model, training materials, and the conceptual model. From this basis, four requirements for representation techniques for design purposes can be put forward.

(a) A conceptual model should be both based on the point of view of the user, and provide a complete and accurate representation of the design. Hence, formal models should provide a complete description of the intended system, at the different levels of abstraction.
(b) Representation tools should have a wide applicability. In computer systems design, a modelling technique is of little use when one has to resort to another technique, for example, just when the style of interaction is to change from one to another. Therefore, formal modelling techniques should be applicable to a variety of different kinds of users, styles of interaction, and types of tasks.
(c) Formal models are used as conceptual representations of computer systems to analyse and predict usability aspects, about knowledge, and about performance. Inferences from analysis and prediction are useful to the extent that they are valid. The same applies to the modelling techniques on which the inferences are based.
(d) Just like computer systems are mere tools for their users, formal modelling techniques themselves are tools to perform the tasks of their own users: the designers. Therefore, within the broader context of design, formal modelling techniques themselves should fulfil the requirements of being functional, easy to use, and easy to learn and remember.

The requirements we have selected apply foremost to the modelling techniques for design specifications, but only because the design specifications are proposed with the most general intentions. CLG, with its three views, for example is intended for design specification, but also for user performance prediction and knowledge analysis. On the other hand, the requirements also apply to the techniques with more limited aspirations. For example, to analyse user knowledge, the representation should be a valid one, and include all the relevant aspects of the user's knowledge in a variety of circumstances and be usable for the analyst.

One may imagine, however, that more specific purposes demand a different relative weighing of the requirements.

2.5.1 Completeness

The requirement of completeness means that a formal modelling technique should enable a complete specification of the user interface at all the levels of abstraction involved in using and in designing the interface. The modelling techniques reviewed in this chapter fulfil this requirement to a smaller or larger extent, but none completely cover the whole interface. According to Green (1990) completeness is not a key requirement, and it may be better to have a number of so-called "limited theories", each of which covers its domain of application well, than to have a few large theories which cover more questions but each question only to a limited extent. As such, it may be possible to employ Cognitive Complexity Theory to predict performance times, Task Action Grammar to evaluate the consistency of an interface and Command Language Grammar to specify the interface for design. Contrary to Green's (1990) view, it is argued that, in the practice of design, it is preferable to deal as few as possible different methods since otherwise usability problems may arise. Regarding completeness, the main omissions in the modelling techniques in HCI can be found at the highest and at the lowest levels of abstraction.

At the higher levels, an analysis of the user's concepts of the external tasks and those of the device are either omitted, mentioned without further specification, or the user's goal task hierarchy is much more rigidly specified than this will be the case in reality. For example, in Reisner's Action Language only attention is paid to the syntactic and lexical aspects of the users actions, and no attention is paid to the semantics of the interface. Payne and Green's TAG theory goes a step further, and lists the semantic features of tasks, such as that the so-called 'clipboard' is involved in a cut and paste action. However, they do not in any sense describe the nature of clipboards as a temporary storage place for data. Moran's CLG would contain a description of the clipboard, but it would probably need a comment to fully explain its nature. With respect to the formal methods in the review, only Tauber's ETAG is able to give a formal account of these semantic features because it uses an ontology.
Payne (1987, 1991) discusses the semantics of the interaction device being used in relation to the user's task entities and task strategies. Except for ETIT at a very basic level, none of the models is able to describe the relation between task entities and their e.g. graphical representations. Both GOMS and CCT are able to describe the mechanism of user strategies and, at least in practice, the TAG and Action Language notation might be used as well. CLG is able to describe user strategies even though it does not completely describe the task entities that are required. ETAG is better able to describe the task entities but, being based on basic tasks, strategies are not allowed. It is possible, however, to use ETAG's notation to describe user-level tasks and strategies by means of so-called menu tasks (chapter 6).

At the lower levels of abstraction generally a description is missing of the visual presentation of the interface, especially in relation with the state of the system. Regarding the TAG theory, Green, Schiele and Payne (1988) write that: "it [TAG] does not exhibit the relation between actions and system display; as far as TAG is concerned, the VDU screen could have been turned off" (pp. 30). Although Moran (1981) explicitly mentions a spatial layout level, it has

not been specified any further, and therefore none of the modelling approaches that have been discussed is able to address the presentation component.

2.5.2 Width of Applicability

The requirement of the width of applicability means that a formal modelling technique should be applicable to a variety of user-populations, types of tasks and ways of interacting with a computer. Here also, coverage is limited.
In the first place the knowledge or the performance of the user that is typically described refers either to the ideal user or to the competent user. The ideal user is taken to be one who has perfect knowledge and is only engaged in error-free and most efficient performance. On the other hand, the competent user is only perfectly knowing, but may commit performance errors. Even if the focus is on evaluating the interface as a whole, it is difficult to apply findings to real users, and especially to novice users, who may be characterised by their imperfect and even erroneous knowledge of the system (e.g. Briggs, 1990). This point is not really problematic, because it refers only to the inability to predict task performance by individuals, whereas we are more concerned about evaluation of design and prediction of task performance in general. As an exception, CCT has been applied to model performance of non-expert users.

Secondly, except for Moran's ETIT analysis all the models essentially employ a context free grammar or an equivalent method of representation, which means that performance on a given task is viewed independent of any other tasks and that only isolated tasks can be represented. The assumption that task performance is independent of for instance previous tasks does not seem to be in accordance with the reality of computer use.

The last point is the consequence of both the use of a context free grammar, and the lack of a specification of the visual presentation component. Presently, it is not possible to apply the modelling techniques successfully to model other tasks than those requiring little or no control or revision of planning by the user during execution. In reality however, users do control their tasks based on the knowledge of delegation of other tasks (both in parallel and in sequence), and on the perception and interpretation of information on the screen and other system responses. Rassmussen (1987) points out that because of this restriction, formal models may eventually only apply to the least important and least interesting bits in human-computer interaction. Additional or alternative modelling techniques are required to address the dynamics in user task performance, like the PUMS approach (programmable user models) which covers problem solving to some extent (Young, Green and Simon, 1989), or the Action Facilitation approach which is focussed on facilitating and inhibiting factors for task performance (Arnold and Roe, 1987; Roe, 1988).

2.5.3 Validity

The third requirement of formal modelling techniques concerns the validity of the analyses and predictions delivered. This requirement does not only apply to task environment, knowledge analysis, and performance prediction. The representations for design purposes also carry a notion of what constitutes a good design, by the implicit choice to include certain

features in the model as relevant and leave others out. Here, few problems can be mentioned; within the limited field of application for which the formal modelling techniques have been proposed, it has been shown that they indeed do what they are supposed to. To name a few, Reisner (1981) has shown that action language can be used to predict differences in ease of use between user interfaces. Card, Moran and Newell (1983) and Polson (1987) mention a number of experiments in which user performance was predicted reasonably well by their respective models. The experiments described by Lerch at al. (1989) show experimental application of formal modelling to knowledge of commercially available systems, where both execution times and certain types of errors were successfully predicted. Finally, Payne and Green (1989) describe an experiment in which Task-Action Grammar was successfully used to address subtle differences in the usability of interfaces, that other modelling techniques could not address.

There are several limitations connected to the aforementioned and other validation studies. The validation studies have generally used very simple user interfaces, they have almost always been performed by or under the supervision of the original authors of the method, and the studies have hardly ever taken place outside the research laboratory using full blown interfaces. Apart from establishing the utility of formal models by others than experts of the particular modelling technique, there is a need to establish the validity of the modelling techniques when used by non-experts, and a need to seek the limits of their applicability in real design.

2.5.4 Usability

A final requirement formal models have to satisfy is being of use in the practice of interface design. Here, the relevant points are to what extent formal models can be used in all stages of design, the adaptation to other techniques used by designers, and the usability aspects of applying formal models.

As was argued before, the approaches to formal modelling that have been discussed in this chapter do not cover all the aspects and levels of abstraction of user interfaces. The techniques can not completely specify the semantics of the computer system, the presentation component and other reactions of the system. Presumably, the most important factor in this is the lack of a specification of the semantics of the system, including the "how it works" knowledge, because when the conceptual model of a design is known then it may be easily used, e.g., to guide the choice for a particular screen layout or for the contents of an error message. As a conclusion, we can only say that work needs to be done in this area.

A second point is that formal modelling techniques are usable in design to the extent that they can be integrated with other techniques used by designers. As such, the use of formal models should both adapt to the very first stage of interface design, namely task analysis, and adapt to the final stages of design, such as software engineering, prototyping, and testing. In other words, formal models are tools to communicate very precisely about designs, but in rather abstract terms, which in some way or another have to be related to the real world. Formal modelling techniques may be expanded and adapted to the complementary methods that are applied in design. Summersgill and Browne (1989) report an attempt to integrate what they

call "functionality centred" and "user centred" design techniques. Walsh (1989) notes that, although there is a gap between the techniques and notations of task analysis, formal modelling, and software engineering, this is only a matter of a different focus, and not a matter of the inability to understand each others language.
Regarding the relation between formal modelling and task-analysis, what needs to be done is to find a way to translate the informal or semiformal representations of the tasks of users into a formal representation of the conceptual objects and operations which should be used in delegating tasks to the computer system. The relation between modelling user knowledge and software engineering is probably even less problematic, because both already use formal models. These models do, however, differ with respect to exactly what is modelled: aspects of the user or aspects of the interaction. Barnard and Harrison (1989) propose an interaction framework to model user-system behaviour as a bridge between modelling user knowledge and modelling system behaviour.
Another promising development is the object oriented approach, which may make it possible to create bridges between both task analysis and user knowledge modelling, and software engineering (eg. Coad and Yourdon, 1990; Jacobson et al., 1999). Eventually, it should be possible to use the very same objects to represent the tasks and task entities of the users' task world, the conceptual objects and operations of the user interface, and the data objects and procedures of the computer system, but after more than ten years of object orientation there is still much to be desired by cognitive ergonomics (see: chapter 4) and problems remain (Butler et al., 1999).

A third point, concerns the usability of formal models for designers. Part of the complexity of the models is related to the complexity of their syntax. In this respect, the most simple models are ETIT and GOMS, followed by Action Language, TAG and CCT, and the most complex models are ETAG and CLG. Another part of the complexity lies in applying the models, which involves, among other, reformulating problems into the concepts and syntax, and changing the resulting models. In this respect, and perhaps, except for ETIT, none of the formal modelling techniques is easy to use and each demands substantial effort.
For example, Wilson, Barnard, Green, and MacLean (1988) report that applying formal models does often require a high level of expertise on behalf of the designer. According to Sharratt (1987), who studied Command Language Grammar, this is especially the case when formal design specifications are changed, or design alternatives are to be compared. To attack this problem, there is a need for something like a designers' workbench, built around a particular formal modelling technique, or for providing facilities to employ different formal techniques.
The most important facility the workbench should provide, in our view, is a design approach to guide and structure design decisions. Furthermore, several tools will be necessary to relieve the designer from much of the administrative work, such as generating and changing formal design models (e.g. specialised editors that enable automatic semantic consistency checks, and templates for formalising design attempts). Together, the design approach and the toolkit should facilitate the integration between task analysis, formal modelling and implementation. The workbench should also provide for facilities to decrease the amount of expertise required in dealing with the formalism by providing adequate on line help and explanation facilities during the different stages of design (see van der Veer et al., 1990; de Haan and van der Veer, 1992b; chapter 8).

2.6 Conclusions

In this chapter it was argued that people use computers as mere tools to perform the tasks they have to do, and therefore the computer system should provide for the functions to enable users to delegate their tasks, and minimal effort should be required to learn and use the system. Furthermore, computer users create mental models of the system they are working with, which help them to explain the behaviour of the computer system, and serve to aid in planning their actions, thus reducing the mental effort required. It was argued that the designer of a computer system should consider the development of users' mental models, by taking a conceptual model as the basis for the design. Conceptual models can provide an accurate, consistent, and complete specification of the design, at different levels of abstraction in the knowledge of computer users. There are various, partially complementary methods to enhance a system's usability, each with specific advantages and disadvantages.

This chapter focussed on formal modelling techniques to represent the knowledge users need to operate computer systems, which have the advantages of formality, early applicability, relative time and cost inexpensiveness, and being useful as conceptual models. Various types of formal models were reviewed, according to their primary purpose: models for task environment analysis, models to analyse user knowledge, models to predict user performance, and representation models for design purposes. Formal modelling techniques should meet four special requirements, in addition to the general requirements imposed on formal systems and specification tools. For each of these requirements (completeness, wide applicability, validity and usability) the problems encountered with the formal modelling techniques were discussed, along with possible solutions.

Regarding the question of how to continue, we can make a threefold distinction between addressing the limitations of the formal models, the ongoing development of user-oriented computer system's design, and the use of formal modelling techniques in the design practice.

(a) The limitations of the modelling techniques that have been discussed need to be addressed. Regarding the requirement of completeness, the modelling techniques have to be extended, or alternative techniques need to be developed to enable (1) the specification of the visual presentation component, and (2) the specification of the semantics of the tasks and devices, in addition to task-action mappings. Where it concerns the requirement of a wide applicability, stronger modelling techniques have to be developed to address (1) the context sensitive aspects of the user-computer interaction, such as multi-tasking, and (2) the dynamic aspects of task control and planning, and the presentation of information and other reactions from the computer system.
From the requirement of validity follows the need for validation studies to be done by independent researchers, using real computer systems outside the laboratory. Regarding the usability of formal models, research is needed to bring about an integration between the techniques for task analysis, formal modelling techniques and software engineering

methods. Also, tools should be developed to reduce the additional amount of effort and expertise, imposed by the formal modelling techniques.

(b) Several remarks have been made about how to improve aspects of design approaches to increase the usability of computer systems. The question that follows is how to develop a better design approach, which includes all improvements needed. On the basis of the previous discussion, it may be clear that we are convinced that a user-oriented design approach should be based on a conceptual model: an accurate, consistent and complete specification of the intended system, at the different levels of abstraction, and, most importantly, considering the knowledge of the intended user. A formal modelling technique can be chosen, either from this overview or from another source, on the basis of the four requirements previously discussed, and possibly additional requirements, such as opportunities to extend the model, or to integrate it with software design tools.
Even before developing tools to facilitate the use of the modelling technique, (1) it should be determined if, and to what extent the technique and the resulting models can be integrated with task analysis and software engineering approaches. If such an integration is not possible, or only to a limited extent, then any other effort is useless. Otherwise, (2) an integration can be established, which is presumably not an easy undertaking. After the backbone of an overall design approach is thus created, (3) it may be refined, completed, and complemented with the required tools. Whereas the former two steps may take place as an entirely theoretical project, for the success of this last step practical experience will be inevitable, because only then the weak points and the gaps of the approach will show up.

(c) The approaches to formal modelling that have been discussed in this chapter have almost exclusively been used within the research domain of cognitive ergonomics. This has led to a situation where design has commonly been exemplified by small-scale studies in which either students, or even the authors, of a certain design method took part as designers of a computer system with maybe ten different functions to perform some artificial task. Although there are exceptions, the point will be clear: in order to bridge the gap between theory and practice of interface design, the application of formal models in actual interface design is required, in order to gain new theoretical insight. Only by means of full scale design examples will it be possible to show to the design community the advantages of using formal models. In this respect, one successful computer system is more valuable than ten research papers on interface design. Both the development of formal modelling techniques, and the development of methods to design usable computer systems, are best served by practical experiences.

To continue from these conclusions, on the basis of the requirements of completeness, wide applicability, validity of results and ease of use, we have selected ETAG to further investigate the feasibility of a user interface design method that is based on a formal specification. Table 1 presents a summary of the review of formal modelling techniques in terms of ease of use, semantic and pragmatic completeness, width of applicability, and psychological validity with respect to design purposes.

	syntactic	semantic	pragmatic	width of	validity for

	ease of use	completeness	completeness	applicibility	design
ETIT	high	low	low	low	very low
Action Language	fair	very low	low	fair	low
TAG	fair	fair	low	fair	fair
GOMS	high	low	fair	low	fair
CCT	fair	low	fair	low	fair
CLG	low	high	very high	high	high
ETAG	low	very high	fair/high	high	very high

Table 1. Evaluation summary of formal modelling techniques.

In terms of the completeness of representation, it will be clear that the specification models for design purposes (ETAG and CLG) are the most likely candidates. All other models are restricted in one way or another: task mappings, knowledge about the interaction language, or the use of the interaction language. Moreover, only ETAG and CLG provide the means to describe the conceptual knowledge associated with user interfaces.

Regarding the width of the applicability the same conclusion holds. Here, the GOMS and CCT models have been proposed to be applicable for performance prediction, which is not addressed in ETAG and CLG. The models for performance prediction are able to do so at the cost of mixing up competence and performance knowledge which make them unsuitable candidates for different purposes, like specification. Both the models for knowledge analysis and the models for interface specification are restricted to competence knowledge. In the case that they should be extended for performance prediction, they would (only) need an additional external performance component, yielding a much cleaner architecture. In contrast to ETAG, CLG has also been proposed to serve as a model for user modelling, to represent what is actually in the users' heads (van der Veer, 1990). In our view, this is a matter of suggestion that says very little about the feasibility of using ETAG or CLG for that purpose. Given that ETAG and CLG use a comparable grammar for representation, there is little reason to draw definite conclusions.

With respect to the validity of the models and their predictions there is a disadvantage for the more general models for interface specification relative to the more specific and smaller models. However, since our aim is design specification there is little choice. Of course, even if design specification models fail to yield valid statements about usability aspects of the end-products of design, there still is the opportunity to use 'limited theories' (Green, 1990) at the cost of usability for the designer (see: section 2.5.1).

When comparing CLG with ETAG on the basis of the available evidence, the choice is in favour of ETAG. This is due to the fact that CLG uses an ad-hoc formalism. First, ETAG is an extension of an older model: TAG, and since ETAG and TAG share the same representation for the interaction language, and there is evidence that supports TAG (Payne and Green, 1989), at least part of ETAG seems valid. Secondly, ETAG's User Virtual Machine representation derives from psycho-linguistic work by Sowa and Jackendoff, and psychological research by Klix, which also provides indirect evidence for ETAG's validity (see: chapter 3). The fact that parts of ETAG seem valid does not free us from putting ETAG to the test. The lack of evidence on behalf of CLG only implies that ETAG is the more mature model; not that it is the better one. In addition, that there is evidence for the psychological

validity of parts of ETAG does not necessarily imply that the model as a whole, once used in practice will yield valid results (see: chapter 7).

With respect to the usability of formal modelling approaches, only a few usability studies are available which makes it necessary to use the models' syntactic complexity as a main measure. No well-documented usability studies about ETAG have been available at the time of its selection (but see: chapter 7). Sharratt (1987) studied the use of CLG in classroom settings and concludes that CLG models are complete but tend to become complex because of their size which he blames, at least partly, on the not-so-neatness of the formalism. For example, CLG employs a somewhat different representations at each level of abstraction. In this respect, ETAG is much cleaner. Because the structure of ETAG is formal and relatively simple it may be manipulated by standard software engineering tools (van der Veer et al., 1990) and/or translated into more task-appropriate ways (de Haan and van der Veer, 1992b; chapter 8).

On the basis of having selected ETAG for further investigation, in the light of the general conclusions of this chapter, we plan to investigate the psychological validity of ETAG (chapter 3), its use for analysis and specification purposes (chapter 7), its use as a formal resource of information about a user interface (chapter 8), and from an analysis of how ETAG models relate to (requirements for) task analysis (chapter 5), arrive at a design methodology for interactive user interfaces on the basis of the formalism (chapter 4).

This chapter is based on the following papers: de Haan, G., van der Veer, G.C. and van Vliet, J.C. (1991). Formal Modelling Techniques in Human-Computer Interaction. De Haan, G. (1993). Formal Representation of Human-Computer Interaction.

Chapter 3:

The Psychological Basis of Extended Task-Action Grammar

Grammar does not tell us how language must be constructed in order to fulfil its purpose, in order to have such-and-such an effect on human beings. It only describes and in no way explains the use of signs.

Wittgenstein, L. (1953).

Abstract

In this chapter we discuss the psychological basis of Extended Task-Action Grammar (ETAG). ETAG is both a method to represent the knowledge a competent user has about the user interface and a method to represent the user interface for the purpose of design. The psychological considerations underlying the format of the model are: the use of a feature grammar to represent the interaction language, and the decomposition of the description into four levels of abstraction, similar to those in (natural) language processing. The psychological choices underlying the content of the model concern the choice of the canonical basis. This is the most abstract level of representation in ETAG. The concepts defined at this level are analytical with respect to acquiring knowledge of the world, and serve as the source from which all other concepts in ETAG are derived. The concepts in the canonical basis are derived from psychological research into semantic memory by Klix and colleagues, and from psycho-linguistic analyses into the semantics which underlie language by Sowa and Jackendoff. The chapter discusses and exemplifies the structure of ETAG in terms of psychological considerations and it concludes with a brief comparison between ETAG and other psychologically inspired design models in Human-Computer Interaction.

3.1 Introduction

In this chapter we discuss the psychological and psycholinguistic considerations and theories underlying Extended Task-Action Grammar (ETAG), a method to describe user-interfaces for the purpose of design. In addition, we will also briefly discuss psychological concerns regarding the practical usage of ETAG by user interface and dialogue designers, the primary users of the method.

An ETAG specification consists of a description of the knowledge a fully competent user must have in order to use a computer system successfully. To the extent that knowledge of the workings of the computer system is required to perform tasks, the internal functioning of the system is regarded as part of the user interface as well (Tauber, 1988).

ETAG descriptions consist of four parts: the Canonical Basis, the User's Virtual Machine (UVM), the dictionary of Basic Tasks and the Production Rules. The Production Rules describe how each basic task is to be invoked by physical actions, by means of a derivation along four levels of abstraction or processing, similar to the four lower levels of abstraction involved in producing and understanding linguistic utterances. The Dictionary of Basic Tasks lists the tasks available to the user in terms of commands which do not require supervision during execution.

The User's Virtual Machine is the most important part of ETAG, as it describes the computer system (solution) representation of the problem domain and the internal workings of the system in terms of human concepts the user has to master in order to use the system. The

concepts used in an ETAG description are hierarchically derived from a canonical set of concepts; the Canonical Basis.
The concepts in the Canonical Basis are assumed to be the very basic concepts in any human language and thought, such as Object, Place, State, etc. These concepts are directly taken from psychological research by Klix and colleagues (1984b, 1986, 1989), and from psycholinguistic analyses by Jackendoff (1983, 1985) and Sowa (1984).
On the basis of these ontological concepts it is possible to derive and very precisely define the concepts needed to describe a user interface, such as that of a File, a Mailbox or an operation like Move. Figure 1 presents the general structure of an ETAG representation in plain English. Each level is exemplified by mentioning a typical element specified at that level, disregarding the specific format requirements.

- the Canonical Basis "the world consists of Objects and Places"
- the User's Virtual Machine (UVM) "files are Objects at Places"
- the Dictionary of Basic Tasks "there is a task to move files"
- the Production Rules "to move a file one hits the keys ... "

Figure 1. Levels of an ETAG specification.

The structure of this chapter is as follows. Section 3.2 explains what ETAG exactly is, and what it represents in which way, and how it relates to other formal modelling approaches in terms of knowledge representation.
Section 3.3 discusses the psychological considerations underlying the particular structure of ETAG representations.
Section 3.4 discusses the considerations regarding the content of ETAG representations; which concepts are used to describe the semantics of a user interface, modelled in ETAG. This section explains the reasoning process behind making these choices and focuses on the contribution of Klix and colleagues (1984b, 1986, 1989).
Section 3.5 treats the psychological basis of ETAG at a more detailed level, discussing the logical problems that arise when creating a formal system to represent meaningful material. This section extensively discusses the contributions of Sowa (1984) and Jackendoff (1983, 1985), including the problematic difference between formal and human reasoning.
Section 3.6 provides a brief example how the psychological basis is used in ETAG to represent part of an imaginary but otherwise very real user interface.
Section 3.7 briefly summarises the foregoing and discusses the merits of the model in comparison to several alternative models with more or less the same purpose.

3.2 Extended Task-Action Grammar

Extended Task-Action Grammar (ETAG; Tauber, 1988, 1990) can be viewed both as a method and a tool to design user-interfaces, and as a theory about knowledge representation. ETAG is a design method because it enforces, or at least stimulates, a particular approach to the process of designing user interfaces.
Using ETAG as a method, the design process can be characterised as top-down. It is based on considering the knowledge that users need in order to utilise the emerging system for

task-performance. As a method based on user-knowledge considerations, an ETAG representation is a specification of the knowledge a user should have. As such, it should have a high degree of psychological validity.

ETAG is also a tool to support the design process. Its specification language serves as a vehicle to specify and lay down the outcomes of design decisions at progressively increasing levels of detail. As a specification tool, ETAG should allow for a complete specification of the relevant aspects of the structure and behaviour of a computer system.

ETAG is also a knowledge representation theory. It serves as a vehicle to specify human knowledge on the basis of certain assumptions about such knowledge. Because of these assumptions, ETAG can also be regarded as a theory about human knowledge representation. User-interface design using ETAG is based on the specification of the knowledge that a competent user of the computer system will have about the basic elements in the domain of application which are relevant in the context of the computer system, and about how the internal representation of the domain within the computer system changes by performing tasks through the user interface. At least implicitly, and insofar as it is relevant for design purposes, assumptions are made in ETAG about human knowledge representation concerning what is represented, how it is represented, and the relative importance of aspects of such representations.

3.2.1 What ETAG represents about Users

In ETAG assumptions are made about human knowledge representation: what knowledge is represented and how it is represented. Nevertheless, ETAG does not describe the mental model of a user and not even the mental model a competent user will have about a user interface.
Mental models are the mental representations of the user interface that users "run" when they interact with a particular computer system in order to help them plan actions and understand the behaviour of the system (Norman, 1983). Because mental models only serve to guide their owners, they need not be complete and correct. According to Norman, mental models indeed do generally contain omissions, errors and misapprehensions.
ETAG representations, however, exactly specify what has to be known about the procedures and workings of a computer system in order to successfully perform tasks with it. ETAG representations describe the necessary and sufficient knowledge to submit tasks and, as such, they do not contain errors, omissions or superfluous elements. ETAG representations are what Norman (1983) calls conceptual models of user-interfaces, which are created and used by, for instance, teachers and designers for the purpose of specification and communication.
ETAG representations only describe user-interfaces and, as a representation method, ETAG does not attempt to model real users. However, because the interface is modelled in terms of the knowledge of a hypothetically all-knowing competent user, it does describe something about real users as well, namely: what they would 'ideally' know about the interface. In Chomsky's (1965) terms, ETAG is a competence model because it conceptualises the knowledge required to correctly submit tasks to a computer system. Similarly, ETAG representations describe the knowledge that should be included in a user's mental model as a

prerequisite to make successful use of that system, although it is not specified how users structure this knowledge.
Concluding, it is not claimed that either the contents of ETAG representations or their structure are psychologically valid for real users. It is claimed however, that the knowledge described in an ETAG representation is a psychologically valid description of the necessary and sufficient knowledge users should have for competent task performance.

3.2.2 What ETAG represents about User Interfaces

ETAG provides a language and structure to model user interfaces for the purpose of design, by means of modelling what an ideal competent user should know about the interface, and, insofar as a user needs to know, the workings of the underlying application. ETAG describes the interaction language between the user and the system, the available commands, and a description of the internal workings of the computer system insofar as it is needed to understand what happens when delegating tasks.
As such, ETAG describes the lexical, the syntactic and the semantic aspects of a user interface. ETAG does not describe the details of the presentation of information on the display screen and the specific knowledge of particular users and the strategies they use. The choice, what ETAG is about and what it is not about follows directly from a specific view on what constitutes a user interface.

The user interface as it is viewed in ETAG extends beyond what is usually considered as such. In software and computer industry magazines, for example, the user interface of a product is generally understood to denote the way in which information on the user's screen is organised and presented, together with the elementary command actions such as mouse clicks and menu selections.
Interface builders such as Gilt in the Garnet system and InterViews' Build are indeed meant to create such a presentation interface and define the set of available elementary actions. For this reason their respective authors (Myers et al., 1990; Vlissides and Tang, 1991) call these tools user-interface builders. However, from the perspective of ETAG they are only concerned with a small aspect of the user interface, namely, the graphical presentation interface.
This concept comes close to what Norman calls the system image (Norman, 1983), which covers all aspects of a computer system the user comes into contact with. The interaction hardware, the presentation graphics and the interaction language constitute indeed the first areas of contact between a user and the computer system. However, the system image is not the same as the user interface.
The user interface as it is viewed in ETAG includes everything a competent user has to know in order to make use of the functionality of a computer system, including knowing what functionality is provided, what the procedures are in order to invoke functions and, most important, knowledge about the internal workings of the system or "virtual machine" which is needed to plan and evaluate command procedures in relation to the user's goals (Tauber, 1988).

This means that when it comes to attaining task goals, users will also need to know about the computer system's representation of the (task) world in addition to the 'usual' elements of the user-interface: the interaction language and the graphical presentation interface. To put it

bluntly: knowing how to click on file icons and how to select from a command menu is still far removed from meaningful computer use.
As a small example, consider the Macintosh clipboard (for those who do not like this White Rat of HCI, the same applies to the buffer feature in Vi, Emacs or Teco). When users do not know what the clipboard is or how to use it then certain tasks will be very difficult to perform and the 'perceived functionality' of such a system will remain very limited.
In experiments of Payne (1987) and Howes and Payne (1990), for instance it was clearly shown that when subjects do not know what a clipboard is and what it is meant for, they will have great difficulty in removing a piece of text from one location in a text-file and inserting an identical piece of text at two different locations. These results show that in order to completely describe a user interface, it is necessary to include a conceptual model with the semantic and task level aspects of the computer system.

Instead of separating the user interface from the application based on, for instance, what is visible, van der Veer (1990) proposed to draw the line on the basis of the (knowledge) aspects relevant for task performance which the user comes into contact with. In this view, there are three different, increasingly complex interfaces between the user and the machine:

- the **presentation interface** addresses all the visible task-relevant aspects of the user-interface,
- the **perceptual interface** is the combination of the presentation interface and the interaction language,
- the term **user-interface** is reserved for all aspects of a computer system, perceptual as well as conceptual, which are relevant to attain task-goals.

ETAG does not completely cover the definition of the user interface as given. First, it does not address the presentation interface. In ETAG, elements of the presentation interface such as the labels of menu items and the act of clicking buttons are named or mentioned, but these are only included insofar they are needed to completely describe the non-graphical aspects of interfaces.
As a second omission one might point at the fact that ETAG does not include user-strategies in the description of the user interface. In a sibling model of ETAG, Moran's (1981) Command Language Grammar (CLG), user strategies are also described as part of the topmost 'task level' description of the interface. Moran's strategies can be called 'methods' as they involve reformulating high level tasks into simple sequences of basic tasks to be submitted to the computer system. For example, to invoke the mail program and issue the command to produce a list of messages in order to see what is in the mailbox, requires a simple combination of two basic tasks with no decisions in between. ETAG can be extended to describe such methods or, as van der Meer (1992) called them: menu-tasks.
On the other hand, user strategies which require the user to gather information or choose between alternatives, such as choosing between saving the text of a message or the whole message, are outside the scope of ETAG. Any behaviour which involves making explicit decisions depends on the performance characteristics of (individual) users, which goes far beyond a description of the user's (competence) knowledge. Interfaces may or may not facilitate strategy development by users (see: Payne, 1991; Howes and Payne, 1990), but the strategies themselves belong to users' performance characteristics which are not

characteristics of the user interface. For this reason, user strategies are left out of the ETAG representation.

3.2.3 Related Approaches to User Interface Representation

ETAG is not the first model to describe user interfaces, but it is one of the few models which attempts to capture interface knowledge, including conceptual knowledge (see chapter 2; de Haan et al., 1991). Other, more restricted models in HCI aiming at representing the user's knowledge of the interface are: Reisner's Action Language (Reisner, 1984), the GOMS family of models (Card, Moran and Newell, 1983) and Cognitive Complexity theory (CCT; Kieras and Polson, 1985). These models are restricted to describe the knowledge and use of the interaction language. Nevertheless, Kieras and Polson state that CCT is able to describe device complexity, but since they base their conclusions on analyses of transition networks of the interaction language, they are simply confusing device and interaction language complexity. Regarding the clipboard example mentioned before, CCT is perfectly able to describe (the complexity of) the tasks to move or copy material between the clipboard and the object that is being edited, but it is not able to address complexity issues that follow from device characteristics, such as whether or not the contents of the clipboard are visible, or that its contents is overwritten by subsequent cut and copy operations.
Models like Action Language, GOMS and CCT describe what Kieras and Polson call the "how to do it" knowledge of performing tasks. Evidence, collected mainly by the authors of each of these models, shows that they are useful and able to explain various experimental manipulations of the task language (see e.g. Reisner, 1981, 1984; Card, Moran and Newell, 1983; Payne and Green, 1989; Polson, 1987).

Two other models come -relatively speaking- close to ETAG's scope of description: Task-Action Grammar (TAG; Payne and Green, 1986) and Command Language Grammar (CLG, Moran, 1981).
Task-Action Grammar also aims at describing the interaction language, or the how-to-do-it knowledge. In addition to merely describing the structure of the interaction language, TAG uses a particular variant of a set grammar (called a feature or an attribute grammar) to describe the similarities and differences in the form of the task actions on the basis of the meaning of the tasks to the user. TAG descriptions address some aspects of the semantics of the interface, such as that interfaces which allow for recognition instead of recall of command names are easier to use. However, TAG still does not address the complexity of the (e.g. clipboard) device. In addition, TAG lacks a sound basis to describe the meanings of task features to users. Payne and co-workers have proposed extensions to TAG as first attempts to address device knowledge (Payne, Squibb and Howes, 1990) and presentation display information (Payne, 1991).

Moran's Command Language Grammar (CLG; Moran, 1981) and ETAG both aim at what Kieras and Polson (1985) call the "how it works" knowledge of a device and, for that purpose, both provide a language to describe the conceptual structure of user interfaces. In addition, CLG also aims at describing users' individual knowledge of user interfaces by means of a description of task strategies. In a design study by Sharratt (1987), CLG did produce mixed

results regarding facilitating the specification of user interfaces. This was mainly due to its size and lack of restrictions on (ab)using the formalism.

3.2.4 The Structure of ETAG Representations

In ETAG a set of abstract concepts derived from physical reality is used to describe the types of objects and events, etc. which are part of the computer system. These in turn are used to describe the workings of the commands and finally to describe the command procedures themselves and their derivation.
In developing ETAG, Tauber (1988, 1990) used elements from Jackendoff (1983, 1985), Klix (1984a, 1986, 1989) and Sowa (1984) to determine a canonical basis to describe the how-it-works knowledge of computer systems for task performance. A canonical basis is a set of definitions of "basic" concepts, in whose terms a particular field of application (the Universe of Discourse) will be described. The canonical basis defines concepts like OBJECTs, PLACEs, EVENTs and ATTRIBUTEs in order to represent the so-called User's Virtual Machine (UVM).
The UVM describes the knowledge required to understand how a particular computer system works when performing tasks with it. For example, the use of a trashcan object in a graphical user interface (GUI) can be understood when it is regarded as an object which provides the place where e.g. file objects go when they are deleted, until through some special events they are recovered or definitely made to cease to exist.

The functionality of a computer application is described in the Dictionary of Basic Tasks which lists all tasks the interface provides, together with the arguments to be specified by the user. This representation connects the how-it-works knowledge with the how-to-do-it knowledge by means of specifying for each task: which (conceptual) event it invokes, and which commands and command-arguments should be provided by the user.
Finally, a set of rewrite rules further specifies the user's how-to-do-it knowledge, somewhat similar to other formal models. The rewrite rules are structured into four levels, according to Moran's CLG (1981), to reflect psychologically valid levels of abstraction in interaction. The rules are represented using a semantic feature grammar, borrowed from Task-Action Grammar (Payne and Green, 1986), in which the concepts of the UVM are used as features. Because the UVM determines which features are used to distinguish between tasks, and also because the basic tasks of the interface are described rather than the simple tasks of the user, the assignment of features in ETAG no longer depends on the intuition of the designer about the user, as it does in TAG.

To conclude, ETAG can be used to describe the knowledge that is needed to use a computer system, both where it concerns semantic knowledge (how it works), as well as knowledge about the interaction language (how to do it). Because ETAG represents this knowledge from the point of view of a user, this knowledge represents what is relevant about a user-interface to perform tasks.

3.3 ETAG as an Eclectic Choice of Theories - Structure Choices

ETAG is a model for knowledge representation and as such it should be based on psychological considerations about how knowledge is represented by the user. ETAG is restricted with respect to both the field of knowledge it describes, namely knowledge about the user interface, and the type of user whose knowledge is described by ETAG, namely the competent or perfectly-knowing user. Nevertheless, apart from the additional practical consideration that the model should be suitable to support the design process, ETAG simply must be psychologically valid with respect to both types of knowledge it describes: knowing how-it-works and knowing how-to-do-it.

The how-to-do-it knowledge describes for a certain set of tasks, how to derive valid command specifications to meet the goals of these tasks. The how-to-do-it knowledge or knowledge about the interaction language captures both how users represent tasks with respect to goals and effects, and how to derive the actual command procedures from the task representations.
The how-it-works knowledge describes what the idealised competent user knows about the internal workings of the system in terms of which things change in what ways when processing a task. The how-it-works knowledge or, in ETAG terms, the User's Virtual Machine (UVM) describes what a user should at least know about the domain of application or the task world as it is represented in the computer system in order to make meaningful use of the system. As such, the UVM formally represents what knowledge a user should have about the computer system's input-output behaviour in order to understand and use it.

The how-it-works knowledge consists of knowing what types of elements (objects or 'things') exist in the domain of application, the characteristics of each of these types, how they relate, and how the execution of a task by the computer system changes the state of the task world.
ETAG's UVM describes how the task world is internally represented in the computer system. In principle, however, this representation can be applied equally well to provide a conceptual model of the task world as it may exist outside the computer system.

In ETAG, psychological considerations grounded the choice for a feature grammar to model how tasks are represented, the distinction into four levels of abstraction along which command procedures are derived from task representations, and an event-based model which uses objects, features and relations to describe the semantics of the task world, respectively.

Tasks are represented using a feature grammar which is adapted from Task-Action Grammar (Payne and Green, 1986). In TAG all tasks have the same feature slots but different features or feature values. For each task, the set of feature values defines the psychological similarities and differences with other tasks in terms of task effects and task elements. For example, a task command to delete a word may be like a task to delete a character, except that the unit is different. Likewise, a task command to move a pointer a line downwards may be like the task to move it a line upwards, except for the difference in direction.
Each individual task has a different set of feature values, and to the extent that tasks are similar in terms of meaning and form they will share a larger part of each other's description or rewrite structure. For example, when a task command to delete a word has a form similar to the task to delete a character, their difference in meaning will be represented at a low -lexical- level in the rewrite structure. When these two tasks would have a dissimilar form, their similarity in meaning would get lost in a different rewrite structure. In the first case, the

user will be able to infer one command procedure from an other, but in the second case the absence of semantic-syntactic alignment will not help the user to do so.
Having decided to use a feature grammar to describe task representations still leaves a number of alternatives with respect to modelling the structure of the knowledge used to derive the actual command specifications: the valid sentences of the interaction language. In ETAG the knowledge of the so-called rewrite rules is described using different levels of abstraction. More specifically, there are four such levels, describing the order in which concepts are specified (the syntax), the way in which each of the command elements should be specified (the specification or interaction style), the external representation of each element (the lexicon), and finally how each element should be specified in terms of physical actions.

The representation of the how-it-works knowledge or the UVM is the most important part of ETAG, because it describes those aspects of a user interface which are most important to the user, and because this is what sets ETAG apart from almost all other formal models in human-computer interaction. To describe the UVM, a combination of Klix' theory of semantic memory and Jackendoff and Sowa's linguistic theories of semantics is used.
In order to describe the structure of human semantic memory, Klix investigated (1984b, 1986, 1989), by means of experimental research, which concepts are basic to memory.
For a slightly different purpose, Jackendoff (1983, 1985) and Sowa (1984) each independently devised an otherwise strikingly similar linguistic theory about how to represent semantic knowledge about the world. The theory proposes a set of ontological categories or basic concept types such as [OBJECT], [PLACE] and [EVENT] to serve as primitive "parts of speech" (Jackendoff, 1985, pg. 150), and uses graph theory and a graph grammar to glue these parts together into representations of meaningful concepts and statements.
This theory is far more powerful than is needed to describe the UVM. Still, as the UVM description uses its main parts, the psychological validity of ETAG does depend on the underlying psychological and linguistic considerations.

The following sections will briefly describe the use of a feature grammar to distinguish among tasks (section 3.3.1), and the separation of rewrite rules into four levels of abstraction (section 3.3.2). Although these subjects are less important and have been discussed before (chapter 2; de Haan et al., 1991), treating them is necessary to show how they fit together with, and complement the contributions of Klix, Jackendoff and Sowa in shaping ETAG. Section 3.4 will discuss the reasoning underlying the contents of the canonical basis, and section 3.5 will focus on more formal aspects of ETAG and the particular logic that is used. Section 3.6 will discuss how all the pieces together form ETAG.

3.3.1 A Feature Grammar

ETAG derives from a theory-based tradition in HCI with strong links to cognitive psychology, and more specific, it derives from the tradition which attempts to apply linguistic and grammar research outside the area of natural language processing. The basic idea of this research tradition is that, since information structures facilitate human performance in many areas of conduct, and formal grammar help describe natural languages, it makes sense to apply formal grammar to not-so-natural kinds of language, such as command languages. To this may be added that grammar rules are not only useful to describe linguistic structures, but

also as a means to prescribe how languages should be structured to facilitate learning and using them.
These ideas are especially applicable to HCI where human computer users have to learn very strictly defined interaction (and programming) languages, in order to be 'understandable' for the computer device, and where computer programmers are completely free to design an interaction language without being hindered by e.g. tradition. As an approximation of the Grammar In the Head (GIH; Payne and Green, 1983), the grammar rules describing an interaction language can be used to measure cognitive complexity and be used to predict human performance characteristics, as well as to prescribe how an interaction language might be improved to suit human users.

Historically, Reisner (1981) was the first to apply formal grammar for psychological purposes: analysing the ease of use of command languages. Before that, formal grammars were already used in language research (Chomsky, 1965) and Software Engineering, where Backus Naur Form (BNF) was specifically developed to facilitate the formal definition of the programming language Algol60 (Backus et al., 1964).
In Reisner's now classical study, she represented two different command languages for a drawing package in standard BNF and used the difference in the number of rules required to predict differences in ease of use. Whereas Reisner (1981) only used BNF as a means to analyse interaction languages, Richards et al. (1986) applied an enhanced version of BNF, both as a means to analyse ease of use difference between design options, and as a notational language to represent full human-computer dialogue designs. The following example, from Reisner (1981), show how BNF can be used to identify and to resolve inconsistency in a command language:

```
Inconsistent system:
        Rules-
                to select a line     ::=  Set Line Switch + Go
                to select a circle   ::=  Set Circle Switch + Go
                to select a square   ::=  Set Square Switch + Go
        "Necessary" rules-
                to select a shape    ::=  Set Shape Switch + Go
                shape                ::=  LINE | CIRCLE | SQUARE
                to select text       ::=  NULL
Consistent System:
        "Necessary" rules-
                to select a shape    ::=  Put Cursor in Shape Icon
                shape                ::=  LINE | CIRCLE | SQUARE | TEXT
```

Although BNF is a relatively simple and easy-to-use grammatical formalism, Payne and Green (1986) argue that it has important limitations. First, BNF descriptions are restricted to syntax and they don't say anything about the semantics of the language described. Secondly, compared to more recent formalisms, BNF is not a very powerful description language. To describe a particular language a lot of different rules are required because BNF has no facilities to describe similarities and differences between rule-forms other than a simple choice between alternatives.

In order to deal with this problem, and to address the lack of semantics in the BNF notation, Payne and Green (1983) first use a Van Wijngaarden two-level grammar, with separate levels to describe rule forms and rule derivations, and later (Payne and Green, 1986) switch to a more flexible grammar, a feature grammar. In feature grammars, the form of the derived sentences depends on the presence of feature values. Feature values may depend on the context in which they occur. As such, feature grammars are more flexible than Van Wijngaarden or BNF grammar that use the principle of "uniform replacement" of terms. Features are said to 'tag' sentence forms, which explains the name TAG for Task-Action Grammar. In TAG representations of command languages, the features represent a part of the meaning of a command to the user, which enable one, if the structure of the command language allows so, to represent the family resemblances among command language sentences in terms of their meanings and forms. The following example shows TAG's use of similarities in rule forms, applied to cursor movement:

```
Simple Tasks:
        move cursor one character forward      { Direction = forward, Unit = char }
        move cursor one character backward     { Direction = backward, Unit = char }
        move cursor one word forward           { Direction = forward, Unit = word }
        move cursor one word backward          { Direction = backward, Unit = word }

Rule Schemas:
        Task [ Direction, Unit ] -> symbol [ Direction ] + letter [ Unit ]
                symbol [ Direction = forward ]    ->     "ctrl"
                symbol [ Direction = backward ]   ->     "meta"
                letter [ Unit = character ] ->    "C"
                letter [ Unit = word ]      ->    "W"
```

To describe interaction languages, Payne and Green (1986) propose to use the smallest possible set of features to describe the similarities and differences in a meaningful way. In addition to the number of required rewrite rules, the number of features can now be used as an indication of the relative psychological complexity of the interaction language.

To some extent, TAG is able to express 'world knowledge' about the lexicon. When, in the example, the command-names are replaced by sensible ones or arrow keys, all the "name" rules could be replaced by one rule: "**name[direction] ::= Known-Item[direction]**" (see Payne and Green, 1986). Although TAG offers the tools to relate the mental 'Grammar in the Head' to command language grammar, TAG does not say anything about higher level semantics, and in particular TAG does not address how mental models help users to understand the computer system and plan their actions (Norman, 1983).

Tauber (1988) and the COST-11-ter interdisciplinary "Human factors in telematic systems" working group (van der Veer et al., 1988) proposed ETAG in which the mental model of a perfectly knowing user is used as the conceptual or design model for the user-relevant aspects of a computer system. In ETAG, Payne and Green's (1986) Task Action Grammar (TAG) is used as a notation for the interaction language, and extended (hence: *Extended* TAG) with a conceptual specification of the user's device knowledge, connected to each other by means of the psychological features of the grammar.

3.3.2 Levels of Rewrite Rules

It is common to distinguish several levels of abstraction in describing the interaction between the user and a computer system (Moran, 1981; Nielsen, 1986; Fröhlich and Luff, 1989). Formal models like TAG, GOMS, CCT, ETIT and Reisner's Action Language do not describe the interaction language in terms of levels of abstraction, even though there are two compelling reasons to do so.

It is common practice in psycholinguistics to distinguish between the levels of abstraction such as semantics and syntax, and it is also psychologically valid because evidence points out that each of these levels independently influences language understanding (Foss and Hakes, 1978).

Levels of abstraction are also very important to user interface design, because they generally correspond to different types of design decisions, such as the interaction style, the order of command argument specification, and the naming of commands.

In both natural languages and computer languages, 'users' have to pass through these levels top-down to produce sentences or commands, and pass through them bottom-up to understand sentences or system feedback.

Moran (1981) distinguished the following levels (the equivalent, though not identical levels in the production rule section of ETAG are shown in square brackets; the remaining equivalent higher level parts of ETAG are shown in rounded brackets):

Task Level	
Semantic Level	(User Virtual Machine)
	(Dictionary of Basic Tasks)
Syntactic Level	[Specification Level]
	[Reference Level]
Interaction Level	[Lexical Level]
	[Keystroke Level]
Spatial Layout Level	
Device Level	

Moran's Task Level, which deals with higher level user tasks, is not used in ETAG. Insofar as the Task Level addresses sequences of Basic Tasks, ETAG provides equivalent information in its Dictionary of Basic Tasks. Where Moran's Task Level addresses user strategies, it falls outside the scope of ETAG as a user interface description method. In addition, neither the Spatial Layout Level nor the Device Level are used in ETAG. These levels, which are only mentioned but not described by Moran (1981) address very low level considerations such as device input-output characteristics, which generally fall outside the user interface proper.

The other levels of Moran are used in ETAG, although with significant adaptations.

3.3.3 Knowledge Representation

To represent knowledge in a formal manner requires several decisions to be made about the form and contents (the basic constituents; Sowa, 1984, pg. 96) of the representation. Regarding the form of the representation, the choice is between a propositional representation, a graphical one, a network, etc. For the purpose of knowledge representation for design, this choice has to be made mainly on the basis of suitability; truth or psychological validity are less important.

In principle, knowledge represented in one formal system can be translated into another one without loss and without causing ambiguity. As such, the most important consideration is whether a particular form of representation may be more or less suitable for specific purposes, such as inspection by human beings, execution by a computer, being subjected to numerical analysis methods, etc. This can easily be exemplified by programming languages. In formal terms, programming languages are equally powerful, as they all have the power of an abstract Turing machine. Nevertheless, in terms of programming languages as notations to implement a solution, certain languages are more suitable because they allow for different kinds of formulations at different levels of abstraction.

An important consideration with respect to suitability is that the structure of ETAG and specifically ETAG's structuring of the production rules is adapted to decisions in the design of a user interface and the interaction language. The validity of ETAG for design purposes will not be discussed here as this concerns a different kind of validity than psychological validity.
A second consideration regarding the validity of ETAG is of lesser importance for theoretical reasons, but for practical reasons face validity or intuitive psychological validity can be worth consideration. Just like a good after shave should smell nice from the first sniff, an ETAG representation has to look promising to people involved in both software engineering, knowledge representation and cognitive psychology. This has not been an explicit consideration behind ETAG, but working in an interdisciplinary setting will have helped to create a greatest common denominator.
The most important consideration regarding validity is the validity of ETAG as a representation of user knowledge. Without validly representing what a competent user should know, ETAG will not be suited for design purposes. The question whether ETAG representations are valid incurs two subquestions. On the one hand the question whether the form and structure of the notation complies with the form and structure of mental representations in real users and, on the other hand, the question whether the concepts used (in the canonical basis) in ETAG to represent the content of the knowledge of a competent user match those of real users.

Just like FORTRAN is not a very suitable notation to express, for example, linked lists or B-trees, whereas Lisp is, ETAG's propositional representation determines what kind of knowledge can be or cannot be easily expressed.
ETAG uses a propositional representation to represent knowledge in the form of sentences. Propositions make it easy to represent or communicate about declarative knowledge and make it hard to represent or reason about procedural knowledge (Gilmore and Green, 1984). Procedural knowledge or user performance, however, is not what ETAG is meant for, whereas declarative or competence knowledge is.

The second aspect of ETAG as a knowledge representation notation or scheme relates to what can be expressed in ETAG, in terms of meaning. In order to make a knowledge representation scheme applicable to the real world of user interface design requires choices regarding the contents of the model. These choices consist of choosing a particular set of categories of phenomena that should be distinguished (the types of 'objects' and relations) and labels to mark these, which serve as the basis for applying or 'stuffing' the form of the representation. This is what Sowa (1984) calls the canonical basis of a model (actually, Sowa’s canonical

basis is only a part of the canon, but, in order to keep things simple, only the term "canonical basis" will be used). The canonical basis determines how the Universe of Discourse is viewed and it specifies what the modeller presumes to be the basic knowledge categories in the object being modelled: the competent computer user.

Since this subject forms the cornerstone of ETAG, it will be treated in a separate section; the next one. Here it will suffice to briefly exemplify the importance of the concept of a canonical basis in setting a particular point of view on the Universe of Discourse (of user interface design).

In a user interface description language with an implicit view of the interface as a communication channel to an obedient agent with a very limited vocabulary, the focus of modelling will be on the creation of the syntactically correct commands. This is the view underlying, for instance, Reisner's psychological BNF interpretation.

In a different view, the user interface can be seen as a collection of handles to a machine. Once the handle assignments have been mastered, what remains is the appropriate distances between them and the forces needed to activate them. This is the GOMS or the CCT model in a nutshell.

Finally, the user interface can be seen as an abstract little world within the computer, where it matters to find out what there is, before it can be manipulated. This view, which is quite similar to ETAG's, will stress the complexities in learning about the internal world or universe.

The essential point of this example is that although the user interface may be exactly the same, the way it is viewed may lead to very different statements and predictions regarding usability.

3.4 A Canonical Basis for System Knowledge - Content Choices

In the previous section several choices regarding the form of ETAG representations were discussed. In this and the next section the focus will be on 'deep' considerations regarding what can be meaningfully expressed in ETAG, or what considerations led to the selection of ETAG's canonical basis.

According to Sowa (1984) a canonical basis consists of a hierarchical set of category types, each with a particular label or name, and a set of rules represented as conceptual graphs to relate the types to each other, and to relate the types to their instances in reality.

As such, a canonical basis serves as the starting point to derive a potentially infinite set of sensible, but not necessarily true facts or conceptual graphs.

Other authors may use different terms to describe a canonical basis (see e.g. Minsky, 1975; Jackendoff, 1983; Norman and Rumelhart, 1975). The main thing is that to build a formal knowledge representation, one needs categories to distinguish among 'things', definitions to apply these categories to reality and relations among these categories to extend and use the representation.

The size of the canonical basis, or the set of primitive propositions, graphs, etc. ('facts') defines the initial world knowledge of the system. Logically speaking there are no constraints on the size and contents of the canonical basis, as long as it is internally consistent. Internal consistency is mainly a matter of the form of the representation. In practical terms however,

constraints on the size and the content are also important because the canonical basis is modelling reality. From this it follows that the canonical basis should at least be understandable to modellers and, because the truth of the resulting model depends on the canonical basis, it should be psychologically valid.

The psychological validity of the canonical basis refers to the choice of the semantic concepts and the relation between them: the concepts chosen to serve as the basis of ETAG should be the very same ones as those which are presumed basic in terms of human cognition in general. Regarding this aspect, ETAG derives from three different but related sources:

- psychological knowledge about human semantic memory, and especially the contributions by Klix and colleagues (e.g. Klix, 1984b, 1986, 1989),
- the work of Sowa (1984) aiming at the development of semantic databases,
- Jackendoff's (1983, 1985) work on conceptual representations in the context of psycho-linguistics.

The research of Sowa and Jackendoff has been directed at analysing the grammatical features of language. Sowa's main purpose was to find solutions for the problem on how to create semantic databases. For Jackendoff, the main incentive was psycho-linguistic in nature: to develop a model of conceptual representations underlying language production and understanding. In both cases, the two main sources of inspiration were the developments in the research of reasoning processes in man and machine, and the (partial) failure of Chomsky's transformational grammar as a model of language production and understanding.
Although Jackendoff and Sowa worked and developed their theories independently, their theories are similar to such a remarkable extent that they will be treated together in section 4. Only where it is necessary and appropriate, differences will be mentioned. The remainder of this section is devoted to a discussion of Klix' theories.

3.4.1 Semantic structures

During the seventies, considerable effort was spent in psychology on the problem of representations in semantic memory (see e.g. Schank and Colby, 1973). As a direct consequence of the shift towards the symbolic information processing view of human behaviour, it was argued that descriptions of semantic long-term memory would help to explain many higher level cognitive behaviours, such as reasoning, language understanding and procedural learning, and create opportunities for improved machine implementations of these faculties.
Based upon semantic networks (at a much higher level than the currently fashionable neural nets) prototype systems were built which could indeed understand and translate some natural language and partially simulate belief systems. Although semantic networks and spread of activation were established as the main paradigm for long-term memory research (Schank and Colby, 1973; Norman and Rumelhart, 1975), in practical terms semantic network theory did not fulfil its high expectations.
It did not prove too difficult to build specific purpose demonstration systems, but it was rather difficult to build general systems, or even to build comparable systems. By way of conclusion

one might say that too much effort was spent on getting quick results instead of finding out how things work in real humans.

3.4.2 Klix' contribution

In order to establish a common ground for knowledge based systems it is necessary to determine which concepts play a basic role in (human) cognitive representations of real world problems. To this end various methods are available ranging from highly subjective self-observations to very formal logical analyses of meaning in language. Using mere self observation it can be recognised that there is a difference between property-related concepts and event-related concepts. For example, the association between the concepts 'cat' and 'dog' can largely be explained by referring to their common and different properties. In the same way there are associations between concepts like 'to cut' and 'knife'.

The main problem that Klix and his co-workers took as their research subject was to establish a network-based model of long-term memory which is general enough to be applicable to a variety of psychological phenomena and problems in different semantic domains, yet specific enough to yield empirically testable hypotheses and predictions (Klix, 1984b). This model has not been provided with a specific name. For practical purposes it will be referred to as the Humboldt Universität Gruppe - model (HUG) to signify that it is the result of a collective effort.

According to Klix, the association between the concepts 'to cut' and 'knife' is not mediated by a more-or-less arbitrary number of shared properties but by a semantic core: the event of 'cutting' in which 'to cut' represents the act itself and the 'knife' the instrument involved. Event concepts are sets of object concepts linked by a semantic core and this core is usually represented by a verb (Klix, 1989, pg. 323). In the HUG framework the links are provided by the relations: ACTors, INSTRuments, OBJects, LOCations and FINalities, which are derived from both phenomenological and linguistic analyses. Figure 2 presents an event concept according to the HUG model.

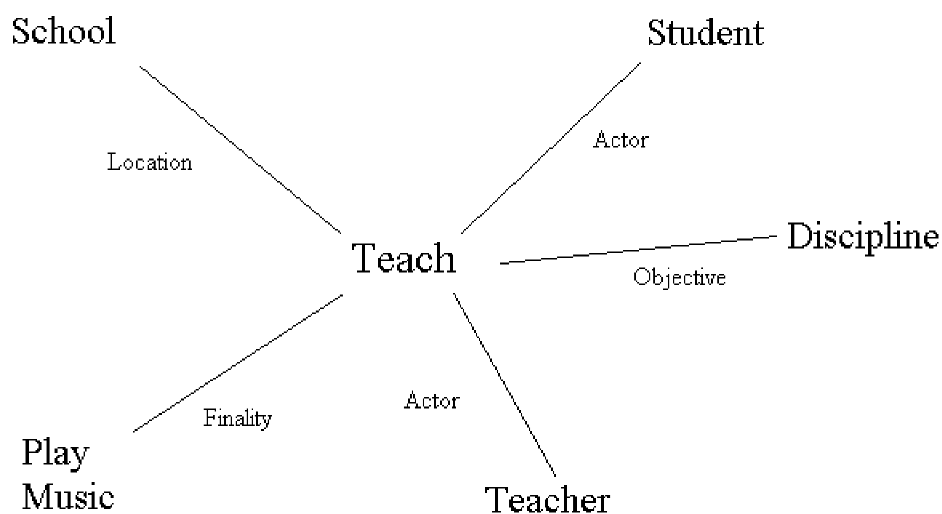

Figure 2. An event concept according to the HUG model (Klix, 1986).

Event concepts in the HUG model are related and defined by means of the following standard relations:

-ACTOR refers to any kind of active party involved in the event. Klix (1984a) notes that there can be a difference between an actor (The wind cries) and an agent (Mary cries), where the agent is a special kind of animated actor who is able to initiate the event or the actions at will, whereas the actor is merely a role-player. He leaves it to further research to determine whether this is a genuine difference in thinking. The number of actors follows from the valence of the event-verb. For example, selling involves both a seller and a buyer, whereas crying or cooking requires one actor only.

-The OBJECT relation refers to the 'what' element involved in the event relation with the actor(s). As the object connects the event with at least one actor, it has a valence of at least two. The classical example of an Object is the book being sold by one Actor to another.

-INSTRUMENTS refer to the special kind of things with which the event is brought about, the means by which the agent brings about the event. Instruments are not objects in the regular sense, but, in general they define the event relations more precisely. For example, one may plant a tree with a spade, ones bare hands or a dragline.

-The LOCATION relation refers to where the event is taking place. Klix (1984a) is not completely clear about it, but location probably also refers to relative place, such as in travelling to a destination. As such, while travelling one may reside in a train, moving from an origin to a destination location.

-the FINALITY relations refers to the purpose, if provided, with which the actor lets the event take place. Some people write articles for fun, others do so as part of a PhD thesis, and some for both. Note that although bringing about an event usually involves more than one final cause, in sentences and user interface commands the direct purpose is usually singular.

It may be noted that the exact differences between the categories are not always extremely clear, and also that categories are not always used. Finally, there are certain types of sentences for which it seems that the different categories should hold the same information. This may be taken to pinpoint that our mental grammar or grammar in the head (Payne and Green, 1986) is less precise than formal grammar and formal logic.

The relations in the HUG model are understood to be both universal and also specific with regard to the semantic event concepts they are linked to. Every event concept is characterised by this very set of relations, or a subset when particular relation instances are empty or absent. The event or core concept is specific because it 'defines' or limits the subset of properties of the concept at the other end of the relation. For example, in the case of 'cutting' there is always

an INSTrument involved, and more specific, always an INSTrument that is suitable for 'cutting' while an unsuitable instrument like a spoon seems to be excluded.
Klix and co-workers (e.g. Klix, 1984b) do not present any direct empirical evidence to support their particular choice of categories. This would indeed be very difficult, if possible at all. Given that in terms of among other the brain's hardware, every cognitive concept is in one way or another related to every other concept, it is in principle impossible to measure which individual one is most basic. Instead, it is possible to see how well the whole conceptual framework is able to explain empirical and theoretical phenomena in comparison to other theories, using indirect empirical evidence from the predictions derived from the framework.

Klix (1984a) does not exclude the possibility that apart from the ontological categories such as ACTors, LOCations, etc. more categories or refinements may be found or required. However, compared to other models such as LNR (Norman and Rumelhart, 1975) and models based on the notion of scripts or schemes (Schank and Abelson, 1977) the HUG approach will be more economical. Because HUG is able to account for relations within concepts (object-attribute relations) and relations between concepts (event relations) at the same time it will need less additional concepts in order to apply it to modelling phenomena in general.

Regarding indirect empirical evidence, in Klix (1984b, 1986 and 1989) a number of experiments is reported which provide indirect evidence for various parts of this theory. For example, Klix (1984b) reports a study showing evidence for a close relation between HUG's categories and the development of cognition in children. Another experiment in Klix (1984b) reports order effects in pupillometric measures among different categories in word recognition experiments.
Finally, whereas models like LNR and script theory have been applied to various empirical phenomena and backed by anecdotal evidence reported by the authors, the HUG model has been subjected to a number of experimental tests in which it showed to be valid.

With respect to ETAG, it is also important that Klix (1989) mentions that the event concept relations are not only related by the universal common links. From the standard set of relation concepts it is also possible to create more complex concepts which signify the relations between those concepts. Concatenation of concepts, for instance, can represent what happens in mental processes such as deduction and analogous reasoning.
Event concepts can also be related at a higher level of abstraction by means of semantic relation concepts like condition, causality and temporal follow-up. In this way, relations between concepts or rather concept instances can be described in terms of time and space. Since this aspect of HUG has not been worked out very well, ETAG's pre- and postconditions and the concept of states have been derived from Sowa's (1984) and Jackendoff's (1983, 1985) work.
Of the twelve kinds of relations Klix (1984a) mentions, the relations Attribute, Object, Event, and Location are directly used in ETAG. The Type relations and the notion of Actor are used implicitly in ETAG. In order to model CSCW applications using an extended version of ETAG, van der Veer and Guest (1992) make explicit use of the concept of Actor.

3.5 Modelling Reasoning

Regarding reasoning processes, Boole stated in 1854 that logic is the sole language of human thought. According to this doctrine, thoughts are in one way or another mentally represented as logical clauses, and inferences are drawn on the basis of the form of the propositions by applying the rules of logic. As such, the rules of logic are the rules of thought. Boole did not deny that there are, indeed, errors and logical fallacies in human thought, but the point is that the principal mode of thinking is about the form of propositions and guarded by the rules of logic. Errors and illogical thoughts are merely externally caused deviations of the regular thought. According to Boole, but for example also according to Aristotle (ca. 340 bc), and especially rationalists like Descartes (1637) and Leibniz (1686) the fallacies and errors in common sense reasoning are the mere result of a failure to think clearly, use the proper method, or 'noise'.
In its most extreme form, Boole's ideas have generally been rejected in psychology because of more or less analytic and experimental evidence against them. One argument Sowa (1984) mentions against 'logic thought' is that the truth of compound propositions only depends on the truth of its parts and not on their meaning. As such, it is not illegal to consider the following two statements to be true:

"If Socrates is a monkey, then Socrates is human"
"If elephants have wings, then 2 + 2 = 5"

Likewise, although on the basis of the meaning content it is obviously false that:

"A unicorn is a cow"

it is completely legal to state that:

"It is false that x is a unicorn and a cow"

Finally, in formal logic the syntax and the use of variables differs from what people consider to be natural. An English sentence like:

"A (some) girl screamed"

is a most simple sentence. Yet, logically, it is represented with variables, and, as a compound statement, with a conjunction:

"There is an x for which it is true that x is a girl and x screamed"

or, formally:

"$\exists x\ (Girl(x) \wedge Screamed(x))$"

Miller (1956), in his well known and often misquoted article mentions experiments showing that the immediate or short term memory span does not depend on the logical (binary)

information content of the stimulus material but depends, among other, on the effectiveness of the recoding schema and the resulting number of 'chunks': it is much easier to remember an octal or decimal digit than it is to remember the same information as a binary digit. Also, experimental research on reasoning (see Wason and Johnson-Laird, 1972; Evans, 1983) has shown that in making logical inferences, people process meaningful propositions in very different ways, and generally much better than meaningless propositions. In solving meaningful deductive reasoning problems, for instance, human subjects can be shown to have, or alternatively not to have, access to the results of intermediate steps which are logically required to reach the proper conclusion.

Although, in its extreme form, Boole's idea that mental behaviour proceeds according to the rules of formal logic has been refuted, in a weaker form a principle that might be called 'the principle of rationality' functions as a cornerstone in cognitive psychology. Often used as a hidden assumption in several theories of semantic memory, including ETAG, this principle states that, although human thought is not governed by the rules of logic that operate upon different forms of propositions, there must be a general kind of mental representation of knowledge that is acted upon systematically by discernible rules. In Jackendoff's terms:

> "It is assumed that there is a level of mental representation that (1) encodes the meaning of linguistic expressions, (2) permits a formal account of linguistic inference, and (3) serves as an interface between language and other mental faculties" (Jackendoff, 1985, pg. 158).

In plain English, this essentially means that Jackendoff and Sowa seek a theory similar to Boole`s except that they seek formal laws of thought which can handle the meaning of mental representations to replace classical predicate logic which only deals with their form.

Chomsky's transformational grammar formed the second incentive behind the theories of Sowa and Jackendoff. According to Chomsky (1965) all utterances start with thoughts, which by means of semantic rules are turned into Basic Structures which represent the meaning of the message to be conveyed. According to a number of semantic-syntactic and syntactic transformations (hence: transformational grammar) the Basic Structure is transformed into a Surface Structure. The Surface Structure is subsequently transformed into an utterance by applying phonological operations which add stress and intonation. As this process transforms thoughts into utterable sentences, Chomsky's grammar is a Generative grammar. In the opposite direction, the same processes are used to generate meaning from external utterances.
There are a number of problems associated with theories like Chomsky's. Apart from technical linguistic problems, which are too specific to discuss here, there are psycho-linguistic problems and the problem that Chomsky's ideas are based on formal logic.

The main psycho-linguistic difficulty is that the process between understanding and the production and perception of utterances does not seem to follow a simple sequential "waterfall" model of processing stages. Apart from the problem that specific Chomskian transformations do not always appear to fit human behaviour, there is evidence that actual language processing involves simultaneous top-down and bottom-up components. In Stroop tasks (see: Preece et al., 1994), characteristics of a word at different levels of processing can

be shown to interfere with each other, such as, for example naming a colour word when the word itself is printed in colour. To explain why character and word recognition is much better when the stimuli are placed in the context of a word or sentence (see e.g. Marslen-Wilson, 1980), it is also required to assume connections between non-adjacent levels of language processing. Finally, in 'shadowing' tasks, the process of (quickly) repeating meaningful spoken sentences appears to be largely independent of the process of understanding the same sentences. This, but also other evidence strongly points out that, opposed to Chomsky's views, there are connections between non-adjacent levels of processing, and that language understanding and production processes are not solemnly connected by the level of meaning.
Thus, theories which are to replace Chomsky's standard theory of language understanding and production should allow for more diverse interconnections between the various levels of processing.

Another group of problems, strongly related to the psycho-linguistic ones, has to do with logic as the basis of Chomskian grammar.
First, it must be noticed that Chomsky hardly says anything at all about how meaning is represented. Although Chomsky's theory is considered a cornerstone in psycho-linguistics, its top level rules which describe the alignment of syntax with semantics are far from complete. Since a treatment of the representation of meaning is lacking and since the rules connecting syntax and semantics are incomplete, Chomsky's theory should first and foremost be regarded as a theory of grammar. Sowa and Jackendoff on the other hand especially stress the importance of investigating the higher level processes at the semantic or conceptual level.
Secondly, insofar as Chomsky treats the meaning of language, he seems to assume a kind of predicate logic. In this respect it may be added that the transformational rules in Chomsky's grammar apply to the different forms or structures of sentences, in most cases completely independent of their meaning. The fact that Chomsky's theory does not distinguish between true, false and absurd sentences is perfectly acceptable from his point to describe human competence in linguistic behaviour. It is, however, problematic when aspects of the theory are interpreted as truthfully representing human cognitive behaviour in general.
A theory which is to supersede Chomsky's theory should at least express that, in language, meaning considerations precede over form considerations. Not only as a prerequisite to explain why language is often not used in a grammatical manner, but especially as a prerequisite to explain why language is generally used to express truthful meaning. This brings the argument back to the problems related to the fact that logic is based on the form of propositions only, without regard for the meaning of propositions and conclusions.

In order to solve the logical and psycho-linguistic problems, Sowa and Jackendoff, independent of each other, seek to develop theories which are centred around the meaning of thoughts and expressions, instead of merely their forms. In addition, their proposed theories are meant to be able to account for the fact that processing meaning, as in drawing inferences and producing and understanding language, is not one simple sequential process, but rather a complex set of overlapping processes at different but connected levels of abstraction.

In principle, the solution of Sowa and Jackendoff is quite simple. They adhere to Chomsky's idea that at some level thoughts or Basic Structures are represented as units of cognitive behaviour. Instead of following Boole's and, implicitly, Chomsky's ideas that these are

represented as propositions which are operated upon by the rules of logic, Sowa and Jackendoff borrow a special form of logic, existential logic, from Peirce (quoted in Sowa, 1984) to serve as both a means of representation and as the language of thought. In Peirce's existential logic, knowledge is represented in conceptual graphs, in which meaningful concepts are related by way of natural (as opposed to formal) relations. In conceptual graph theory, all concepts and relations eventually derive from Basic Concepts or Ontological Categories which are the most primitive as well as general categories, required for any knowledge or language, much like the analytical categories from Kant (1881).

3.5.1 Ontological Categories for Knowledge and Reasoning

In terms of intention or meaning, Ontological Categories are, by definition, the most primitive and basic elements of human language and thought. In terms of extension or the set of 'things' referred to, Ontological Categories are the most general concepts. Because Peirce attempts to simultaneously account for the extension and intention of concepts, it can be regarded as a form of reasoning with meaning.
Especially in Jackendoff's work, the identification of ontological categories is prominent. Sowa mentions and uses the categories, and focuses on formally defining the logic or rules in using conceptual graphs in reasoning about and representing reality.
In order to create an ontology, auxiliary concepts of Type and Token, and the relation Is-an-instance-of are required. With these concept-type rules, Jackendoff can identify ontological categories like: Thing or Object, Place, Path, Action, Event, Property, State, Manner, Number, and the concepts of Go, Be and Stay.
The first concepts all refer to singular concepts, and some can be turned into functions, such as Place and State functions which relate a place or state to an object. In order to describe in a general way how, e.g. two objects are related, the concepts of Go, Be and Stay are used, in combination with the name of a singular concept which serves as a label indicating the specific type of relation, as in:

stateBE(objectCAT, objectCHAIR)

It may be noted that in order to completely describe the meaning of a sentence or conceptual graph, the concepts Go, Be and Stay may need a further label to distinguish between their positional, possessional and identificational use. Compare:

the cat is on the chair
the cat owned by the chair
the cat is still on the chair

Sowa uses a slightly different nomenclature, and uses for example Concepts instead of Jackendoff's Types, and Entities for Things, and interprets Labels somewhat differently but, basically, they mean the same. Neither Jackendoff nor Sowa regard such a list as necessarily limited. In this respect, Jackendoff mentions for example Sound, Smell and Time as categories of lesser importance. In figure 3 several examples of related canonical graphs are presented.

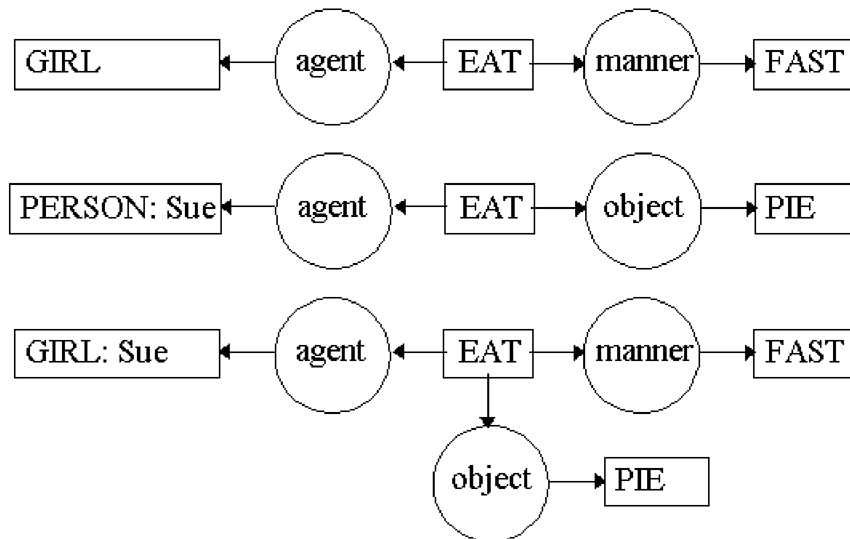

Figure 3. Examples of Conceptual Graphs (Sowa, 1984).

Ontological Categories serve as a priori concepts in both language and knowledge and can be found or established by analytical methods. Both mention that the categories can be exemplified by, and derived from, wh-word questions. For example, the question "What did you buy" exemplifies asking for a Thing, whereas the question "Where is my coat" is asking for a concept of the Place type. In a similar manner, different wh-type questions presuppose the other concept types.

3.5.2 The Structure of Knowledge

It may be noted that there is a strong relation between the conceptual graphs in Sowa's and Jackendoff's work and the representation of event concepts in the HUG model of Klix and colleagues. Since conceptual graphs can represent any kind of concept, they are more general than HUG's framework for event concepts. Apart from this, however, conceptual graphs are essentially similar to HUG's event concepts in functioning as a semantic core in mental representation.

There is also a remarkable similarity between the concepts that Sowa and Jackendoff mention as ontological categories and HUG's event concept relations. Apart from naming differences between the three theories, they seem to share the most important concepts of Object, Attribute, Location, Event, etc.

In addition to deriving the basic ontological concepts, a complete ontology needs various kinds of formal rules. This aspect is treated extensively by Sowa (1984). As this is a highly technical subject matter and not of particular relevance to ETAG, it will only be discussed briefly here.

First, rules are needed about how to relate different conceptual graphs, including the use of quantifiers.

Second, rules are required to describe which relations between types, labels and instantiations are valid. Where it concerns the 'formal' side of e.g. instantiation, there is no problem at all. On the contrary, it is most useful that definitions can be used to replace complex and recurring graphs by labels, or ETAG's use of hierarchical definitions of Types.

Where it concerns reality, however, the same problems of classic logic occur and, in a certain sense, they are more prominent. Conceptual graph theory is still a kind of formal system which represents reality, and as such requires some kind of translation to apply it to real problems. For example, it remains non-trivial how to represent a family resemblance among objects. Although Jackendoff proposes to seek special syntax rules, like e.g. a preference rule scheme, and Sowa throws in a large number of definitions (and assumptions), it remains to be seen if suitable solutions can be found.
Finally, there is the problem that it is often difficult to discern the rules of human cognitive behaviour. For this purpose, Sowa and Jackendoff propose to assume that the mental representation and processing of language of thought in terms of conceptual graphs takes place within a semantic network, in which also other than purely logical or formal processes occur.

Adopting existential logic and conceptual graph theory solves the problem that formal logic only deals with the form of propositions, but not their meaning. It does, however, not solve the problem of the often illogical human cognitive behaviour. In order to account for the phenomenon that human knowledge is practical rather than logical or truthful, Sowa and Jackendoff assume that conceptual graphs are embedded in a semantic network. In semantic networks, the nodes represent meaningful concepts, such as objects, sentences, words, events, etc., each connected with all other concepts with which a meaningful relation exists or can be established.
For practical purposes, the relations are generally chosen from a small set, but in principle they can include anything from very specific to very general and from concrete to extremely abstract like imaginative and associative connections. Semantic networks are only limited by what can be experienced, mentally or physically and, as such, they are a superset of logical and conceptual graph nets. On the other hand, semantic networks are not arbitrary, because they are limited to what is, or can be meaningful to humans.

To study how Long Term Memory is used in humans, different researchers have focused on different aspects of semantic networks, as these are assumed to underlie all kinds of memory stores and processes. Here, it is important to note that even though there is only one semantic memory, it can be used in many different ways. For example in playing chess (de Groot, 1966) and mentally manipulating objects (Shephard and Metzler, 1971) people employ highly spatial mental 'pictures'. In dealing with episodic material, such as in understanding stories, people can be viewed as using experience-based structures called frames (Minsky, 1975) or scripts (Schank and Abelson, 1977).
Perhaps even more important is to realise that people are able to use representations which, logically speaking, exclude or contradict each other. For example, Smith, Shoben and Rips (1974) and Collins and Quillian (1972) proposed semantic feature models to explain that in comparing e.g. bird names, the pattern of reaction times or the subjects similarity ratings closely follows the logical similarity in terms of features or attributes. On the other hand, Rosch (1975) has shown that categories are not logically structured, but rather organised by means of family resemblances around prototypes.

3.5.3 Human Reasoning

Assuming that conceptual graphs are embedded in a semantic network releases human thought from the restrictions to be realistic and logically valid. Unicorns are not -yet- shown to exist, and, as a combination of otherwise real attributes they cannot be banned from mental behaviour. Likewise, the conclusion that "swans are white" is a logical fallacy (Popper, 1959), but it is not an illegal thought as experience generally proves it to be true.

The notion that reasoning takes place within the setting of 'folk psychology' or everyday mental behaviour solves the problem of how a content oriented logic might work, and it provides explanations why everyday experience creeps into logical thinking. It does, however, not guarantee that a complete, working and truthful logical system can be built. For that purpose it is necessary to describe and define the relations between aspects of reality and conceptual graphs in sufficient detail. Jackendoff and Sowa have to finish their description of these so-called "laws of the world", which are assumed to apply to each conceptual graph. It remains to be seen whether anybody will ever be able to finish this work.
For the purpose of ETAG, there is no need for such a complete and exhaustive logic. ETAG is meant to represent user interface designs, and, as such, the area of application is both very small, abstract, and artificial.
First, because ETAG is not meant to describe real users there is no need to be able to describe the knowledge of specific individuals and, therefore, a general, but psychologically valid taxonomy of things in reality will suffice. Because ETAG only deals with e.g. the general concept of a computer file as storage space, there is no need to consider how Tom, Dick and Harry view files. This general taxonomy or categorisation system is exactly what Sowa, Jackendoff, but also Klix and colleagues have presented as the core of their work.
Second, ETAG is only meant for user interface design, which is not only an extremely small part of reality, but also a highly artificial and human-made part of reality in which all, or almost all elements are defined very precisely. Apart from the physical implementation, even to a computer user, a computer file is just a computer file and always a thing with a name on a particular place which may contain text or executable code, etc. Computer files and other elements in the field of user interfaces can be defined straightforward without the problems of e.g. vagueness and family resemblances.

3.6 The structure of ETAG in terms of psychological considerations

In this section the structure of ETAG will be briefly exemplified, guided by the Clipboard functionality of a hypothetical graphical user interface. The top-level part of an ETAG specification is the Canonical Basis, which is directly modelled after Klix (1984a), Sowa (1984) and Jackendoff (1983, 1985). The canonical basis lists and defines the concepts the user may get into contact with using the application at stake, such as objects, attributes, and places, without, however, referring to the particular computer system. The canonical basis describes the concepts needed to acquire knowledge about a particular part of reality, but only in terms of general knowledge of the world. Figure 4 presents part of an example canonical basis.

```
CONCEPT    ::=    OBJECT | PLACE | EVENT | ATTRIBUTE | STATE
[PLACE]    ::=    place.IN [OBJECT] | place.ON_TOP [OBJECT]
                  |  place.ON_END [OBJECT]
```

```
[STATE]        ::=    state.IS_AT [OBJECT, PLACE] | state.IS_ON [OBJECT, PLACE] |
                      state.IS_IN [OBJECT, OBJECT] | state.HAS_VAL [ATTRIBUTE]
[EVENT]        ::=    event.DELETE_ON [OBJECT, PLACE]
                      | event.COPY_TO [OBJECT, PLACE]  |  ....

type[EVENT  =  event.MOVE_TO [OBJECT: *o, PLACE: *p] ]
        precondition:     state.IS_AT [OBJECT: *o, PLACE: *p0] ;
        clears:           state.IS_AT [OBJECT: *o, PLACE: *p0] ;
        postcondition:    state.IS_AT [OBJECT: *o, PLACE: *p] ;
end
   ....
```

Figure 4. Part of a Canonical Basis.

The canonical basis in figure 4 explains that in the universe of discourse there are concepts, such as objects, places, etc., and that objects can be (in the state of being) at a place or in another object, and so on.

The syntax of ETAG follows standard conventions of formal representations, wherein "*p", "#p" stand for "any" and "one particular" instance of variable p. It may be noted that all the information in the canonical basis is implicitly present at the lower levels of an ETAG representation. Although the canonical basis can be left out of a specification, laying down that only such-and-such object-place relations exist helps to avoid confusion and inconsistencies.

The second part of an ETAG description is the User's Virtual Machine, which describes the specific concepts used in the user interface as type instances of the concepts of the canonical basis. The purpose of the UVM is to describe the (internal) workings of the user interface as a partly hidden machine. This corresponds to what Kieras and Polson (1985) call the "how it works" knowledge.

Logically speaking, the concepts of the UVM are conceptual graphs, directly or indirectly derived from the canonical basis according to the grammar of graph theory. In the opposite direction, they can be found through a logical analysis of a given user interface, if there is one.

For each type concept mentioned in the Canonical basis, the UVM has a separate type specification. In addition to the object, event, etc. type specifications, ETAG distinguishes a type hierarchy to show the inheritance relations between types and a spatial hierarchy which lists how objects provide space for each other. Figure 5 shows part of the UVM.

```
type [OBJECT = CLIPBOARD]
        supertype:        [SINGLE_OBJECT_BOARD] ;
        themes:           [STRING: *s] | [WORD: *w] | [PICTURE: *p] |  ..... ;
        instances:        [CLIPBOARD: #clipboard] ;
end

type [EVENT = CUT_STRING]
        description:      event.MOVE_TO [STRING: *s, PLACE: document]
                          event.DELETE_ON [STRING: *s, PLACE: document]
        precondition:     state.IS_AT [STRING: *s, PLACE: document],
                          state.HAS_VAL [STRING: *s, ATTRIBUTE: selected] ;
        postcondition:    state.IS_AT [STRING: *s, PLACE: clipboard] ;
        comments:  "move a string from a document to the clipboard"
end
   ....
```

Figure 5. Part of the type specification of a UVM.

The part of an ETAG representation which follows the UVM is the Dictionary of Basic Tasks. The dictionary lists which tasks may be performed with the user interface, and the arguments which should be specified by the user or the system. In ETAG, basic tasks are defined in terms of the system, and not in terms of the user (Tauber, 1988). The dictionary of basic tasks connects the semantics of the task world, as described in the canonical basis and the UVM, to the command procedures which invoke them, according to the production rules. Figure 6 describes the standard clipboard functions, in which the event entry provides the link with the UVM and the task entry, with an arbitrary number, the link to the top level specification production rules.

```
ENTRY 1:
task:     CUT_STRING
event:    CUT_STRING
system:   PLACE = document, PLACE = clipboard
T1 [EVENT = CUT_STRING, OBJECT = STRING: *s, ATTRIBUTE = selected]
comment: "move a selected string from a document to the clipboard"

ENTRY 2:
task:     COPY_STRING
event:    COPY_STRING
system:   PLACE = document, PLACE = clipboard
T2 [EVENT = COPY_STRING, OBJECT = STRING: *s, ATTRIBUTE = selected]
comment: "copy a selected string from a document to the clipboard"

ENTRY 3:
task:     PASTE_STRING
event:    PASTE_CLIP
system:   PLACE = clipboard, PLACE = document
user:     OBJECT = STRING: *s
T3 [EVENT = PASTE_CLIP, PLACE = EDIT_SPOT: *p]
comment: "copy the content of the clipboard into a document"
          ....
```

Figure 6. Clipboard Functions in the Dictionary of Basic Tasks.

The production rules of ETAG describe the command language of the user interface in terms of the feature grammar borrowed from TAG (Payne and Green, 1986). The production rules are specified according to four levels of abstraction, according to the levels involved in e.g. natural language production and understanding, as well as according to the four decision points in command language design.

For example, to change to a different command naming set will require changes at the lexical and keystroke levels, whereas changing from a command line interface to a graphical interface will also require changes at the specification and reference levels. Figure 7 shows the production rules to cut strings in a typical graphical user interface.

Specification Level - the order of specifying elements.

```
T2 [EVENT = CUT_STRING, OBJECT = STRING: *s, ATTRIBUTE = selected]  ::=
          specify[OBJECT = STRING: *s, ATTRIBUTE = selected]  +
```

specify[EVENT = CUT_STRING]

Reference Level - the style of interaction (e.g. pointing vs. naming).

specify [OBJECT = STRING: *s, ATTRIBUTE = selected]] ::= object_select [STRING: *s]
specify [EVENT = CUT_STRING] ::= menu_select [EVENT = CUT] | key_name [EVENT = CUT]

Lexical Level - the names of command elements.

object_select [STRING: *s]	::=	click [STRING: *s] + drag_over [STRING: *s]
menu_select [EVENT = CUT]	::=	click [MENU: "Edit"] + drag_to [Item: "Cut"]
key_name [EVENT = CUT]	::=	symbol [EVENT = CUT]

Keystroke Level - the physical actions involved.

click [ANY]	::=	press_mouse_button [ANY]
drag_over [OBJECT]	::=	drag_mouse_to_end_of [OBJECT] + release_mouse_button
drag_to [OBJECT]	::=	drag_mouse_on [OBJECT] + release_mouse_button
symbol [EVENT = CUT]	::=	press_key ["key-for-cutting"]

Figure 7. Production Rules to Cut a String.

3.7 Conclusions

This chapter discussed the psychological foundations of ETAG. ETAG is based upon the notion that the user interface does not end at the borders of the display screen, but encapsulates the system's functionality, or rather the universe of discourse as it is presented by the user interface with respect to the user's tasks. From this starting point it is relatively easy to arrive at a most fundamental 'law' of user interface design: if the conceptual structure of a system is what counts, the design should be based upon a specification of the conceptual structure. How this structure is eventually implemented in computer code, or a as collection of gadgets, bells and whistles is only of secondary importance.

> "Since the Macintosh, which was actually a step back in comparison to Lisa, GUI was understood by many as a problem of "look" (a matter of never-ending legal battles, as Pamela Samuelson documents in CACM 26, 4, April 1983). In reality, GUI is a conceptual problem; and so are the problems of its future. .... What is essential for UIs is the concept of computation they embody." (Mihai Nadin, 1993).

With this view in mind, Extended Task-Action Grammar has been devised as an extension to Task-Action Grammar (TAG, Payne and Green, 1986) which only covered the interaction language. Considering that ETAG specifications should not only be complete specifications of the user interface, but above all psychologically valid representations of a competent user's knowledge of such a system, this chapter discusses both surface or format considerations regarding validity, and deep or content considerations.
From TAG and other HCI models, two format considerations regarding psychological validity were adapted in ETAG. First, to describe the derivation of actions from tasks using a feature

grammar and to distinguish tasks on the basis of semantic features. Second, to use four levels of abstraction in the derivation of actions from task specifications similar to the levels of abstraction in the process of producing and understanding natural language.

The most important part of ETAG is the user virtual machine which completely specifies the conceptual structure of the computer system as far as the user is concerned. The user virtual machine specifies the universe of discourse of the users task world as it is internally represented in the computer system. In order to arrive at a description of the universe of discourse which is valid in human terms, special attention has to be paid to the choice of the canonical basis which, as it were, dictates this view.
On the basis of psychological research on the structure of semantic memory by Klix and colleagues, a discussion of the problems in theories of language production and understanding, and a solution proposed by Jackendoff and Sowa, ETAG is built around a canonical basis which defines the world in visual-spatial terms. This world consists of a collection of hierarchically related Objects which provide Spaces for one another and can be further distinguished by Attributes. Changes of the state of the Object world of the user interface are brought about by Events which connect the abstract description of the knowledge about the computer system's workings with the user's tasks and, eventually, using the dictionary of basic tasks and the derivation or replacement rules, with the concrete actions at the interface.

This chapter focused on the psychological basis of ETAG. A sufficient psychological foundation is, however, no guarantee that the model itself will be adequate as a design model.
ETAG is not the only model based on psychological considerations. A number of less sophisticated models have been proposed, such as Reisner's psychological BNF (Reisner, 1981, 1984), GOMS (Card, Moran and Newell, 1983) and Task-Action Grammar (Payne and Green, 1986), etc. All these models are based on one, or a few, psychological principles and they are not meant for design specification.
ETAG is neither unique as a combination of modelling the user and the interface at the same time. Moran (1981) proposed Command Language Grammar (CLG) as a model to represent (in three views) the interaction language, the user interface and a real user's mental model of the interface. In our view, Moran's targets were set too high. Also according to Norman (1983), mental models of individual users are much too idiosyncratic to be captured by a single, perhaps any, theoretical framework. Regarding CLG's use as a design model, it is complex due to its size and amount of redundancy. A study on design practise by Sharratt (1987) indeed shows that CLG's size is the main barrier for successful design. Apart from Moran's original paper and the study by Sharratt, we have not been able to find other applications of CLG.
Johnson et al. (1988) proposed a theory of Task Knowledge Structures (TKS). Although not a model, TKS as a method allows for user knowledge specification, and Johnson and Johnson (1991) present a-posteriori psychological evidence to support the theory and discuss issues related to its integration with software engineering methods. An interesting feature of TKS is the concept of roles as typical collections of user tasks. The concept of role has been adopted by van der Veer and others to create an enhanced ETAG for CSCW design purposes (van der Veer and Guest, 1992; van der Veer et al., 1995). TKS is used as the central model of a design approach (Wilson et al., 1993). In our view, TKS does not seem to present an integrated

model of the user interface or the user's knowledge. Rather, TKS may be said to consist of an activity model, derived from task analysis, that may be extended with information about object, features, roles, etc. In addition, Johnson and Johnson present psychological evidence to support the validity of the assumptions upon which TKS has been built, instead of TKS itself.

At least for the moment, it seems safe to say that ETAG is the only model which is, right from the start, built upon psychological considerations.

More in general it may be concluded that ETAG, as a formal model to specify user interface designs that is based on psychological considerations exemplifies and may be used as a "common language for delivering psychology to HCI" as meant in Green, Davies and Gilmore (1996).

This chapter is based on: de Haan, G. (1995). ETAG: the Psychological Basis of a Formal Model for User Interface Design.

Chapter 4:

An ETAG-based Approach to the Design of User-interfaces

But what satisfied me the most about this method was that, through it, I was assured of using my reason in everything, if not perfectly, at least to the best of my ability. Moreover, I felt that, in practising it, my mind was accustoming itself little by little to conceive its objects more clearly and distinctly, and not having subjected it to any particular matter, I promised myself that I would apply it just as usefully to the difficulties of the other sciences as I had to those of algebra.

Descartes, R. (1637).

Abstract

In ETAG-based design, user interface design is regarded as the incremental specification of the mental model of a perfectly knowing user. The design process is structured after the ETAG model into discrete steps, each covering a specific set of design decisions. ETAG-based design is an example of model-based design, and it shows what an HCI specific design, aiming at usable computer systems should look like. ETAG-based design exemplifies how theoretical considerations may be used to develop a psychologically valid and practically useful design method that stimulates designers to consider the usability of computer systems.

The first part of this chapter introduces some basic ideas underlying design in HCI, its problems, and the role that ETAG might play to resolve these. The second part describes the 'received view' on HCI design, and how it relates to some of the problems of user interface design. The third part discusses the view on users, tasks and the concept of the user interface underlying ETAG-based design, and it briefly describes the implications for the design method. The fourth part describes the steps that form ETAG-based design and it exemplifies how the ETAG formalism is used during the design process. Finally, ETAG-based design is compared to other model- and task-based approaches to user interface design, and conclusions are drawn about ETAG, ETAG-based design and model-based design in general.

4.1 Introduction

This chapter is about the larger picture of ETAG (Tauber, 1988, 1990) in relation to design. The ETAG-based approach to design is a combination of a number of specific ideas about users, tasks, the role of the user interface in performing tasks, and something one might call the received view on design in Human-Computer Interaction (HCI). The received view (or: paradigm) provides the general structure of the design process. It prescribes the main design steps and their ordering into a sensible and coherent whole, combining logical necessity with an evolved common understanding of good pursuit. The received view of a design science or engineering discipline may be identified as the greatest common denominator among the different approaches to design that are used within the discipline. There are two main issues connected to the received view on user interface design in cognitive ergonomics.

- First, according to Suppe (1974) the received view not only prescribes how things should be done, in this case, how user interfaces should be designed but the view also expresses which research subjects are relevant and worthwhile to investigate.

- Secondly, in prescribing how things should be done and how user interfaces should be built, the received view also expresses which design aspects should and which need not be addressed.

Regarding the issue of the relevance for research purposes, cognitive ergonomics may be characterised as constituting a huge collection of ideas, insights, theories and approaches about what is and what is not relevant but that a general notion about what is relevant and which research is required to develop cognitive ergonomics is lacking.
It has already been pointed out that to arrive at cognitive ergonomics as a science and engineering practice, one should focus on establishing a firm foundation of basic knowledge. Provided that regular sciences are characterised by the presence of only a few different approaches, cognitive ergonomics is still in a pre-paradigmatic stage. Monk and Gilbert (1995) distinguish, at least, ten different approaches to HCI, ranging from an applied psychology perspective to software engineering and organisational approaches.
In addition, Newman (1993) performed a meta-analysis on the main publications in cognitive ergonomics, such as ACM/SIGCHI's Computer-Human Interaction proceedings, and concluded that the majority of accepted papers is characterised by the concept of "novelty". From the point of view that for effective human-computer interaction and for the acceptance of cognitive ergonomic principles, instead of pursuing novelty, effort should be spent on the unification of the research field; a view that is underlined by the recent ACM/SIGCHI initiative to develop a research agenda for human-computer interaction (Scholtz et al., 1999).

With respect to the problem of relevance for design purposes, cognitive ergonomics provides a huge collection of guidelines, methods, heuristics and tools to support design aspects, to answer particular design questions or to support the design of particular types of interfaces. In addition, to the extent that a received view on user interface design can be identified as the common method for user interface design, it is not different from design methods from other disciplines.
Cognitive ergonomics provides ample facilities to support user interface design, such as standards, guidelines, and methods for analysis and evaluation. Even in the area of formal modelling, which constitutes the subject matter of this chapter, research has generally shifted away from design methods towards analysis of partial design aspects such as predicting user performance and measuring consistency of design products.
However, well-defined and documented design methods, such as they are available in, for instance, software engineering seem to be lacking. Provided that supporting user task performance is the primary aim of cognitive ergonomics there is a clear need for methods for task- and user centred design. Taking a conservative stance with respect to cognitive ergonomic design practice, first, a general task- and user centred user interface design method should be developed and only thereafter, methods should be developed to address any specific issues, related to the particular interaction types, technology, and context.

The two problems, the lack of a general paradigm for cognitive ergonomic research and the lack of a distinctive cognitive ergonomic design method form the background for proposing ETAG-based user interface design, as a small but hopefully a significant step towards establishing cognitive ergonomics as a systematic design science and engineering practice.

Science and engineering disciplines are supposed to have three characteristic features: a body of basic knowledge, (based on) a set of generally accepted assumptions, and a collection of standard methods. ETAG-based design is a method to solve the basic question of the HCI: how to design user interfaces that are usable for human task performers. With respect to the engineering part, ETAG-based design is a systematic and fairly rigorous formal method for specifying the most important parts of most user interfaces. It is neither an empty framework that is applicable to all types of user interfaces and design problems, nor is it an overly restricted blueprint to completely solve one particular problem or type of user interface design. With respect to the scientific part, in ETAG-based design, user interfaces are specified in terms of user competence knowledge that is expressed in a notation that derives from theoretical, psychological and psycho-linguistic considerations. ETAG gives a fairly complete psychology of competence knowledge representation with respect to computer task performance, and thereby provides a theoretical framework for research into the questions that ETAG-based design does not account for, by itself.

At the basis of ETAG-based design is Norman's observation that performing tasks on a computer as a general purpose tool lacks many of the affordances which signify in the external world what is and what is not possible (Norman, 1988). According to Norman (1983), when interacting with a complex device, users have to rely on a mental representation or a mental model of the workings of the device, to plan task performance and interpret the computer system's feedback. He further assumes that the completeness and correctness of mental models can explain the success or failure of using computer systems. Indeed, Bayman and Mayer (1984) show that the availability of a suitable mental model affects performance, and van der Veer (1990) shows that there is an inverse relation between the availability of graphical guidance (the affordances) and the development of mental models in users.
To design computer systems such that users may indeed form and run appropriate mental models to attain successful task performance, Norman suggests to use a conceptual or design model that considers user mental models as the basis for the complete design of the user interface, including its perceptual and training aspects. Although he gives no details about how to do that, there is a clear implication that computer applications should not merely offer a set of task commands, but rather an organised view of how to cope with tasks in general: a task world.

According to Butler et al. (1999), software design has historically emphasised completeness, consistency and the efficiency of computer code. As such, in software engineering it is common to use design models to ensure among others the completeness and correctness of the computer code that implements a user interface. Software engineering models are not useful as conceptual models in the meaning of Norman because they do not represent the user interface as the task-environment or task-world that the user should get acquainted with. Software engineering models represent the structure and workings of computer programs for the purpose of building them, not for the purpose of learning to use them. For the user, however, there is no such things as "the part of a computer program that handles the output to the display and the input from the person using the program" (Myers, 1994, pg. 2).

For users who want to perform tasks with a computer system, the user interface simply **is** the computer system, or at least, all the aspects of a computer system that are relevant for task

performance. When the user interface in the cognitive ergonomic sense is defined as "all aspects of a computer system that are relevant for performing tasks" it is something very different from the software engineering definition of a user interface.
First, whereas the user interface in the software engineering sense is a physical or functional object, what is relevant to perform tasks needs to be learned by users, via the information on the display, instructions or manuals, and as such it is a cognitive or knowledge object. Just like, for example, language understanding requires physical, lexical, syntactic, etc. knowledge, the user's "how-to-do-it" knowledge consists of knowledge at different levels of abstraction.
Secondly, whereas software engineering user interfaces exclude the workings of the computer system, to plan and evaluate actions users need to have "how-it-works" knowledge. To make use of "cut and paste" operations, for example, users have to know how a clipboard works (Payne, Squibb and Howes, 1990). Likewise, to understand that deleted files are no longer available for viewing, users need to know how a delete command works. Since the user's understanding doesn't need to be similar to the exact workings of a computer system (only the input and output should correspond) what the user needs to know about the workings of the computer system is called the User's Virtual Machine (Oberquelle, 1984).

User knowledge, both "how-to-do-it" and "how-it-works" is what ETAG represents, and in ETAG-based design, design is specifying what a perfectly knowing user would know about performing tasks. Specifying user interfaces in terms of user knowledge helps to keep focus on task performance aspects, rather than on technical software issues. Interface design in ETAG is formal to enable to take advantage of formality, such as precision and non-ambiguous communication. ETAG uses a layered representation of user interfaces since both human-computer interaction and user knowledge are layered along several levels of abstraction (conceptual, semantic, etc.), and to structure design into a number of discrete steps where each step aims at solving specific design questions, appropriate for the level of detail. Using one representation throughout the design process instead of different design representations particular to each design step helps to keep the design process simple and manageable.

ETAG-based design is a loosely top-down structured design method. Design iterations will especially take place within design steps, and only where necessary between steps. Structuring the design process in discrete steps helps to make the design process manageable, it provides guidance for a timely use of supplementary HCI methods and tools (e.g. Lim and Long, 1994), and it makes explicit where and how cognitive ergonomics and software engineering should communicate their respective results. The design structure of ETAG-based design is not very different from either the 'received view' of good HCI design practice (e.g. Rubinstein and Hersch, 1984, Shackel, 1986) or from software engineering design (see eg. Boehm, 1988; Sommerville, 1985; van Vliet, 1994). What is very different, however, is the separation of concerns between cognitive ergonomics and software engineering design and the almost exclusive focus on user knowledge that in our view should characterise the former.

Given that cognitive ergonomic design should be directed at creating user interfaces to enable people to successfully perform tasks, in terms of design stages, design should consist of task-

and context analysis, task design, conceptual design, perceptual design and possibly, but not necessarily, implementation, installation and usage. As a strategy to acquire this goal, we view design as the progressive specification of the knowledge users need to perform their tasks by means of the user interface that is being designed. As such, although the eventual goal of design is the creation of (usable) computer code, user interface design is concerned with user knowledge specification.

Finally, to design user interfaces according to the design strategy and structure, we use Extended Task-Action Grammar (ETAG) as the main design notation. Within our design approach, ETAG functions as a vehicle to urge or to persuade designers to consider users, user-tasks and task-knowledge in design. The approach is meant as both an umbrella and as a supermarket for design: it is a generally applicable method which, depending on the circumstances, allows one to include optional parts, or leave parts out.

4.2 The Received View on User Interface Design in HCI

In cognitive ergonomics many different methods to design user interfaces have been proposed. Suppe (1974) coined the term "Received View" to describe the common understanding of how scientific theories should be structured according to science philosophy. Here, the term will be used to describe the common understanding or, to abuse the word again: paradigm, of how user interface design should be structured according to cognitive ergonomics (e.g. Rubinstein and Hersch, 1984, Shackel, 1986).

It is characteristic of a received view that researchers and practitioners in a field of study implicitly or explicitly adhere to it. Individual researchers may stress very different aspects of the view, but in principle, research and application take place within its boundaries. The received view of design in cognitive ergonomics roughly consists of the following stages:

Task- and context analysis
Global Design
Detail Design
Implementation and Testing
Installation, Use and Evaluation

As it should be, the received view contains nothing new or unexpected and also ETAG-based design adheres to the view. In essence, the received view states that design starts with an analysis of the current situation and the needs to change it, followed by a top-down specification of the new system. The results of the specification process should be tested and prototyped along the way and, eventually, the specification should be implemented on a computer system and the impact of the new design should be compared to the original goals of the enterprise. Stated this way, it is clear that the view perfectly fits the practice of computer system design in software engineering, where the focus is on creating the software system rather than on creating a system that is usable for task performance (e.g. Davis, 1983; Hice et al., 1974).

The received view, as it turns out, has nothing to do with the specific design of interactive systems, user-centred systems design, design for usability, or whatever specific cognitive ergonomic terms are used. The received view in cognitive ergonomics is nothing more than a

general model for designing complex artefacts, regardless of the specific purposes and characteristics of the users of the product. On the one hand, the received view is able to capture all the different approaches to design in cognitive ergonomics as variations on a theme, yet, on the other hand, there is nothing specific cognitive ergonomics about the reviewed view to set it apart from other design disciplines.
In section 4.3 it will be argued that there are specific issues that distinguish cognitive ergonomic design from other types of design: design in cognitive ergonomics aims to support human task performance rather than e.g. machine task performance, it has to do with individual tasks and relations between tasks rather than with e.g. isolated tasks and, finally, it focuses on the cognitive characteristics of tasks rather than e.g. physical characteristics. The remainder of this section is dedicated to a more detailed discussion of the received view on design and the problems that are associated with it.

4.2.1 Task and Context Analysis

At the basis of design or redesign of a computer system there will always be a request, usually from the client (the funding party or the users) to create or change a work practice. Although human factors specialists may have been involved in formulating the request, which is a good thing, the decision to honour a design request is strictly a management matter which falls outside design proper.
Following the design request, in task and context analysis the old task situation or work context is analysed, if there is one, to understand the reasons why it needs to be changed. In HCI the result of this step is a semi-formal tree of the user's task structure in which the computer system plays an important role (Annett and Duncan, 1967; Diaper, 1989b; Johnson, 1992). Apart from the task structures, it is important to analyse the context in which the tasks were or will be performed in order to avoid designing a computer system which fits the formal work situation but fails to meet the unwritten or collegial rules about good work practice. From the nature of unwritten rules it follows that such an analysis cannot be completely formal. Although context analysis is essential, an in-depth analysis is not always necessary.
In both cognitive ergonomics and software engineering it is common practice to produce a document containing requirements and constraints which set the boundaries of the new system in terms of functionality, performance, human factors requirements, etc. (Maguire, 1997; IEEE, 1996). Requirement specifications can be arbitrarily complex, depending on the circumstances in which the new system has to be used, and the political strength of each of the stakeholders, such as users, computer system administration, managers and organisation- and human factors specialists.

4.2.2 Global Design

Global design is concerned with specifying a new system in broad outline on the basis of a formal representation of the problem domain and the requirements. Sometimes the more general term 'logical design' is used instead of global design, as in SSADM (Ashworth and Goodland, 1990). The result of this step is a conceptual model of the problem domain (the universe of discourse or problem space) and a model of the mapping of the description of the

problem domain onto the particular solution (the solution space). The term conceptual model means different things in cognitive ergonomics and software engineering. In software engineering the conceptual model is a representation of the problem and solution spaces in terms of data and software structures, and computer functionality. In cognitive ergonomics the conceptual model is foremost a representation of the user's task world and user task functionality which may considerably differ from those of the system.
In design on the basis of ETAG, global design is distinguished into two different design steps: *Task Design* is concerned with improving the task situation that is descriptively modelled during Task and Context Analysis, and *Conceptual Design* is concerned with conceptually modelling the user interface as part of the new task situation.

4.2.3 Detail Design

The focus of detail design is turning the logical organisation and behaviour of the system from global design into a technically feasible specification of the software. In software engineering this stage is concerned with designing the behaviour and internal structure of programs, program modules and software interfaces. Often detail design will, at least partially, coincide with implementation or coding. Traditionally, it is here that cognitive ergonomic design starts with dialogue design and screen layout. When human-interface design is treated as a part of detail design of the system, there is not much room left for usability concerns because the functionality and the structure of system as well as the data structures have already been laid down, leaving only the perceptual aspects of the system to provide the user with a consistent and coherent conceptual model of the computer system.
In ETAG-based design, during conceptual design the functionality and the workings of the computer system are defined. Detail design is then concerned with completing the user interface specification to allow it to be built. As a result, detail design is concerned with creating the perceptible aspects of the user interface: the interaction language, the presentation interface and documentation and training material. Because detail design is not merely concerned with filling in the details, but rather with creating the perceptual support for the conceptual design model, we use the term *Perceptual Design* for this stage.

4.2.4 Implementation and Testing

During implementation and testing all software specifications are translated into program code, compiled and tested to determine if it works properly. To make (part of) the system work, and to determine if it does so appropriately, test plans and test data must be prepared. Part of creating a working system consists of writing technical and user documentation and help functions. At this stage, testing, either as debugging or as usability testing, are done outside the regular working environment, to determine if the software behaves like it should, and to determine if users can easily perform their tasks. When it is at this stage that the first usability tests of the system are performed then it is inevitable that little can be done to repair them, other than by 'fixing' them in training and documentation. If the system is important for the organisation, it is common to prepare the installation of the system by having users trained and organising pilot sessions, in parallel to the regular use of the extant system.

4.2.5 Installation and Evaluation

The final stage of design is installation and evaluation. Installation involves more than simply installing the software on the destination machines. It may also include adapting the work context, including users themselves, such that users can smoothly perform their tasks with the new system. Especially the introduction of a large and important system requires careful planning according to a strategy that guides the introduction along a more or a less pathway (Eason, 1990). Poorly prepared installations (as well as poorly designed systems in general) can kill companies. Evaluation of computer systems is as often 'forgotten' as it is mentioned in text books as a necessary step in design. However, judged by the amount of effort spent on maintenance, skipping evaluation is not a very sensible thing to do.
In software engineering, evaluation is the second time that users are involved in the design process of a new system. During the requirements analysis step the users provide input to the design and during the evaluation step users engage in an acceptance test to determine whether the system lives up to its requirements. At this stage, any mismatches between the formal requirements and the originally intended requirements and any mismatches between the formal requirements and the newly evolved requirements are magnified and, as such, for cognitive ergonomics user evaluation is the opportunity par excellence to blame software engineers for not making timely use of human factors expertise; the too little too late argument (Lim and Long, 1994). In a more positive sense, Good et al. (1986) argue for the use of usability requirements as a means to shift the focus from the static properties of a system to the dynamic properties of using a system and, indirectly, as a means to get the usability of a system recognised as an equally measurable design criterion as other technical criteria.

4.2.6 Discussion

The received view has many problems. Its main problem, albeit implicitly, is that it takes a product-oriented view on designing computer systems, without taking the specific requirements of the users of the product into account. The received view provides a structure to design products in a stepwise top-down fashion, but its steps reflect a concern with making the design process manageable and following a logical order of design steps, rather than a concern with the question which design decisions belong together in order to create systems for human use.
A number of suggestions has been made to improve the design process, either by changing the structure of the process, adding user-oriented tools and requirements to the process, and by adding steps and processes for user interface design to the process.

First, it has been suggested to replace the sequential order of design steps (the step-by-step and the waterfall models) by a smooth incremental spiral process model (Boehm, 1988) or by a fountain model with discrete iterative steps. Interpreting a linear process as an iterative one does not guarantee that the process is performed in a qualitatively better way. Moreover, apart from creating opportunities to improve design, iteration also adds to the complexity of managing projects.

In the context of user-centred design it has been suggested that adopting design iteration as a main principle may help to solve usability problems (e.g. Gulliksen, Lantz and Boivie, 1999). Although it is clear that the waterfall model is not suitable to design usable systems because with our limited knowledge it is not possible to know in advance what the best design solutions are, merely replacing it by the requirement to iterate is of little help. Iteration without specifications of where, when and to what extent it is necessary, and without a specification of where, when and how many design resources are required introduces additional uncertainty.
Attempting to take usability more seriously by adding uncertainty to project planning may be seen as an open invitation to deal with cognitive ergonomic matter in an even more arbitrary way. Instead, we adopt from rapid application development (RAD) approaches the idea to structure design projects in time-windows (Martin, 1991) and propose a method that attempts to replace global iteration by local iteration. The purpose of reducing the scope of iteration is to increase the ability to manage human factors resources and, thereby, to bring human factors concerns on an equal footing to software engineering concerns, at least where it concerns resource allocation.

Secondly, there are many suggestions to improve the design process by improving the contents, or what has to be done at each design step. For example, it has been suggested to include continuous user involvement (Gould and Lewis, 1985), and to provide tools that have knowledge of users and user interface guidelines (Vanderdonckt and Bodart, 1993), and to use usability requirements that allow for objective measurement (Shackel, 1986).

It is beyond doubt that any means that facilitates the visibility, the accessibility or the measurement of cognitive ergonomic concerns is able to cause improvements to the design process. However, all the suggestions to improve the content of design steps still leave the effectuation to the designer. Prescribing that designers perform additional user-centred actions, such as demanding that user task requirements should be specified, does not necessarily lead to the design of more usable systems because they do not increase the amount of resources spent on usability and, most important, they do not require that software designers change their perspective from a software-centred to a user-centred perspective on the design.
What is necessary is a change of perspective: the purpose of design is not to create working systems, but to create systems that people can work with. Design is not about producing program code or software specifications, but about supporting the users' work. As such, we expect more from methods in which "designing systems for people to use" is inherently provided for, like in specifying user interfaces in terms of the user's knowledge.

Thirdly, there are a number of proposals to employ user interface design methods in parallel to the software design process or to use object-orientation as a means to integrate software and user interface design. With respect to using parallel design processes, Summersgill and Browne (1989) propose to add various "things to do" to the stages of SSADM and to require several user-oriented deliverables. Lim and Long (1994) proposed MUSE*/JSD as a user interface design method adapted to, in this case, JSD to ensure a timely exchange of design information.

A special case of advancing user interface design is to integrate it with software design in an object-oriented approach (Jacobson, 1995; Jacobson, et al., 1999; UML, 1999). In object-oriented approaches software and user interface design share a common domain model and, to specify user interaction, there are facilities for scenario-like use-cases and interaction diagrams. In comparison to the structured methods a specification of the domain from the user's point of view is simply lacking, and the facilities to specify user interaction are as additional as they are in the structured methods.
Since use cases are a particular type of partial scenario and neither UML nor use cases address the user interface as a whole, from the point of view of cognitive ergonomics UML may be regarded as a step backward. However, Jacobson (1995) does suggest that user interface specification and prototyping should take place as early as during requirements analysis, in which case, an approach like ETAG-based design would be most useful in informing software design about usability concerns. To have user interface design take place during requirements analysis creates an excellent opportunity to solve the too-little too-late problem but, as a suggestion, it is not yet a formal or required part of unified software design process and without a continuous focus on usability concerns, it may lead to a "too-much too-early" problem.

Provided that these methods are used in an environment that is open to spend resources on user task performance and the additional expertise is available to use the methods, we may expect improvements. In general, however, prescribing that designers perform additional user-centred actions, such as demanding that user task requirements should be specified, does not necessarily lead to the design of more usable systems. A problem with the parallel and integrated approaches is that the design methods themselves without the user interface part are moderately to highly complex. Adding the user interface part not only increases the complexity of the whole process but it also requires the assistance of user interface designers or method specialist.
Furthermore, it is also necessary that the resources are available, even though professional user interface designers report shortages of time, money, and equipment as their most frequent problem (Bekker and Vermeeren, 1996). Finally, also these methods do not require that software designers change from a software- to a user-centred perspective on the design.

The conclusion is that the design of usable systems requires a design method that, in one way or another, ensures that in each design step concern with the usability of the system is stimulated. In ETAG-based design concern with the usability of the system is stimulated by specifying the computer system in terms of specifying user knowledge.

Computer system design as user knowledge design is a means to keep the focus on the requirements of the users, throughout the design process. When design is based on a specification of user knowledge, designers are implicitly requested to pay attention to usability thus stimulating that concern with the user becomes the primary focus as well as the norm during design. In ETAG-based design, concern is first and foremost with the user's knowledge and, more specifically, with the amount and the complexity of the user's knowledge in relation to the tasks to perform. ETAG says little to nothing about program code, software specifications or computer systems which are regarded as just the material means to embody the user interface design rather than the end that it is all about. Computer

system users are not there to serve the computer system, but the system is there to help users perform their tasks, a view that also needs to be reflected in user interface design methods.

4.3 Users, Tasks and Task Concepts.

ETAG (Tauber, 1988, 1990) and ETAG-based design are about users, user tasks and task concepts. ETAG-based design is meant to design the task world within the computer system, which the user has to learn in order to perform his or her external tasks. In creating ETAG, Tauber (1988) called the internal task world "the User's Virtual Machine" (UVM) to signify how the particular device should be operated with respect to a certain task domain. A UVM offers one of the possible views on the users' task world. External tasks are the tasks of a user in the real world, whether it is controlling a power plant, writing newspaper articles or keeping a shop's finances in good order. External tasks may not have anything to do with computers, but the moment they are performed with a computer, users will have to map between the external and internal task worlds.
Moran (1983) mentions in the ETIT model a mapping between the external task entities and commands and the internal ones. Removing a word from a letter should thus be mapped to double-clicking and pressing the delete key, when using a computer text processor. Payne (1987) goes a step further: not only command names and entities have to be mapped, but it is necessary to map from the "pencil, eraser and paper device" to the "keyboard, display and cursor device".

The need to map between devices is most clearly exemplified by the difference between primary and secondary tasks (van der Veer, 1990). Primary tasks are the tasks the user would like or has to perform. They can easily be mapped between the external and internal task world. As Moran noticed, external primary tasks, such as writing a letter or removing a word from it may require decomposition into several internal tasks, but, for the rest, mapping is straightforward.
Secondary tasks are the tasks that must be done on a particular (external or internal) device in order to enable primary tasks to be performed. For example, in order to start writing a letter, one should pick a pencil and paper, or power up a computer and invoke a text editor in insert mode. Only at that point can the primary task commands be mapped between the two devices in a more or less direct way.

Because task performance goes beyond a simple mapping of tasks and task entities, in ETAG-based design the user interface also includes device knowledge. More precisely, the user interface is defined as all aspects of a computer system which need to be known to perform tasks, as the (specification of) the user's task world which has to be realised, in one way or another, in a computer application. This definition goes beyond the more common, much too restricted definition of the user interface as the interaction language and the information on the display screen as exemplified by Myers (1994).

4.3.1 Views on the Concept of the User Interface

In order to determine what should and what should not be part of a conceptual model of a computer system, in terms of how much of the functionality of an application should be included, a definition of the term user interface is needed. To answer this question we will use the Seeheim model (Pfaff, 1985). The Seeheim model is an early architectural model of computer systems that relates the functionality of the system to the user and the user interface by way of a component that is in charge of the dialogue between the user and the system. The model originates from the argument that to design usable computer systems it is necessary to use a neat and flexible architecture to guide their design and implementation that allows for a separation of the application and the user interface components (Versendaal, 1991). Because the architecture according to the Seeheim model is still rather inflexible and monolithic, more recent object-oriented architectural models use the Seeheim components on a per-agent or per-object basis, such as in the PAC model (Presentation Abstraction and Control; Coutaz, 1987; Nigay and Coutaz, 1998). In addition the Seeheim model gives a technical rather than a psychological view but it does provide a clear view of the logical components between the user and the application's functionality (see Figure 1).

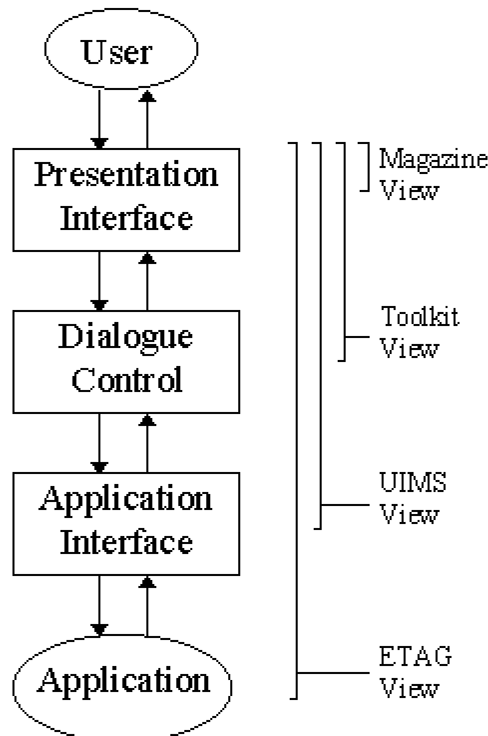

Figure 1. The Seeheim Model and User Interface Definitions.

A most primitive answer to the question what constitutes a user interface is the *Magazine View* that may be derived from (earlier) Software Engineering texts and hobby computer magazine adverts. Here, the user interface is defined as the combination of the input and output devices, and everything that is visible on the screen. Such a definition might suffice to sell software with adaptable screen colours as a user customisable user interface, but it lacks

an account of anything meaningful in the interaction between user and computer with respect to performing tasks. Following the Seeheim model (Pfaff, 1985) the term 'Presentation Interface' is used in ETAG to describe how the computer system presents itself to the user. In the ETAG formalism the Presentation Interface is not (yet) specified, although for a complete specification of a user interface, it should be. No suitable grammars for information display are available. In actual design there is no absolute need to specify the Presentation Interface within the same formalism because sufficient information can be derived from the ETAG model to use design it by means of separate tools. However, since ETAG is intended to be used in order to completely specify the user interface, a specification of the Presentation Interface should be part of ETAG.

A slightly enhanced *Toolkit View* definition can be found in the context of widget sets and user interface toolkits, where the user interface is regarded as the collection of widgets in the software library, such as buttons, menus and dialogue-boxes, their presentation on the screen, and the primitive, local (user) actions they understand, such as mouse actions and button clicks. Using this definition, Gilt in the Garnet system and InterViews' Build (IBuild) can indeed be called user interface builders by their respective authors (Myers et al., 1990; Vlissides and Tang, 1991). From the perspective that the user interface allows the user to perform tasks, this is still an incomplete definition of the term user interface since it does neither describe which tasks can be performed nor how they should be performed because the dialogue component is restricted to individual widgets.

User interfaces are most commonly defined as the combination of the presentation interface and the dialogue control, which completely defines the input-output level between the user and the computer device (e.g. Pfaff, 1985; Hartson and Hix, 1989). This *UIMS View* definition also takes into account which external user tasks are and are not provided for, and how these tasks should be invoked. From the user's point of view, the question which tasks are available is part of the user interface, but from the classical Software Engineering point of view, it belongs to the application's functionality. What users think about what an application may be used for, on the basis of what they see of it, may differ from what designers intended that the application should be used for.
We call this the UIMS (User Interface Management System) view, since that area first recognised the need to separate between the dialogue and the presentation components of accessing computer functionality. In ETAG-based design the UIMS view or the perceptual interface, that is, the combination of the interaction language and the presentation interface, is still an incomplete definition of a user interface.
Even if it is clear which tasks are available in the computer system, and how to perform them, it may remain unclear what such tasks (or commands) are meant for, in terms of the tasks the user would like to perform. In order to use cut-and-paste commands, an edit buffer (or clipboard), and even a file, it is not enough to know how they are specified in terms of the computer's command syntax. In these cases, a user also has to know or infer what an edit-buffer is and what cut-and-paste commands are used for, in terms of the semantics of the **user's** task domain.

In the *ETAG View* the user interface is defined as everything a user needs to know in order to perform tasks with a computer, which is a specification of the knowledge of an abstract,

perfectly competent user. Thus, instead of specifying a computer system in Software Engineering terms, as what it should do, and what it should look like, in ETAG-based design we ask what a user should know about it to use it. Note that ETAG is used to specify competence knowledge; as such, it does not specify how a real user will use this knowledge to actually perform tasks.

What is missing in the user interface as a combination of the presentation and the language interfaces, is a specification of the task semantics (the meaning of task commands and objects) of the computer system. Within ETAG, the view on the task world that the computer system offers is called the User's Virtual Machine (UVM, Oberquelle, 1984). It is a *User's* virtual machine to express that the computer system is only of interest for the user to perform tasks, and it is a user's *Virtual* machine because it is a specification in terms of user knowledge, which may be different from the actual technical workings of the machine. Note that the ETAG view is not restricted to ETAG-based design only, and has also been expressed in, for example, Dourish (1995).

The user interface in ETAG is everything a perfectly competent user knows about the computer system to perform his or her tasks, and it consists of the User's Virtual Machine (the machine's objects and functions) and the perceptual machine, which in its turn consists of the language interface (the interaction language) and the presentation interface (the display information). Figure 1 shows how the different user interface definitions fit onto the Seeheim model.

4.3.2 ETAG's View on the User Interface in Design Practice

With the definition that the user interface includes everything that is relevant to the user's task performance and, therefore, should include what the user needs to know about concepts and workings of the functionality of the application, it is exactly specified what information should be present in a conceptual model for the design of the user interface or computer system. From the point of view of the user, or the point of view of cognitive ergonomics, the user interface and the computer system are synonymous, whereas, from the point of view of software engineering the user interface is merely a part of the computer system. For design purposes, the conceptual model needs to be formalised (e.g. in ETAG) to create a correct and complete representation of the user interface, and, since such a specification is the result of the design activity, rather than its input, in ETAG-based design, design is regarded as creating a user interface specification or the ETAG model.

The ETAG model as a knowledge specification of the interface being designed differs from the mental model that users will develop when working with the system, and it may differ from the conceptual model of software engineering. Since the ETAG model is, according to Norman's (1983) distinction, a conceptual model, the actual user's mental model will not only differ in terms of completeness and accuracy but, in addition, even if a user develops perfect knowledge about the interface, the content and organisation of the two models will most certainly differ.

The ETAG representation of the user interface is a conceptual model to facilitate the design of the interface itself, and all other aspects of concern to the user, such as help and feedback messages, user manuals, etc. (in short: the metacommunication). ETAG need not be the only conceptual model used in design. In relatively simple and predominantly interactive systems

there is no need to use other conceptual design models. When the task world described in the ETAG representation has close relations to the tasks of other people and other task worlds it may be necessary to use, for example, models of business procedures or production processes to describe the context. When the technical structure that is required to implement the computer system is highly complex it may be necessary to use software engineering design to facilitate the implementation of the means to support the task world description. However, from the idea that computer systems are there in the first place to help users perform their tasks, in ETAG-based design it is assumed, as a principle, the ETAG model should always be used as the primary model.

4.4 ETAG-Based Design

ETAG-based design is best viewed as an umbrella or supermarket method for the design of user interfaces. It enforces, or at least stimulates a focus on the user's needs, provides a structured method to yield timely and communicable results, but leaves it to the preferences of the designer and the demands of the situation which faculties, tools or techniques, etc. are used. A basic idea underlying ETAG-based design is to divide the design process into discrete design steps, each asking a specific set of questions according to the part of the ETAG representation being worked on. As such, the design specification (the ETAG representation) structures the design process such that it may proceed in an orderly and timely manner. In addition, the different parts of ETAG are structured in such a way as to require few iterations between design steps, while allowing many iterations within design steps. Within each design step, it is completely up to designers to decide how to proceed.
Figure 2 shows the general structure of ETAG-based design. This structure is the result of a framework for the design of user interfaces in general, developed with van der Veer (van der Veer et al., 1995), but simplified and adapted to the ETAG notation in de Haan (1994). In the figure there is no specific box dedicated to ETAG-modelling since the specification is meant to be used throughout design, although ETAG modelling to represent the new design is concentrated in the box entitled "Conceptual Model".

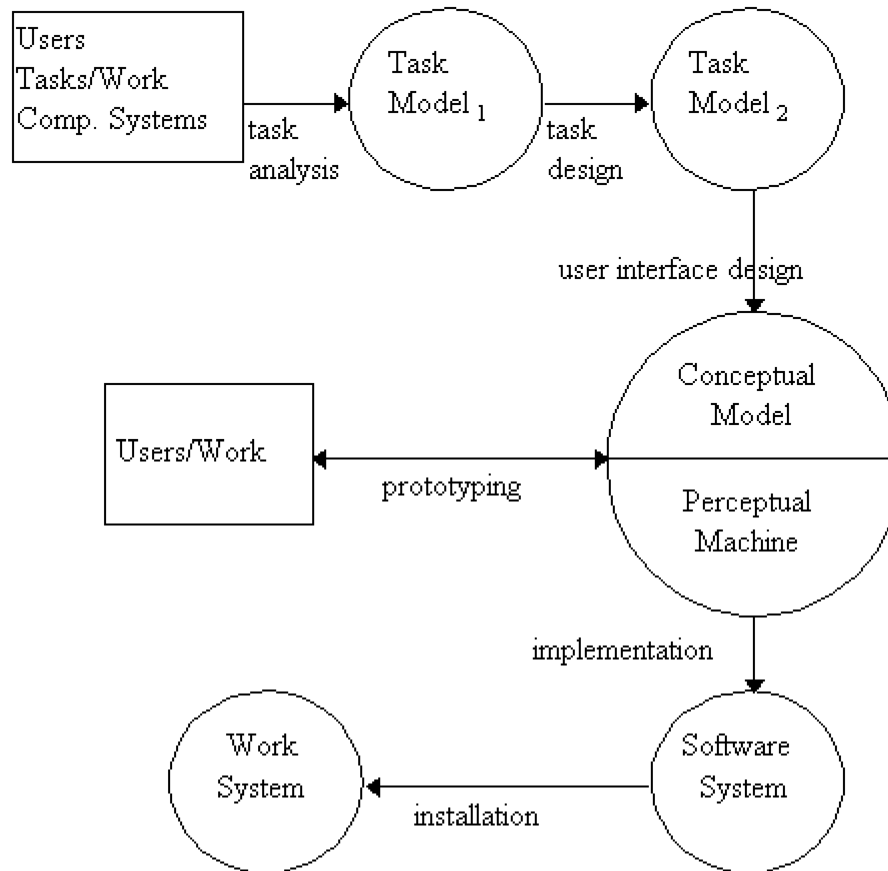

Figure 2. An Overview of ETAG-based Design.

4.4.1 Task and Context Analysis

As a starting point for ETAG-based design it is assumed that a request for (re-) design has been granted and a design team has been formed. The first design step is to get a sufficient understanding of the request within the context of the users, the user's tasks and the work organisation. To what extent contextual information is required depends on the nature of the design. ETAG-based design is a task-based design method. As such, it is assumed that task analysis information is always required and that context analysis information comes at the second place and may not always be necessary.

Information about the user's tasks that must be supported by the new design can be derived from the current or extant task situation. If, for some reason, there is no current work situation, task information can be derived from related task situations in which, for example, the same types of tasks are performed using different devices or competing products or other organisations.

To proceed in a systematic way, after an initial formal and informal familiarisation with the current task situation there should be a point at which it is decided which information to gather, how to do that, how much information and information details to collect, and how to present the results. Finally, it is necessary to decide about which results of the task and context analysis should be put in the system requirements specification, alongside information of a more technical nature, and about how specific these requirements should be. Human-

factors information is often said to be "too little too late" (e.g. Lim and Long, 1994; pg. 4-5). "Too late" refers to the situation when human factors input is called upon after the main design decision are taken, with the result that the input is "too little" to make a difference. Other reasons for too little human factors input are overly general requirements, and stating requirements which, irrespective of their number and seeming precision do not capture what is relevant to task performance. For reasons like these, Gould and Lewis (1985) argue for an early focus on the users of the system and their tasks. In ETAG-based design task analysis is used for this purpose, and in recognition that focusing only on the tasks may not capture all the relevant information, for instance when tasks cannot be readily identified, this design stage is referred to as task and context analysis.

The most important part of task analysis is the analysis of the user's tasks which are related to the computer system. A model of these tasks, the so-called task model 1 shows a functional or procedural decomposition of the user's tasks, laid out in time, and derived from observations and interviews of users. Its purpose is to provide a baseline to reason about how a computer system might fit in with the user's tasks, and which functionality, operating procedures, etc. should be provided. To reason about the user's tasks there need not be an existing task situation, just as it is unlikely that design projects aim to re-create existing design solutions. Rather, the idea is that task analysis is performed on existing systems that have relevant characteristics in common with the system that is designed, including less advanced, competing and, otherwise antecedent systems. According to a survey by Bekker and Vermeeren (1996), analysing similar applications is the most important source of information for user interface design, followed by a specification of the activities that are to be supported and interviews with prospective users.

In addition to providing a baseline, task analysis is also a means to guard against design conservatism. Contrary to those who argue that task analysis and modelling the existing work situation restricts designers from creating genuinely innovative user interface designs (e.g. Suchman, 1987; Bannon and Bödker, 1991), we think that an explicit representation of the existing work situation helps to avoid a main danger of designing from scratch, namely to create designs based on hidden and possibly false assumptions. For example, adding functions to an existing task model implicitly asks the question whether such functions are really needed. Without a model of the current task situation it depends on the individual insights and skills of the analyst. According to Smith (1998), in order to design new generation products, task analysis cannot help to identify new product opportunities only, but it also allows an understanding of the underlying functional reasons for a task beyond the surface procedures associated with it.

Even if we accept the view that modelling the existing work situation would not in itself stimulate designers to come up with creative solutions, and even if we hypothetically would accept the view that such modelling would rather stimulate conservative designs, it should be kept in mind that genuinely innovative designs are rare. Johnson et al. (1989) write in a retrospective of the Xerox Star, whose user interface is generally regarded as highly innovative, that it is not so much creativity that sets it apart, but rather the combination of existing techniques that resulted from an analysis of its assumed users and their tasks.

The observation/interview method may not always provide all required information because it is restricted to formal (verbalisable) and task related information. It may be necessary to use additional analysis methods such as work-flow analysis, participatory observation or

ethnographic methods to attain information about aspects of the work organisation that are difficult to make explicit and formalise.

For task analysis in ETAG-based design we follow an eclectic approach, using the standard observation and questioning methods to derive a hierarchical decomposition of the user's tasks and adding (aspects of) other methods and techniques where necessary. Standard task analysis consists of repeatedly finding out what a person is doing, and establishing how these activities fit together. After abstraction and cleaning up task analysis yields a number of task trees: graphically represented hierarchical decompositions of a person's main tasks into multiple subtasks and action procedures.

To facilitate task decomposition, we adopted from Sebillotte's (1988) Hierarchical Planning method, to ask How-questions to acquire information about the lower levels of a task decomposition hierarchy and Why-questions to acquire information about higher, goal levels.

When the sequence of tasks is important and cannot be expressed in the simple left-to-right ordering of task trees, we use the idea to add ordering-tags and And-Or diagrams to the task decomposition trees from Scapin's MAD (Method for Analyzing task Descriptions; Scapin and Pierret-Golbreich, 1989). Ordering tags and And-Or diagrams allow expression of, for example, alternation, priority, necessity and choice between subtasks and procedures. Ordering tags and And-Or diagrams are a very effective notation to increase the expressive power of a task decomposition tree, almost without increasing its size and complexity. Several tools are available for visual task model creation, such as Euterpe (van Welie et al., 1998b), a tool for task analysis, and ConcurTaskTrees (Paternò et al., 1997), a notation and tools to task model editing and code generation.

When tasks are not bound to individual jobs or people and may be performed by different people it may be useful to assign parts of task trees to Roles, such as, for example, when meetings may be organised by both managers and secretarial workers. We borrowed the concept of a Role from TAKD (Task Analysis for Knowledge Descriptions; Diaper, 1989c) to model sets of tasks and responsibilities that are shared among people. Like differences in expertise, the existence of roles may have implications for the user interface of the shared part of the computer system.

Regular task analysis focuses on activities rather than the things involved in performing tasks. In order to find out which objects may be relevant for computer system design and eventually to create ETAG's object specification, additional information must be gathered. Apart from plainly asking and observing users, we found it useful to create ETAG models (of parts of) the work context. Here we use the characteristic that ETAG is not bound to computers and computerised tasks, but may be applied to any device whatsoever. Creating ETAG models does not only yield information about the objects used in the user's tasks, but it is also useful to check the completeness of the task representations and to serve as a starting point for conceptual modelling later on in the design process.

The result of task analysis is called task model 1, the task model of the extant work situation. It should contain sufficient information about characteristics of the users, the work organisation and the user's tasks to proceed with task (re-) design, in the form of, at least, a task decomposition tree and information about the task objects.

Example: A manufacturer of automatic 35mm camera's receives complaints from its customers about a model that provides an automatic fill-in flashlight, regardless whether the use of the flashlight is switched on or off. As a result, the users have to depend on the camera to determine whether or not to use the flashlight in low-light situations.
The customers want to be able to put the camera in a state that under no circumstances a flashlight is used, either by changing the effects of the operations to switch the flashlight on and off, or by adding an operation to selectively switch the fill-in flashlight on and off. Note that the first solution does not change the structure of the task decomposition tree and only changes the effects of operations, whereas the second proposed solution adds an operation as an additional task leaf in the task decomposition tree.

4.4.2 Task Design

Task Design corresponds to the first part of global design in the received view, when it is applied to improving, rather than merely automating, the user's task situation that has been described in task model 1.
The overall purpose of the combination of task and conceptual design in ETAG is to create the user interface as a conceptual entity; that is, both as a view on and as a set of operations (basic tasks) to act upon the task world of the computer system. In Payne's (1987) terminology, the interface provides a device space, which is meaningful only within the goal space of the user's general task structure.

During task design a new task model 2 is created of the functional components in the future work situation, on the basis of the information in task model 1. The purpose of task model 2 is to enable reasoning about how the user interface will be used. It is difficult to specify exactly how this should be done, especially when it is not just the tasks but also the jobs that change. If user interface design concerns in some way a redesign of an existing work situation or a redesign of a computer system that is already in use, it is possible to use the old task decomposition tree from task model 1 as the starting point for task design. This also applies to situations where a 'would-be' old task situation is available in the form of competing products and well-understood application domains. If there is insufficient information available in task model 1, envisioning techniques can be used to reason about the 'would-be' task situation, such as user exploration (Kemp and van Gelderen, 1996), envisioning (Monk, 1998) and scenario techniques (Carroll, 1995).
When a task tree is available in the task model 1 it may be annotated with indications of things like frequency, duration, difficulty and the occurrence of problems. Starting with the higher level tasks, superfluous and a-typical tasks may subsequently be removed, newly required tasks added, and complex tasks decomposed or otherwise simplified to form the core functionality of the user interface: the set of tasks that reflects the purpose of the user interface.
From having established or identified what the user's main tasks will be, the process of removing, adding and simplifying tasks continues downward until all the main tasks are decomposed into more simple tasks which should allow for and be assigned to either the user or the machine. Eventually, all tasks should be decomposed into, for example, command procedures, it is better to stop at the level of simple or unit tasks. The aim of decomposing tasks is to arrive at a preliminary solution without, at this stage, laying down the specific details of the design. As such, further decomposition will only make it more difficult to improve the task tree and adapt it to the results of conceptual and perceptual design.

Sebillotte's (1988) categorisation of procedures as specific or not specific for a work domain may help to determine the domain-independent tasks or 'generic' commands (Rosenberg and Moran, 1985) which the user interface should provide. Generic tasks should be higher level tasks that decompose into domain specific subtasks.
The allocation of tasks between the user and the computer system aims at creating jobs that are stimulating but not overly demanding. As a general rule, control, initiative and decision making tasks in the work domain should be left to the user, while tasks that concern repetition, verification, and handling of details and memory should be assigned to the system. When the 'interesting job' criterion and user opinions are not conclusive, standard ergonomics texts provide ample guidelines (e.g. Sheridan, 1988; McCormick, 1976). The tasks allocated to the user-system combination provide an idea about what the dictionary of basic tasks might look like. Identifying basic tasks at this stage can also be used as a bottom-up approach to task design.
We use the term preliminary solution here to indicate that the conceptual structure of the virtual machine has not yet been created and formally analysed. When the preliminary solution results in an inconsistent internal task world, it may be necessary to change the decomposition, and possibly even override the user's choices. As such, iteration can be expected here.

Example: The first solution proposed to solve the problem of the fill-in flashlight is to change the effect of the operation "switch-off flashlight" from only switching-off the main flash whilst keeping the fill-in flash enabled to switching-off both types of flashlight. The solution requires that the effect of the operation "switch-on flashlight" changes from always using the flash to using a flashlight when necessary.
To the majority of users, who leave all thinking to the camera, this may create confusion, because now the flashlight may not work when it is switched on. Also because adding an additional operation would only slightly increase the dialogue complexity of handling the flashlight, the second solution is followed. According to this solution there will be three states: flash-on, flash-fill-in, and flash-off, and an additional operation and leaf in the task decomposition tree to choose the flash-fill-in state.

4.4.3 Conceptual Design

Conceptual design is the first stage of user interface design proper. It consists of specifying the canonical basis, the UVM and the dictionary of basic tasks: the user's view of the workings of the computer system.
The specification of the user virtual machine derives from the new task model 2, and in particular from the task entities, the preliminary dictionary of basic tasks, and, if available, (an ETAG representation of) the extant user interface. The UVM is a complete specification of the device space, or the task world of the interface, in terms of a competent user's conceptual knowledge.
It is important to realise that the UVM includes all the conceptual aspects of the functionality of the computer system as well as those of the user interface in the traditional software engineering sense of the word. Because users don't (need to) distinguish between functionality and interface aspects, but do require a consistent and coherent view on both, both aspects, as far as they are conceptual, should be designed together. This is a main difference with the software engineering design approach in which, during detail design, the functionality is already laid down when the design of the user interface takes place. In other words, in ETAG-based design both functionality and user interface aspects, insofar as they concern conceptual

matters, have equal precedence and are seen as equally important in shaping the user's view on the task world.

There is no algorithm for creating the virtual machine specification. Design and specification generally require a yo-yo strategy that combines a top-down approach, starting with ontological or otherwise abstract concepts, and a bottom-up approach, starting with basic and more general user tasks. In particular, when there is an current computer system it is possible to use guidelines to derive an ETAG representation from an existing user interface (Yap, 1990; de Haan and van der Veer, 1992a; chapter 7) and to change the representation to the new design. In other cases, it is possible to create an ETAG representation of the new design at once by using, for example, the guidelines for identifying the elements for object-oriented specifications (e.g. Coad and Yourdon, 1990) or the guidelines developed in the conversation analysis approach to human-computer system design (e.g. Greatbatch, et al., 1995).

It is easiest to start conceptual design with a list of user tasks. A main part of the dictionary of basic tasks, the primary tasks, follows from task model 2 and a low-level task allocation between user and the computer system. Part of the UVM can be specified by abstraction from the objects, relations, etc. of task model 2. The canonical basis can be specified by subsequently choosing from among the standard canonical basis concepts those that are required for the concepts of the UVM. In this way, those parts of the canonical basis, the UVM and the dictionary of basic tasks that are required for the tasks for which the computer system is actually designed can be specified. This is a specification of a minimal interface (Neerincx and de Greef, 1993): a service hatch that does not yet provide an integrated view on the user's task world. Creative design is now necessary to add the tasks and task concepts which enable the user to perform the tasks of the minimal interface in a uniform, consistent and intuitive way. As an example, when a minimal interface has a task to query-a-database, the full interface might have a task to select-a-database, a window object to present the results and possibly additional tasks to manage window objects in general.

Analytic methods can be used in the process of creating the ETAG representation, for example to keep the model as compact as possible, restrict the number of concepts of a particular type, or to limit the number of hierarchical levels of definition. Validated complexity measures, specific to the ETAG notations are not available but de Haan and van der Veer (1992a; chapter 7) and Olson (1993) discuss various possibilities. For in-house use, a prototype consistency checker has been developed (Bakker and van Wetering, 1991).
The results of the conceptual specification of the user interface may need to be iterated back to the task model 2 specification when new insights about the task situation develop during conceptual design. More likely is the situation that iterations are required between the specification of the UVM and subsequent design stages because of the one-to-one relation between the dictionary of basic tasks and the top-level specification of the interaction language.
Most iterations, however, and this is a practical reason to design all aspects of the conceptual model at once, will take place within conceptual design. In particular, the dependencies between the UVM and the dictionary of basic tasks will be important. For example, the choice whether the user or the system should provide particular task entities may influence the choice of concept features, and it may also be difficult to specify all secondary tasks at once. The

development of prototypes of the new design is a helpful means to speed-up the conceptual specification of the system and the co-ordination between the UVM, the dictionary of basic tasks and the interaction language.
At the point where the virtual machine, or the user's view of the system, has been specified in sufficient detail and stability, the system engineers will be able to create the software specification models of the new system and start building it. In contrast to the "too little too late" problem the completion of conceptual design does not mark the starting point for cognitive ergonomic input to software engineering but rather the point after which the need to co-ordinate cognitive ergonomic and software engineering responsibilities decreases. After a good specification of the virtual machine is created there is no need anymore for user interface designers to insist on major redesigns or to resign to patching up usability problems.

The results of this step, probably in the form of a layman's description of the virtual machine, should be discussed with users and, in a more formal sense, compared to the system's requirement specification. Last but not least, there is the possibility that the dictionary, and with it the virtual machine, may have to be adapted to the interaction language.

Example: In terms of ETAG's UVM, the solution to the flashlight problem is to change the state of the flashlight from two states: flash-on and flash-off to three states: flash-on, flash-auto, and flash-off, and to adapt the event to set the state of the flashlight accordingly. As such, the relevant part of the old UVM reads:

```
type USE_FLASH isa ATTRIBUTE
    value_set:          {On, Off}
    remark: "use-flash off really means: main flash is off and fill-in flash enabled"

type SET_USE_FLASH isa EVENT
    system parameter:        USE_FLASH is ONE-OF {On, Fill-in}
    precondition:            POWER-ON is true
    postcondition:           USE_FLASH is Next [ ONE-OF {On, Fill-in} ]
```

The new UVM reads:

```
type USE_FLASH isa ATTRIBUTE
    value_set:          {On, Fill-in, Off}
    remark: "use-flash auto means that main flash is off and fill-in flash enabled"

type SET_USE_FLASH isa EVENT
    system parameter:        USE_FLASH value: ONE-OF {On, Fill-in, Off}
    precondition:            POWER-ON is true
    postcondition:           USE_FLASH is Next [ ONE-OF {On, Fill-in, Off} ]
```

Because the proposed solution only involves the addition of an attribute (and one "fill-in flash" that may be presumed to be well known) with no additional states or events, the usability consequences of the change will probably be very small.

4.4.4 Perceptual Design

Perceptual design of a user interface consists of specifying the perceptual interface (the presentation interface and the interaction language) and metacommunication facilities (user documentation, user training and on-line help). At this level, and perhaps with the inclusion of the dictionary of basic tasks, the design process has arrived at the traditional definition of the

user interface as the observable features of a computer system, which Norman (1983) calls the system image. Curiously, even in HCI people hold on to this definition (e.g. Systä, 1994).
The system image (Norman, 1983) or 'everything the user comes into contact with' is the object of perceptual design. The perceptual interface consists of the presentation interface, the interaction language, and the metacommunication facilities (e.g. user documentation, on-line help). The *presentation interface* can not be specified in ETAG, because suitable grammars for information presentation have not been available and because it is not very urgent, given the number and quality of user interface (screen) builders and e.g. windowing standards. The ETAG specification does provide a good starting point to design a presentation interface that is compatible with and supports the Conceptual Design model.
The *interaction language* can be specified in ETAG's production rules by subsequently specifying the ordering, manner of reference, lexical access, and physical actions associated with each basic task. The interaction language is specified in Task-Action Grammar (TAG, Payne and Green, 1986), with the features derived from the concepts in the UVM, and using the four levels of specification that reflect the main design decisions of the interaction language.
The production rule specification of command language interfaces may be sizeable but, in that case, it is also useful to analyse the consistency and complexity of the interaction language. In contrast, the specification of the interaction language for GUI interfaces tends to be rather simple and highly redundant (de Haan and van der Veer, 1992a; chapter 7). This finding not only underlines the value of formal specifications in relation to usability characteristics, such as redundancy and complexity, but it also indicates that there is a trade-off between the effort required and the utility provided by formal specification: it makes little sense to model trivial languages.
For the design of *metacommunication*, the ETAG representation may serve as a source of exact information about a computer system. In addition, ETAG representations have been used to automatically generate user documentation drafts, on-line help, and rough prototypes, which helps keep design and metacommunication mutually compatible (de Haan and van der Veer, 1992b; chapter 8).

When the user interface is meant for a standard graphical environment, such as Microsoft Windows or X-Windows, dialogue standards can be consulted to create the interaction language specification, besides that various tools are available to facilitate the design, testing and implementation of this part of the specification, such as 'user interface' builders and dialogue, help, and widget editors. When these tools are used without the ETAG representation some care is required, as the consistency with the virtual machine may easily get lost. Both Payne and Green (1989) and Neerincx and de Greef (1993) have shown that inconsistencies of the interaction languages between various parts of a computer system affect the conceptualisation of the system and its performance.
Apart from analysing the usability characteristics of the new user interface design, the formality of ETAG representations may also be used in combination with software tools to automatically derive on-line help or rough user documentation drafts (de Haan and van der Veer, 1992b; chapter 8). No tools have been developed to facilitate the design of the presentation interface in ETAG-based design because of the effort required, in advance, to understand the formal structure of visual information (e.g. semantics, syntax, etc.). There are tools to generate, at least, partial presentation interfaces by taking a less principled approach

by mapping the elements of a conceptual- or an interaction language specification via user interface styleguides on to the elements of specification languages to implement presentation interfaces (e.g. Bodart et al., 1994; Balzert, 1995). A similar approach may be developed that uses ETAG as the underlying language. Another possibility is to extend ETAG, and in particular the dictionary of basic tasks and the production rules with concepts to specify presentation and feedback aspects of the system, similar to Hix and Hartson (1993) and Richards et al. (1986).
During perceptual design it may be best to use repeated user testing. ETAG-based design is built on the assumption that it is the conceptual structure of the user interface that makes the difference, but the perceptual interface also determines performance, and inconsistencies may ruin the effect of a well-designed virtual machine.
At this step we only expect corrective iterations towards earlier steps when things are seriously wrong. On the other hand, we do expect many small iterations within detail design, to fine-tune between interaction language, presentation interface and metacommunication. User testing at this stage will also diminish the chance that overall prototyping and the final evaluation (in use) will show serious problems.

Example: The solution to solve the flashlight problem involved the introduction of an additional flashlight state and the adaptation of the event to change the state. The change of the interaction language involves that, in the flashlight dialogue, the user presses a plus- and a minus-button to choose between "On", "Auto" and "Off" instead of between "On" and "Off". The task to activate the flashlight dialogue does not change. Each buttonpress is associated with a change in the presentation interface in the highlighting and de-highlighting of a particular icon.
The proposed re-design involves only minor changes to the interaction language which are accompanied by known and clear changes to the presentation interface (and presumably, the documentation), so it may be assumed that there won't be major usability consequences. The changes to the interaction language indicate the need to design a new icon to represent "fill-in flash" and that may not be too easy.

4.4.5 Evaluation and Implementation, Installation and Use

Regarding the last two design steps of the received view, two very different directions are possible, depending on the question whether the ETAG-based design approach is used only for design specification, or ETAG is also supposed to play a role in the day-to-day operation of the computer system.
The ETAG that is used to specify the new user interface design can also be used a the formal specification to design and implement the software, either by hand or by using software tools that utilise the information in the ETAG representation. For example, using a BNF representation of ETAG, the interaction language specification and standard Unix tools like Lex and Yacc, it is possible to implement a command interpreter. One step further, Bakker and van Wetering (1991) have written an ETAG machine or emulator to prototype the ETAG part of designs.
Implementation is facilitated because the ETAG notation and programming languages are both formal. In practical terms, when creating a prototype in a procedural language like Tcl/Tk, it is often possible to map directly between the structure in the ETAG specification and the structure of the code, and to use parts of, for example, event specifications as code.
In more theoretical terms, ETAG also shares important features with object-oriented specifications. ETAG object types are hierarchically defined by means of inheritance relations with 'parent' objects. In order to avoid complexity problems associated with the size of

specifications, redundant information, such as repeating attribute slots from parent objects in a definition of an object type can be left out. As such, ETAG representations, although they may look different, are very similar to the more common object-oriented software specifications (Monarchi and Puhr, 1992; UML, 1999). In addition, the methods (in ETAG terms: the events) of objects are named but not defined within object definitions. Whether this is an essential feature of object-oriented representations is open to debate. However, due to the presence of ETAG's object hierarchy, it should not be too difficult to translate an ETAG representation into a object-oriented programming language.
When Jacobson's (1995) suggestion to specify the user interface as early as during requirements analysis is accepted as good practice, ETAG would be a most useful tool for informing software design in its own object-oriented lingo about the elements of the design models which are essential to the usability of the system and should be treated as given. For this to happen, it is necessary that the 'best practise' of an early focus on the user in cognitive ergonomics is also adopted in software engineering. The completeness of the ETAG notation may subsequently help to avoid shifting (user interface) design specifications but a continuous and more structural involvement between user interface and software design as exemplified in MUSE (Lim and Long, 1994) may be preferable and needs to be worked-out.
Regarding any subsequent implementation, installation and evaluation steps, design in which ETAG is only used for specification will not be significantly different from other design methods.

ETAG-based operation is best seen as a research goal for the future. In ETAG-based operation of a computer system the ETAG specification continues to function during normal operation of a system, next to the regular computer application code that implements the functionality. In this case, an ETAG representation functions as the run-time specification for the system similar to the use of user interface languages in user interface management systems.
In a most simple case the ETAG representation can be used to create a user-selectable type of presentation interface, or to provide on-line help about a simulated computer system (Bakker and van Wetering, 1991; chapter 8). When further formal specifications are added, such as task domain descriptions, student- and teaching models, and agents, like process interaction monitors, history- and inference machines, it is possible to create a range of intelligent computer systems which autonomously adapt to the user, provide tutoring services, etc. (van der Veer, 1990). ETAG-based operation of computer systems offers the possibility for a much smoother transition between specification, implementation and use, as well as facilities to support the continuous short-term and long-term adaptation of the computer system to the requirements of its users.

4.5 Discussion

4.5.1 Related Approaches

There are few traditional formal models in HCI that come close to the completeness with which ETAG is able to describe user interfaces. Most formal models are limited to describing the command or interaction language to analyse cognitive complexity aspects (CCT, Kieras

and Polson, 1985; TAG, Payne and Green, 1986) or to predict performance times in restricted circumstances (GOMS, Card, Moran and Newell, 1983; CCT, Kieras and Polson, 1985). Command Language Grammar (Moran, 1981) is quite similar to ETAG. Both describe user competence knowledge for design purposes and use more or less the same levels of description. A main difference between CLG and ETAG is that each level of a CLG representation is meant to be a complete representation of the user interface, at that level, whereas in ETAG the higher levels are built upon the lower ones. As a result, CLG models are notably large and complex, and as a consequence difficult to change and keep consistent (Sharratt, 1987).

Since the early nineties, the interest in developing formal models to represent user knowledge for design purposes has decreased and much of the research on formal modelling has shifted to the representation and analysis of dynamic aspects of user-system interaction (Howes, 1995; John and Kieras, 1994; Blandford and Young, 1995). The application of static, psychologically motivated formal modelling approaches has generally narrowed to representing particular aspects of systems, such as conceptual properties (Green and Benyon, 1995; Benyon et al., 1999) or interaction languages (Hix and Hartson, 1993). Since interest is not in partial representations of user interface knowledge, neither from psychological approaches and nor from software-oriented approaches, they are not discussed here.

Even though the development of models to represent user knowledge has declined, the development of task-oriented design methods which, in one way or another do represent user knowledge has continued. ETAG-based design is one of the structured design methods for user interfaces which represent the knowledge of the user to perform tasks. Here, the main question is the relation between cognitive ergonomics and software engineering. At one end of the scale, there are software engineering methods with HCI extensions. Summersgill and Browne (1989) propose to add various "things to do" to SSADM stages, as well as a few specific HCI oriented deliverables. Apart from the problem that the result is in an even more complex method than SSADM already is, the input from HCI and the usability of systems is still regarded as an add-on to design from a software engineering point of view.

MUSE*/JSD (Lim and Long, 1994) also uses a software engineering method (JSD), in order not to deviate from common practice. In MUSE*/JSD HCI design proceeds in parallel to the software design, and JSD is used to structure the design process as a whole, and enforce well-defined and timely HCI inputs as a means to promote system usability. By facilitating HCI input into software engineering design process, Muse*/JSD may help to create more usable systems. However, Muse*/JSD has no implications regarding the point of view from which the design is undertaken and neither does it enforce nor stimulate that software designers actually use any input from cognitive ergonomics. In practice, Muse*/JSD may be more effective than ETAG-based design, also because it will be more easily accepted but it does not facilitate a principal change in the design of computer systems.

ADEPT (Wilson, Johnson, Kelly, Cunningham and Markopoulos, 1993) is a design method, an environment and a set of prototyping tools. ADEPT makes use of SSADM, but also of the TKS (Task Knowledge Structures) family of task modelling formalisms. TKS models derive from research on task analysis, and as such they are most appropriate for modelling work procedures, objects and roles. TKS is based on psychological considerations (Johnson and Johnson, 1991) even though it is not entirely clear to what extent psychological theory was used to create the notation. Although TKS gives the impression of being less formal than

ETAG, TKS can be formally specified (Markopoulos and Gikas, 1997). Finally, the ADEPT environment offers a graphical design tool to facilitate the design of presentation interface.
DIGIS (de Bruin et al., 1994) is a research tool that offers an object-oriented software design method and environment, specially created for user interface or interactive system design. DIGIS shares ETAG's focus on the user's task world, which may explain the similarity between ETAG's UVM and DIGIS's Domain Application Model (DAM). Whereas ETAG does not specify a software engineering design representation, DIGIS uses a single model for both cognitive ergonomic and software engineering purposes which, consequently, is more difficult to understand. Finally, DIGIS provides a prototyping tool to design the user interface, including the human-computer dialogue.

Without extensive practical experience it is too early to say anything definite about the ETAG and ETAG-based design in comparison to the models and methods mentioned above. In comparison to the other design methods, ETAG-based design uses a formalism that is easier to understand and has a better foundation in psychological theory but, in practice, the formalism is very important. To this may be added that in terms of design steps, ETAG, MUSE*/JSD, ADEPT, and DIGIS have more things in common than differences. All four methods focus on the user's knowledge of the task world, and structure design in levels of abstraction from task analysis to detailed interface design. As a result, they also distinguish very similar representation models for intermediate results, such as a task model, an abstract interface model and a concrete or specific interface model.
What seems most important about these four models is that they present, in combination, a generalised systematic method of how user interface design should take place to ensure usable systems. In providing the requirements for usability from the level of task concepts down to low level details, these methods are likely complements, and perhaps even replacements for the Software Engineering user interface tools and design methods.
Apart from Muse*/JSD, ADEPT and DIGIS, many other design methods are available. These other methods and tools are not discussed, either because they are limited to creating the user interface software or because they have not been relevant with respect to creating ETAG-based design. For an overview of software tools for the design of user interfaces, see: Myers (1994) and for an review of model or task-based user interface development approaches, see: Forbrig and Schlungbaum (1999).

4.5.2 Conclusions

The aim of this chapter is to describe what the design of user interface looks like when it is based on specific ideas about user interfaces, user tasks and task knowledge, and on a formal model of a user's competence knowledge.
What was described as the received view for proper design practice is not very different between cognitive ergonomics and software engineering and, as such, this "standard" design method might fulfil the purposes of both parties. However, the content of each design step is very different. From the point of view that design from a software orientation is the regular course of action, it takes additional knowledge and effort to guarantee that usable computer systems are created. As the received view only prescribes a structure for the design process in which all user-centred concern about the content of design steps can be surpassed, the

received view does not guarantee the design of usable systems. ETAG-based design is proposed as an alternative. Using a formal representation of a user's competence knowledge as the central theme of the design process, designers are kindly persuaded to take a usability stance. In the ETAG-based approach, the design process from a software engineering point of view is put back into a supportive role.
This chapter describes ETAG-based design as a structured method for user interface design, built upon a formal model of a perfect user's competence knowledge. As a user interface representation tool for designers, ETAG is functional in its ability to represent conceptual models of user interfaces in a psychologically valid way. ETAG-based design is proposed as a supermarket approach, which provides a general notation to specify the results of each step in the user interface design process, which allows the designer to leave out those parts and elements that are not required in the circumstances and to supplement the notation with specifications that are required but cannot be captured by the ETAG notation.

With respect to completeness and ease of use, there is room for improvement. ETAG is not able to model the presentation interface. In actual design this may not be a main problem. From the point of view that the presentation interface is there to help users develop an appropriate conceptual model of the computer system, it may be faster and easier to use an ETAG representation in combination with graphical design tools to create the presentation interface rather than to solve the problem of visual grammar first. For throw-away prototyping purposes, tools like Visual Basic, Tcl/Tk and Macromedia director suffice. Advanced user interface and software builders like VisualAge, Glade and Java Workshop are powerful enough for incremental prototyping, in which the prototypes may be as parts of the production system. A disadvantage of such advanced tools is that they are most suited for bottom-up creation of interactive systems from scratch rather than for a top-down creation on the basis of user task models.
A second problem is that it is difficult to create and change ETAG representations. To make ETAG, and indeed many other formal models more usable for designers, there is a need for tools, such as graphical editors to hide away the formalism and its syntactic details. Finally the availability of ETAG-aware tools like presentation interface builders and help editors would fulfil the last requirements for an integrated ETAG-based design environment.

In comparison to other model-based design methods, in theoretical terms, ETAG stands out as a formalism which is relatively easy to understand but has, nevertheless, a firm grounding in psychological theory. In practice, the value of such advantages may be rather limited. For this reason and because there are more similarities than differences between task- and model-based approaches to user interface design, the main conclusion about this chapter is that, in cognitive ergonomics, it is possible to create engineering methods to design user interfaces that fit their users and their tasks are feasible.

This chapter is adapted from: de Haan, G. (1994). An ETAG-based Approach to the Design of User-Interfaces. Part of this chapter is derived from: de Haan, G. (1996). ETAG-Based Design: User Interface Design as User Mental Model Design.

Chapter 5:

Task Analysis for User Interface Design

That the master manufacturer by dividing the work to be executed into different processes, each requiring different degrees of skill or of force, can purchase exactly that precise quantity of both which is necessary for each process; whereas if the whole work were executed by one workman, that person must possess sufficient skill to perform the most difficult and sufficient strength to execute the most laborious of the operations into which the art is divided.

Babbage, C. (1835).

Abstract

This chapter discusses task analysis with respect to ETAG-based design. First, it starts with a preliminary description of what task analysis is or should be, and it discusses several characteristics of task analysis: task analysis as an activity or the result of it, and the detailedness, depth, scope, and contents of task analysis. Second, guided by a discussion about task analysis aspects missing in early approaches, it arrives at a working definition of task analysis.

Third, the chapter continues with a discussion of ETAG's suitability for task analysis, given that it is a formal, a grammatical, and an object-oriented model. In the fourth part, the chapter describes an eclectic approach to task analysis which states that any method should provide information about the context of the work situation, the goals and purposes that play a role in it, the task and actions to acquire the goals, the things such as tools and objects that are used, and the roles people fulfil. In addition, the process of doing task analysis should consist of three steps: to describe the background of the project or problem, to delineate the problem in the project, and to do task analysis proper in describing the work situation. The fifth and last part of the chapter describes a high-level task analysis project as an example from which several conclusions are drawn about the usefulness of ETAG for task analysis purposes.

5.1 Introduction

In this chapter we focus on the relation between task analysis and ETAG. If Task Analysis is defined as creating representations of the tasks that people perform, it may be clear that to apply ETAG as a design representation method, a firm connection has to be established between the representation and the contents and results of the task analysis phase of design. We regard this as essential for any design method that claims to be user centred.

Task analysis is not unique to Human-Computer Interaction design. Task analysis is a common part of any design aiming at creating environments in which tasks are performed. What is special in Human-Computer Interaction is that on a computer system, tasks and task elements are represented by symbols and tasks are performed by symbol manipulation. Many tasks, especially those that involve physical work, are restrained and guided by the physical environment in which they take place in a way that most task performers experience as natural.

In woodworking with saws or axes the physical characteristics of the wood and the tools indicate what may be done, and what may or may better not be done. Task performance in Human-Computer Interaction takes place in a symbolic environment that lacks natural

constraints. In computer-assisted woodworking there is no wood and neither are there saws and axes: only their symbolic representations are available, and regardless of the representations of the wood and the tools -it is even not necessary at all to have e.g. saw-like objects- as icons, tangible virtual objects, sounds, or buttons and dials, the relations between the objects, tools and operations in real woodworking and those in computer-assisted woodworking is arbitrary, artificial, and without logical necessity.
In Human-Computer Interaction information to guide and direct users about which tasks there are, how and with what they are performed, and how they relate to other tasks, in short: the user's task world, has to be explicitly designed into the system. Consequently, user interface design requires more than listing which tasks, functions or actions should be provided.

User interface design is concerned with creating a system to perform tasks that fit together, fit the context in which the system is used, and fulfil the user's task goals, and, as a result, task analysis in Human-Computer Interaction should be concerned with describing and analysing the user's task world. User interfaces in ETAG are described as the Virtual Machines they present, as the virtual task world presented by the computer system. ETAG is not specific to describe user interfaces of computer systems, and may be applied to any other task-oriented 'device' or environment. This chapter discusses how ETAG is and may be used for task analysis, to describe the user's task world as input for ETAG-based user interface design. Using the same representation language -ETAG- for part of task analysis and for subsequent design is assumed to help establish a firm and smooth transition from the analysis to the design phases.

The purpose of task analysis is to collect sufficient information about the task environment that the user interface is meant to support in order to enable user interface design. Within Human-Computer Interaction, task analysis is used in three different ways, each occurring at a specific point within the design process.
The first and most common understanding of task analysis is the description of the user's tasks and task environment, the *extant* task situation, at the start of the design process. Task analysis (proper) in this meaning is the topic of this chapter.
The second understanding of task analysis refers to analysing the proposed user tasks within the new design. Task analysis in this meaning refers to the continuous reflection on the consequences of design choices and should be referred to as simply design or task design.
Thirdly, task analysis may also refer to analysing the tasks as they are or should be performed with a particular user interface. In this meaning, task analysis refers to analysing aspects of a user interface and should be referred to as user interface analysis or user interface evaluation. User interface analysis is the subject of chapter 7.

Task analysis is foremost concerned with the user's tasks. This distinguishes task analysis from requirements analysis or requirements engineering in Software Engineering (Loucopoulos and Karokostas, 1995). Requirements engineering is, as it says, concerned with the required characteristics of the computer system. When the user interface is merely regarded as a piece of software, or as a part of the computer system, task analysis is a part of requirements engineering. When the user interface is regarded as a tool to perform work, and the computer system is merely regarded as a functional implementation of the tool, task analysis is an activity in its own right. The question whether or not task analysis is a part of

requirements analysis is not important with respect to the scientific investigations of the methods and techniques. However, from the point of view of designing usable computer systems the second option is preferable given the little space allocated to usability-related issues in the ANSI/IEEE guide to requirements specification (IEEE, 1996).

Instead of task analysis, we might use more fashionable terms like "task environment analysis" or "work organisation analysis". Terms like these are not appropriate because they imply that it is the task environment or the organisation which are most important and the user's tasks are only of secondary interest. In task analysis, the user's tasks are of primary importance; if there is a problem, it is only because tasks themselves are viewed in an overly restricted manner as if tasks are identical to the actions they require. In this chapter, we will use the term "task analysis", but in a much wider meaning than it is used in most formal models or in old-fashioned physical task analysis.

5.2 What is Task Analysis?

Although virtually everybody will agree that the purpose of task analysis is to collect information about the task situation to enable subsequent design activities, there is a lot of confusion about what it exactly is, which types of information and activities do or do not belong to it, and about how task analysis should actually be performed (see: e.g. Wilson et al., 1988; Diaper, 1989b, 1989c; Johnson, 1992). This section discusses several main points along which opinions regarding the shoulds and should-nots centre. The intent is to clarify matters rather than to decide upon things. In general, it is not possible to make a clear choice between options or choose optimal points on dimensions because such choices depend very much on circumstances and requirements of the particular project. First, we discuss two issues that deserve clarification but do not concern highly debatable issues by themselves. These are: task analysis as an activity versus the result of an activity and the degree of detail that task analysis should be concerned with. Thereafter, we discuss three more debatable issues concerning the depth or the amount of effort to be put in task analysis, the scope or the aspects that task analysis should be concerned with, and the contents or the level of abstraction at which task analysis should deliver information.

5.2.1 Task analysis as Activity or Result

Task analysis is used to refer to both the activity of performing task analysis and to representing the results of such an analysis (Shepherd, 1989). There is no good reason to choose one meaning above the other. It is more common to use task analysis as referring to the activities involved. To avoid confusion we will use it in the more common meaning and refer to the results of task analysis as task model 1 and refer to the results of task design as task model 2 (see: chapter 4; van der Veer et al., 1995).
Generally, task analysis is neither a single activity nor does it yield a single result. Hardly without exception, task analysis requires the use of more than one method and delivers several different models to represent, for example, the task structure, timing aspects, object specifications, etc.

In addition to the difference between task analysis as the activity and the results thereof, confusion may also arise from the difference between task analysis as a set of activities and task analysis as a design step. According to van der Veer et al (1995), a temporally well-ordered sequence of design activities only occurs for well-structured problems with known solutions. In this view, the information requirements of a design project can only be established during the actual design activities. It follows that the required depth and scope of task analysis can, eventually, only be determined during task- and conceptual design and, as a result, task analysis cannot be a separate design step and certainly not a design step that takes place during in the early stages of design. The only feasible option to solve these dependency problems is to iterate whenever that is necessary.

It is beyond doubt that design iterations may be necessary. However, that does not imply that iteration should be accepted as a necessity. Even when many design problems may currently seem to be ill-structured, attempting to resolve these difficulties by allowing for a methodological anarchy is dangerous because this would only help to sustain the idea that design problems are ill-structured. Accepting that design problem solving is very much a matter of iterative trial-and-error is not a very effective way to stimulate research and to learn about better ways to structure design problems. In addition, an a-priori acceptance of the need to iterate makes managing design projects more difficult and increase the reluctance of managers to call for cognitive ergonomic input. Approaches that try to avoid the need for iteration or try to avoid iterations between different design stages do not make project management more difficult and provide better ways to stimulate the use of cognitive ergonomics input (see: chapter 4)

Technically speaking, and regardless whether a waterfall model, a spiral model (Boehm, 1988), or whatever design process model is followed, user interface design consists of a number of activities from task analysis and task design to user interface implementation, introduction and on-site evaluation of the system (see chapter 4). As the first step in the design process, task analysis may also be used to refer to all the activities that precede actual design, including activities concerned with establishing and managing the project, such as formation of a design team, feasibility study, and project requirements formulation. Although this is perfectly legal, especially since establishing the feasibility of a project and formulating project requirements may require information from task analysis, it is better to make a clear distinction between design as product design and as a management issue, for example, by using a project initialisation and survey step.

5.2.2 The Depth of Task Analysis

The lowest level of decomposing tasks into subtasks is referred to as the stopping criterion. Either implicitly or explicitly, the stopping criterion determines when the analyst considers it no longer fruitful to ask what a given task or subtask consists of. The stopping criterion primarily applies to decomposing tasks into their constituents, but the concept of a stopping criterion is also applicable to the task entities subsumed under the question of the detailedness of task analysis, such as, objects, events, tools and time aspects.

The depth of task analysis can only be specified in advance in global terms. For example, when the design is time or safety critical, collecting detailed observations may serve to enable precise modelling of the task performer's actions. For general task analysis, especially when

the newly designed task situation will not have many things in common with the extant task situation, it may only be necessary to identify the user's tasks in global terms and the stopping criterion may be set at a high level of detail. Monk (1998) warns against the tendency to create too deep and detailed task analysis descriptions, and argues that to design new systems it generally suffices to list the available commands.

Since the primary goal of analysing the user's tasks is to identify the user's goals, it would follow that it is sufficient to identify tasks at the interface level, such as commands or basic tasks, or to identify psychologically meaningful tasks such as simple and unit tasks. Unfortunately, the relation between tasks and goals is not straightforward.

First, because HCI deals with symbolic information processing there is no clear relation between the tasks and goals that the interface provides for and those of the user. Van der Veer (1990) distinguishes primary tasks that are assumed to correspond to the user's real goals from secondary tasks that the interface imposes on the user to perform the primary tasks. As such, secondary tasks depend on the particular device and technology. Device dependence is, however, not a sufficient ground to distinguish real from accidental tasks because technology tends to become incorporated in everyday life, such that, for instance, emailing and cc-ing transform from secondary to primary tasks. Secondly, even if it were possible to distinguish primary from secondary tasks, doing so requires abstraction from the tasks and activities which can only take place when such tasks and activities have been identified in advance and at a particular task decomposition level.

We must conclude therefore that setting the criterion to stop task analysis is a local approximation problem that depends on the outcomes of the preceding analysis steps. To know if it is worthwhile to further subdivide a subtask, the analyst first has to identify the particular subtask, which is itself the result of a decision that the stopping criterion is not yet met.

5.2.3 The Detailedness of Task Analysis

The stopping criterion concerns a decision about the detailedness of task analysis with respect to decomposing tasks into subtasks and actions. More in general, task analysis requires decisions about how much, if any, effort needs to be spent on task analysis. The detailedness of task analysis depends on a management decision about how much resources will be allocated to the analysis phase of a design projects, and it depends on a designer's decision about how much information about the extant task situation is required and appreciated. With respect to management decisions about resource allocation there is a common recognition that cost-effectiveness requires that the majority of design resources should be spent on the analysis phases of design rather than on writing the software.

Regarding appreciation by designers, it is sometimes feared that designers who are very familiar with the 'current' user interface will be less able to find radically new and improved ways to solve the design problem, in particular when focussing on the functional task aspects (Suchman, 1987; Bannon and Bödker, 1991). Familiarity may hamper serendipity, so to speak. The implied solution is that design should start with only global information about the extant work situation and allocate resources to evaluate and improve designs during the design process. If creative design is one extreme, the other extreme, called engineering design in Löwgren (1995), will state that information about the extant work situation is a requirement for improvement. To avoid the problems of the extant work situation, to avoid designing the

right system for a different problem, and to control design conservatism, the more information available about the extant work situation, the better.

5.2.4 The Scope of Task Analysis

The scope of the task analysis process concerns decisions about which aspects of the work situation are worthwhile to look into. In contrast to the stopping criterion which sets how detailed observations should be, the scope concerns decisions about whether the analysis process should yield information about, for example, social knowledge, and performance aspects of a task situation. For example, to design a user interface for an office it is not necessary to know a lot about the knowledge of the office worker.
For standard office user interfaces it may be reasonable to assume that the office worker is familiar with the objects and tasks in the task situation, rather than to analyse this knowledge in detail. However, when task analysis is used to create an expert system on the basis of the knowledge of a domain expert, the analysis should obviously include the knowledge of the domain expert. Also with respect to, for example, social and organisational, knowledge, performance, timing, and tool aspects of work situations it is possible to ask whether they should fall within the scope of task analysis. Decisions about the scope of the analysis process require some familiarity with the work situation to assess whether, for example, social aspects are important in the work situation. Contrary to the stopping criterion, decisions about the scope do not generally require the results of an ongoing analysis process, and these may be taken early on in the analysis. Decisions about the scope are part of the analysis process in the creative type of design (Löwgren, 1995), but only when formulating the 'problem definition' is understood to be a part of task analysis.

5.2.5 The Contents of Task Analysis

The contents of task analysis refers to the type of information, and the level of abstraction of the information that task analysis should provide. Whereas the scope of task analysis asks whether information should be gathered about different areas of the work situation, the contents of task analysis concerns the question how abstract the information should be along the dimension that ranges from physical and observable to psychological and conceptual. It is without question that some kind of abstraction is required to describe what was essential in the extant work situation and should be included in a new design. This question goes beyond whether, to describe work situations, it is useful to 'generalise away' individually different ways of performing a task. It is not useful to know that Smith first writes the address on an envelope and then puts a stamp on it, whereas Jones does it the other way around to describe that a letter should have an address and a stamp. Rather, the question is at which level of abstraction and in terms of what type of information, tasks like addressing letters are best described. Parallel to its historical development, task analysis can look into the physical actions of task performers (Taylor, 1911), it may include a description of the perceptual actions between a task performer and a device (Phillips et al., 1988) and it may include conceptual descriptions of the interactions between a task performer and a device (Wilson et al., 1988; this thesis). To describe human-computer interaction and user interfaces similar schemes are followed (de Haan et al., 1991). Which level of abstraction and type of information do best describe work situations depends on the level of abstraction of what the

analyst considers essential in the situation. Interaction in human-computer interaction generally occurs at a high level of abstraction, especially for knowledge-intensive work situations which makes the sole use of physical and physical-perceptual descriptions less suitable.

In what follows, we will define task analysis as:

> creating a sufficiently rich and abstract representation of a task situation as it exists or might exist to allow reasoning about the task situation without having to refer to the actual task situation itself.

This defines task analysis in general, without necessarily referring to task analysis for user interface design. In addition, it goes beyond the notion of task analysis as merely describing "the sequence of steps that it takes for a human being to conduct a task", that Bannon and Bödker (1991) take as the definition of task analysis as it has traditionally been conducted in HCI and more generally in systems design, and which they reject as inappropriate.

In ETAG-based design, the resulting model, or rather the resulting set of models are called: Task Model 1. Task model 1 should refer to *possible* task situations to allow reasoning about hypothetical and about novel types of human tasks (even though new types of human tasks are rare).

It allows reasoning about a task situation *without* having to refer to a specific implementation to demand the possibility that designers do things differently.

Finally, it states that task analysis models should *allow* rather than demand reasoning without referring to particular instances, to allow analysts to use specific task models as the starting point for design, rather than presuppose that they know beforehand what is relevant in every task situation and what is not relevant.

What exactly constitutes a 'sufficiently rich' representation cannot be caught in a definition, unless design was an algorithmic activity. What suffices depends to a large degree on the particular circumstances, but as a minimum, a task analysis representation should capture the user's goals, the tasks or actions to fulfil these goals, and the objects and their attributes tasks operate on, and which form the user's task world.

Task models are generally extended beyond what is merely or minimally sufficient, for a number of good reasons. First, there are functional reasons, which reflect the purpose of a task model. De Haan et al. (1991; chapter 2) discusses how the choice of a particular purpose dictates the criteria that a model has to fulfil. It does not make much sense to create a functional task decomposition tree when users have problems with the concepts employed in a user interface. Similarly, when Edmondson and Johnson (1989) distinguish between descriptive, predictive or prescriptive task models, the distinction is one of purpose.

Secondly, there are reasons connected to specific characteristics of models. Regardless of whether a model is meant for a specific purpose, like description or prediction, it may be important to stress characteristics such as the completeness or the precision (amount of detail). The GOMS model (Card, Moran and Newell, 1983), for example, is actually a family of models to choose from depending on the level of abstraction of information exchange one wishes to analyse.

Finally, task models may be extended or restricted for reasons related to interacting with the representation of the task model. Here, the function of a model to convey information is important, and consequently the 'usability' of the representation is of concern. For example, using augmented transition networks instead of regular transition networks may help to avoid the clutter that occurs when the network grows (Kieras and Polson (1985). In MAD (Scapin and Peirret-Golbreich, 1989) And-Or diagrams are used to allow expression of time and ordering information in two-dimensional task decomposition trees, and Johnson et al. (1988) use the concept of roles for task situations where different people are responsible for particular sets of tasks.

5.3 Missing Issues in Early Approaches to Task Analysis

Task analysis derives, directly or indirectly, from Frederick Taylor. Frederick Taylor is among the first and most influential people who applied engineering measurement methods to manage or rather control manual work. In the "Principles of Scientific Management" (1911) he states, among others, that there should be a strict separation between mental and manual labour, that there is one best way of doing the job, that dividing the craft cheapens its individual parts (the Babbage principle, also known as Taylorism or Fordism), and that tasks should be analysed in their smallest possible actions. This is, in a nutshell, task analysis as it was, and still is practised (except perhaps that Taylor honestly admitted to attempt to make workers redundant).

Apart from the fact that many of Taylor's principles are now regarded as common knowledge, it is not difficult to trace their influence on task modelling approaches in Human-Computer Interaction. The GOMS model (Card, Moran and Newell, 1983), for example, which is, not surprisingly, proposed as an engineering approach to analyse human-computer tasks has many aspects in common with Taylor, such as dividing task into their smallest elements, measuring efficiency in terms of time and error, and the distinction between physical actions on the one hand and mental and perceptual operations on the other. Exactly at the point where Taylor is used beyond its original scope of manual labour, it is also where GOMS runs into difficulties, such as when measuring times with overlapping mental operations and physical actions (see: e.g. Reitman Olson and Olson, 1990), and the need to introduce selection rules to model the 'best way of doing the job' on a per-subject basis (Card, Moran and Newell, 1983, chapter 5).

The point here is not to disqualify Taylor's principles (after all, he *did* succeed in making many workers redundant), nor to disqualify GOMS, since Taylor's principles can be found in any formal HCI model, especially the performance models. Rather, it is a useful starting point to discuss what is restricting (hypothetical or not) traditional approaches to task analysis:

- a one-sided focus on tasks as physical actions
- no account for task objects and tools
- cognitive task aspects are lacking
- little attention for organisational and social work aspects
- no inherent organisation among tasks

5.3.1 Physical Actions

Taylor regards tasks as the linear execution of a sequence of physical actions that attains the task goal. Work consists of the execution, one-by-one, of the physical actions that fulfil the goals of a task. This might have been true in an 18th century steel mill workshop, but it is not so in modern steel plants, and it is certainly not true for human-computer work. Task analysis methods have always aimed at creating task decomposition trees. Implicit in task decomposition is the idea that tasks should be simple for workers to perform them, or perform them rapidly. This may very well be the case in simple manual labour, but in HCI, Reisner (1981) first exemplified that where it concerns cognitive tasks, task complexity is not (only) a feature of individual tasks, but depends on the similarities and differences between tasks.

5.3.2 Objects and Tools

Task decomposition as action decomposition presupposes that the objects and tools with which actions are performed are static entities which are easy to deal with. Computers as tools are not so simple as individual push buttons or a hammer. Van der Veer (1990) makes a distinction between primary and secondary tasks, in which a secondary task is: "an additional task derived from the application of a specific tool (e.g. a computer system) in the course of performing a primary task" (van der Veer, 1990, pg. 19). Secondary tasks, such as how to acquire help about how to perform the primary task may even hamper primary task performance (Neerincx and de Greef, 1993). Secondary tasks associated with the tools can also completely take over and transform the primary tasks. For example, files, lathes and CNC (Computer Numerically Controlled) machines are all used for the same primary task purpose: to remove excess material from a piece of work. However, using a CNC machine requires such complex and careful planning of operations that it is a programming task that has little if anything at all in common with the craft of manual metal-work (Goei, 1994).

5.3.3 Cognitive Task Aspects

In traditional behaviour-oriented methods for task analysis, no attention whatsoever is spent on considering cognitive aspects of tasks. Indeed, 18th century manual labour is, different from nowadays manual labour, genuinely manual, and does not demand abstract problem solving skills. Not attending to cognitive skill is inappropriate when the predominant type of work shifts away from manual labour, and manual labour itself becomes more and more complex. Reisner (1981) has already been mentioned to indicate that cognitive tasks follow qualitatively different rules than manual tasks. Cognitive tasks deal with symbols, and since the meaning of symbols is in essence arbitrary, cognitive tasks lack, to a large extent, opportunities for guidance by external resources.

5.3.4 Social and Organisational Aspects

In traditional methods for task analysis social aspects of work are not taken into consideration, or rather, they are regarded as factors that disturb instead of support the regular work schedule. It was not until the fifties that studies indicated the positive effect of social factors on the workplace which simultaneously invalidated the view that work is only done 'for the money' (Agyris, 1957). In more current views social aspects have gained even more importance. Scandinavian Design (Kyng, 1994), for example, recognises social aspects of

work as an intrinsic property of the work situation which, along with other factors aim at emancipating the worker, and Sproull and Kiesler (1991) indicated how social aspects of the work situation, such as the introduction of email facilities, can change the nature of the work process. Social aspects may be expected to become more important by the increasingly progressing use of computers as a CSCW tool for co-operation between people in work situations, and more in general, as a network tool to connect people for recreation, amusement, and socialising in its own right. It is a requirement for task analysis methods that they provide the means to describe, where necessary, how different people work together on a shared piece of work.

5.3.5 Task Organisation

Finally, and this may be seen as a separate point or as a summary of the foregoing points: Taylor regards work as a linear sequence of executing individual tasks. Except for the logic necessity of behavioural order along the dimension of time, tasks are unrelated to each other. According to Wilson et al. (1988), in traditional task analysis, the analyst describes cues that should be perceived and actions that should be performed, to map these onto behavioural units. In working with computers, however, it is not the perceptual motor skills of a previous generation of technology that must be automated, but the user's conceptual skills, which critically depend on the user's knowledge of the system, its properties, capabilities, and requirements. Wilson et al. conclude: "Units of behaviour can no longer be viewed in isolation" (Wilson et al., 1988, pg. 47). In reaction to the one-sided focus in HCI on features of the user's knowledge, Payne (1991) adds that user interfaces, just like affordances in the real world, are not only passive devices that the user has to learn how to operate, but also function as resources to indicate and guide the user through the work.

5.4 ETAG and Task Analysis

ETAG is in two ways related to task analysis. First, when ETAG is considered in its role as a representation of the evolving user interface design, the results of task analysis are used to create the model. In this role, the ETAG notation determines what is, and what is not useful information for further modelling.
Secondly, ETAG may also be considered as a method or a notation to model interfaces in general, irrespective of whether it concerns a computer device or something in the "real" world. There are, of course, limits to what may and what may not be sensibly modelled in ETAG but, in principle, any system that is organised around tasks and allows for object-oriented representation may be a candidate. In this role, ETAG may be used to explore the task situation, and as a partial task analysis representation.

When ETAG is used for task analysis, the result of task analysis is an ETAG representation and performing task analysis corresponds to creating the model. In task analysis, ETAG is not used to model aspects of the principally mechanical behaviour of user interfaces, but as a means to overview part of the complex, vague and ever changing real world. For task analysis purposes the contents and weight of modelling criteria such as precision and validity is different from user interface modelling purposes. In modelling user interfaces absolute

precision is required with respect to avoiding indeterminism, whereas validity refers only to the results or predictions of applying the representation. In task analysis, however, representations do not have to be very precise, as long as they are valid with regards to giving a true representation of the task situation. To use ETAG for task analysis, the criteria developed in the context of modelling user interfaces (see: chapter 2) may no longer be valid and, consequently, require a closer inspection.

ETAG is proposed as a general tool to perform task analysis. There will be situations, however, in which it cannot be the only tool and the ETAG model needs to be supplemented with other methods and/or representations. Because it is a formal and grammatical model, and because of its object-oriented characteristics, there is information for which ETAG is a very suitable representation method and information for which it is less suitable. As a result, the suitability of ETAG depends on what is most important in different task situations. We will discuss for which task situations and for which types of 'real world' information ETAG is most suitable, considering that ETAG is:

a Formal model
a Grammatical model
an Object-oriented model

To establish for which types of tasks and user interfaces ETAG is a suitable representation it is useless to seek principal answers. Even if the margin would be big enough to prove that ETAG and Turing machines are functionally equivalent (see chapter 8; Thimbleby, 1990) nothing would be gained. To determine a formal definition of the types of problems that may be 'solved' in ETAG is, practically speaking, of no use whatsoever, as long as it remains unclear how much time and effort the ETAG solution requires in comparison to using other modelling approaches. Since it is difficult enough to model the problems themselves, leave alone to model the modelling efforts, we restrict ourselves to general remarks on the basis of the three characteristics.

5.4.1 Formality

As a formal model, ETAG is a good tool to express information that can be formalised, and, as a prerequisite to that, information that can be made explicit. Indeterministic behaviour of systems, such as random failure due to wearing out, cannot be accounted for without special constructs or the mediating role of the user. To model probabilistic systems, such as many games, ETAG is clearly not a suitable notation. Information that cannot or is not explicitly stated cannot be expressed in ETAG. Insofar as task analysis (and user interface design) aims to support explicit task procedures, which is generally the case, this is no problem, and it is the task of the analyst to make things clear. Insofar as things cannot be explicitly stated initially, or in a timely way, it will be necessary to use a more natural language.

5.4.2 Grammaticality

As a grammatical model, ETAG is most suitable to express declarative knowledge, especially if it concerns details, and less suitable to give overviews. This is most clear where it concerns

temporal processes. In ETAG it is easy to define what a teapot is, but it is more difficult to describe how to make tea. Suppose that making tea involves three events, boiling water, adding tea to the pot, and adding the water to the pot. To describe the order of events in purely declarative language such as Prolog, two preconditions are necessary, plus an additional three to avoid that rules fire more than once. In ETAG things are not that bad, since the user controls the order of events. Insofar as procedures cannot be expressed within the context of single events or basic tasks, order has to be specified by means of pre- and postconditions which, if their number grows, make the representation difficult to understand.
ETAG as a grammatical model is very suitable to express type definitions and inheritance relations, but, again, it does not excel in presenting overviews of all available type definitions and hierarchies. Except for very simple interfaces, it will be necessary to use some kind of graphical representation, such as task decomposition trees and inheritance diagrams as vehicles for understanding. In this case, the information expressed in e.g. a task decomposition tree, can also be derived from the ETAG representation, but not in an easy manner.

5.4.3 Object-orientedness

As an object-oriented model ETAG contains a certain amount of information that has little to do with what is modelled, and everything with the object-oriented way of modelling. Task situations which have an object-oriented character by themselves fit the ETAG notation well. According to Rumbaugh et al. (1991) object-orientation generally requires that:

-discrete, distinguishable entities, called objects, can be identified
-objects with the same data structures or behaviour can be grouped into classes
-similar operations may behave differently between classes
-sharing of attributes and operations among classes is based on hierarchical relations

In brief, this means that the task situation allows the identification of discrete hierarchical classes of objects, which share operations and attributes. Situations in which everything depends on everything are not easy to model by means of the ETAG notation, and they are probably neither easy to model by means of any other notation. Circumstances like this occur, for instance, when the attributes and responsibilities of objects and agents change dynamically without clear patterns. When modelling agents which dynamically change roles and responsibilities, it turns out that ETAG is primarily meant as a model for user interfaces and that its concepts of "role" and "agent" have not been worked out in sufficient detail (de Haan, 1997, chapter 6; van der Veer et al., 1996). ETAG may be extended to describe task situations in which several users or agents share different roles, or to create multiple ETAG models, but in more complex situations the use of complementary notations is more appropriate.
ETAG will not be the most suitable formalism for situations in which there are either a few complex objects or data structures and a large number of procedures with little overall structure, or a few complex procedures or methods and many unrelated objects. In such cases, although ETAG might still be able to model the situation, other, non-object-oriented notations are more appropriate in terms of time and effort.

5.4.4 Conclusions about ETAG and Task Analysis

ETAG as a formal, grammatical, and object-oriented model is particularly suitable for the type of task situations which are formally regulated, explicitly and precisely defined, and predominantly consist of objects, as opposed to procedures, and in which there are several well-defined responsibilities. When task situations are very different or when different characteristics are very important, it will be necessary to use models to complement ETAG. Particularly well-defined task situations will be rare, although they may occur, for example when one type of machine is replaced by another, like the transitions from dial to push-button, and from coin to card phones. Well-defined task situations can also be found when the parts of human tasks which are already highly mechanical are automated, such as (central) telephone operation and money or ticket issuing.

Regarding the need to complement ETAG with other models and methods, the following remarks can be made. ETAG is a *formal notation* which is most suitable for information that is easy to describe explicitly and precisely. User interfaces will, of course, always be used for the formalisable parts of task situations. Otherwise it would make little sense to seek assistance from the area of computation. Nevertheless, it is very well possible that, for instance, the choice between using the computer system and another means to perform the task within the actual work situation is determined by conventional, social or other not quite so formal criteria. Information about, for example, choosing between email and the telephone may vanish when specifying the interface. However, since the purpose of task analysis is a complete and correct representation of a task situation rather than a specification of the interface, it should be registered instead of making the information fit the notation. In order to account for information that is not readily formalisable, including information that would be mutilated by doing so, it will often be necessary to keep informal accounts side-by-side to ETAG.

ETAG as a *grammatical model* is most suitable to represent facts or the static characteristics of systems and to represent things in fine details. ETAG is not the most suitable type of model to represent timing aspects, nor is it very good at giving overviews of (the relations between) types of objects. When tasks consist of subtasks it is both necessary and common, to create task decomposition diagrams that show how subtasks and actions stand in a "part-of" relation to the higher level tasks of the user or the interface. From the starting point that a task decomposition tree is the common result and aim of task analysis, instead of being unsuitable, ETAG should be seen as a notation that aims to extend and supplement this information to allow a better description of information manipulation tasks. Timing information may simply be added to a task decomposition tree, and in case the task decomposition does not all too clearly correspond to a decomposition in time aspects (in before-after relations) an additional diagram can be created. The same holds for creating overviews of other relations between, for instance, object types. Creating an "isa" subtype type hierarchy is a standard part of ETAG and similar diagrams can be created to represent other relations between objects and object features.

Regarding the need to complement ETAG's limitations as an *object-oriented model*, it is hard to think about a work situation that does not allow for object-oriented modelling, but still is a

sensible candidate for support by an interactive computer system. In work situations which feature vague and complex objects and agents, and fuzzy tasks, such as in working with human beings, ETAG is clearly not able to model things validly, except where it concerns modelling the formal aspects of tasks. In general, ETAG is only able to model the uninteresting aspects of semantically rich domains in which the content rather than the structure or form is important. In work situations in which simple actions are important and object types are unimportant, ETAG is a too powerful type of model and should be replaced by simpler models such as state transition diagrams, etc. See section 7.2.2 for an example of applying ETAG to a design problem in which simple actions are predominant and that could have been solved better by using simpler modelling tools.

5.5 An Eclectic Approach to Task Analysis

Task analysis should yield the information that is necessary to design or redesign a user interface, where the user interface is defined as the task world of the user or users. This section discusses which specific information is necessary, and how to get at it.
In any rational work organisation, tasks are sets of activities that change the state of things into a more preferred state, according to some pre-set goal. Tasks exist at different levels of abstraction, they may be connected with single persons or larger groups of people, they may have a direct or less direct influence on higher level goals, and tasks may be easier or more difficult to discern. From the point of view of designing user interfaces, what is most important is that:

tasks take place within a context
tasks consist of activities
tasks serve a purpose
tasks operate on things
tasks are performed by people

Task analysis should specify in which context or work situation tasks are performed, which purpose they serve, which activities and objects are involved and, by whom they are performed.

5.5.1 Context

The context of tasks is important in two closely related ways. First, the context gives meaning to the other characteristics. The context is the social, organisational and physical environment of the work situation that determines if and why a goal or an action makes sense. Second, the context sets boundaries on what is relevant within the work situation and outermost boundaries on what is relevant for user interface design.
A problem for task analysis is that the boundaries of the work situation are in many respects not pre-given entities and will only become evident during the analysis process itself. The purpose of a work situation and its physical boundaries may be well defined in advance, but customs and the informal organisation, and the question of their relevance may require familiarity.

In actual task analysis, the formal or well-defined parts of the context are not hard to get at. The initial problem statement or redesign request will indicate which problems the intended computer system is supposed to solve. The design request is initial, because it will often be formulated by people who are directly involved with the work situation and who may lack the necessary distant view on the situation as embodied by the analyst. The problem statement, the management and company documents about the work organisation, purposes, and who are involved provide a starting point to further investigate the more and less formal aspects of the context. Which information about the informal aspects of the context will be necessary will only become clear during the investigations and, especially, when information from different sources is combined to create an overview.

5.5.2 Tasks and Actions

Analysis of task actions provides information about the identity of tasks and actions, about how task goals are decomposed in subtasks and actions, and about problems and opportunities to perform tasks. The common observation that users are very well able to do their tasks and less able to talk about task goals strongly suggests that task action analysis is a good starting point to analyse the organisation of work practices. For user interface design task actions are important sources of information about how tasks might be performed in a new design, and where it concerns problems in task performance it informs about how tasks should not be designed. Tasks are also a major source of information about task goals, and when they cannot be linked to the formal purposes of the organisation they may point at informal aspects of the work situation.
The main aim of task analysis is to identify the user's goals but because goals only exist as the intended outcomes of tasks, the task entity is used as the conceptual unit of analysis that connects the goals to the actions, the things, the people and the tools (see also: chapter 3). To identify tasks it is necessary to know their goals and not knowing them makes it necessary to identify them either from among the task actions or from what users state that their tasks are. In other words, tasks are the sets of task actions that form meaningful units or what users recognise as such. The difference between a task and a task action is that there is no sensible answer to asking how an action is performed except by listing the physical actions involved.

In order to acquire initial information about tasks a method that employs 'templates' is used (Goudsmit, 1993). A template is a fill-out form that is used to collect and document basic information in a standardised form about types of, among others, tasks, objects, and people. If the type is a task, for example, the template will describe pre- and postconditions, which tools and data objects are used, what the task does to the data, and how it is specified by the user. Eventually, for use in design, the aim is to formalise the information in the templates. During task analysis templates are used for exploration and for communication purposes and there is no need to be that precise. Once a more or less complete set is available of the task templates that describe the low level tasks, the task decomposition tree can be derived iteratively, with both a bottom-up and a top-down component. In a bottom up fashion, asking for the (why) reason to perform a certain task yields information about higher level tasks, until eventually the top-most formally specified goal is reached. In a top-down fashion, the formally specified goals and purposes of the department, the computer system, the job, etc. give an indication of which tasks are necessary or relevant. Similarly, asking for a certain high level task for the

way (how) it is performed provides information about tasks, actions and goals at a lower level. When the task decomposition tree reaches completion and correctness, further information from users and from the templates may be added to it, such as about the typical or prescribed ordering of tasks.

5.5.3 Purposes

The purposes of tasks and the goals that exist within the work context are more important to user interface design than the tasks and task actions themselves. Like task actions exist only for the sake of the tasks, so are task goals vehicles for the goals of the work situation. One way in which user interface design may improve the work situation is by changing the work organisation beyond merely changing the allocation of tasks between individual users and the computer system. Because the design of a new user interface will disrupt the work process anyway it is a good opportunity to change work practices, not only to reflect improved capabilities of the computer system but also to make the work situation itself more effective.
Where it concerns formal purposes, including the responsibilities, duties, functions, and tasks of departments, computer systems, and employees, information will be available in company documents. Although, for example, job descriptions will be described in very general terms and may not correspond very well to the actual activities of the person, they do give an idea about who is doing what and why. If the information is insufficient or unsatisfactory, it provides at least something to ask about.
Concerning the purpose of human-computer tasks the situation is more difficult, because users are experts in performing tasks, but not in thinking about them. Studies by Mayes et al. (1988) and van der Veer (1990) suggest that users let the interface lead them through their tasks and further forget about them if there is no reason not to, and de Haan (1998, 1999a) observes how highly-trained professionals know well how to perform their tasks but may lack insight into why tasks are organised as they are.

In order to determine the purposes of tasks and to create a task hierarchy, a combination of template and why-questions is used. With templates (Goudsmit, 1993) users are probed to specify what they regard as their tasks and actions in a standard semi-formal manner by means of forms. Asking "why" questions is part of Sebillotte's (1988) Hierarchical Planning method in which "why" questions are used to acquire higher level tasks and goals, and "how" questions lower level tasks and goals. Applying Hierarchical Planning makes use of the fact that task decomposition trees correspond to goal decomposition trees. The result of asking "why" questions is a task at a higher level, which is also the purpose of performing the original task.
Hierarchical planning is an interview technique and the use of templates is a pencil and paper technique. In combination, they provide for semi-structured interviews in which it is clear what information is requested because it is probed for, and the information is structured by means of the templates and why/how questions, and the results are reported in a form that is suitable for further analysis.

Finally, there are those purposes, such as informal and hidden goals that are not part of the formal organisation of the work, but rather the result of emotional, social, and political factors. To recognise informal purposes requires some human knowledge on behalf of the

analyst to recognise, for example, how an employee might enrich an otherwise below-capacity job. Informal goals will also show up when creating overviews of the work situation as discrepancies between the formal and actual behaviour of departments and employees.

5.5.4 Things

Things are important to task analysis because they provide the other half of tasks, either as the primary objects of task performance or as instrumental to it. Work situation objects do not have to be physical (like this book) but may also be virtual (this textfile) or mental (these thoughts), and do not have to be data (this text) and may also be instruments or devices (this screen). Primary task objects are the objects that are subjected to task actions, and function as the data for the work process. Secondary or non-task objects do not play an essential role in task actions but they may still be important to design when they impose secondary requirements on the user interface, when the use of devices or forms is part of the work habits. In addition to the roles that objects play in task performance, they are also important for the choice and design of metaphors employed in the user interface.

Information about objects that is relevant includes which objects there are, their distinguishing features and structure, types and relations to other objects, and their use in task actions.

Information about things is collected during task analysis similar to information about tasks and actions. Object templates are used to acquire information about distinguishing features of task objects, their relations to other objects and, if applicable, information about super- and subtypes. Again, concrete objects will be most meaningful to the user and act as a starting point for analysis. Since "why" and "how" questions are not applicable, questions ask for more abstract and specific objects to acquire 'isa' information, and for objects that a certain object belongs to and consists of to acquire 'part-of' information. Similar to the analysis of tasks and actions, there are top-most objects that delimit the analysis, which correspond to the objects that play a role in the purpose of the department or the duties of a employee, and there are most specific objects that cannot be specified or decomposed any further, within reasonable limits.

Tools or devices are also things, but contrary to task objects there is no such things as a tool hierarchy or tool decomposition tree. There is no need for that because their number is usually quite limited. In addition, tools either belong to a certain task, like a stapler belongs to a collecting task, or the device forms an environment for a set of tasks that *belong* to it, like the tasks involved in using email. Information about tools is available from the task templates and from users.

5.5.5 People

In task analysis three groups of people are concerned. Management provides the facilities to do task analysis and managers are a source of formal information about the work situation. A similar supplementary role is played by representatives of technical departments concerning information about, for example, the computer systems. A main role is allocated to the users of the computer system. Users make up the work situation and, as such, their organisation, jobs, roles, and tasks are the subject matter of the analysis. In addition, users as domain experts are the main source of information about the work situation. Finally, task analysis should provide

information about how to adapt to characteristics of the prospective users of the computer system to enable effective task performance as well as to make the work enjoyable.

Task analysis should at least provide information about who is responsible for which parts of the work, and how the work of one user relates to that of other users. In addition, numerous other types of information may be gathered ranging from organisational aspects like the type of organisation and working hours to psycho-physical characteristics such as typing speed, colour blindness, etc.

During the analysis, information about people is collected by means of the concept of an agent, where an agent is simply a thing or an object that is able to change the state of objects or perform tasks, such as an alarm clock, an email delivery program, or a regular user. Information about agency should be available from management and users. Machine agency may be less obvious, in which case the technical department or manuals can provide information.

In work situations in which more than one person plays an active part, roles are used. Roles are essentially sets of tasks that a person in the particular role may or may not perform. Roles are a concept borrowed from the TAKD method for task analysis (Johnson et al., 1988), and they are used instead of a user or a person template to allow for the description of dynamically changing responsibilities of single users or between different users. For example, within a Unix environment, the same person may act as a regular user or as a superuser, and these responsibilities or sets of available tasks are not stuck to the person, but rather to the role he or she takes. Formal information about roles can be obtained from the management, either directly or in descriptions of jobs and responsibilities. If there is a difference between what employees are supposed to do and what they do or tell to do in reality, such as when a nurse performs a task that is the responsibility of a qualified physician, or when an employee performs a task that is not related to the purpose of the department, there is reason to ask users for clarification because such "deviations" may contain relevant design information.

5.5.6 Three Steps in Task Analysis

A first step in task analysis is to establish a background for the design project. First, in order to run the project itself successfully, it is necessary to have some knowledge and insights about the organisation in which it runs, such as the type of organisation, how people work together, and where or with whom to look for information or support. Secondly, to create a successful design the same questions apply with respect to the users of the design product and information will be necessary about the background of the problem that the design is meant to solve, and about the information resources, organisation, etc. of the users. It is argued that in both cases gathering such information beforehand will make the task analysis process more efficient, whereas, gathering such information during the process will often come too late, if the methods used for task analysis are able to yield such information at all.

A second step is to delineate a work situation as the subject of discussion (the universe of discourse). Ideally the universe of discourse will be a well-defined part of the goal and task tree but, generally, the task decomposition tree is not a pregiven entity. Generally responsibilities and tasks will only be described in general terms, and for the upper parts of the goal and task tree. Offices and departments may have well-defined business plans and employees have job and task descriptions but, e.g. to ensure flexibility, such plans and

descriptions do not contain information about exactly how to attain the associated goals and how to perform tasks.
In delimiting the work situation, it may also be possible to exclude investigating certain phenomena, either because they are not relevant to the problem that should be solved, or because practical considerations make considering certain aspects of the work situation unfeasible. When design projects aim to solve a particular problem, it may be sufficient that only the characteristics of the problem and not the whole work situation is described in full detail. Furthermore, in real-world design projects, it is common that practical restrictions on, for example, time and human resources define the extent of the project.

Once a workable delineation of the work situation is established, task analysis arrives at its core business: to describe the work situation such that sufficient information about, respectively, the context, the goals, the tasks and activities, the people, and the things is available to start the design.
To acquire information suitable for task analysis, including task object aspects, a range of techniques is available, such as structured and free interview techniques, questionnaires, direct, indirect and participatory observation, and protocol collection and analysis. These techniques will not be considered here, for reviews see e.g. Diaper (1989b), Johnson (1992). Here, we will assume that all the necessary raw information is available about characteristics of the employees, their responsibilities, job description, and individual task decomposition trees, task objects, characteristics of the tools that are used, relations with other departments, etc.

5.6 An ETAG Task Analysis Example

The following example describes the task-analysis part of an actual project concerning the design of the organisational, procedural and technical aspects of a system for information management at a department responsible for system management. The example is derived from: de Haan (1998, 1999a).
Choosing a genuine design project as an example, and especially one in which the focus is on the organisation and the business procedures runs the risk of introducing details, unnecessary for exemplifying the task-analysis process. However, choosing an overly simplified example like the proverbial ATM makes even the most complicated analysis method look very simple and straightforward which does justice neither to reality, nor to the regular difficulties that occur in the task analysis process. According to de Michelis et al. (1998) dealing with the practical aspects of information systems and, consequently, continuous change, it is necessary to consider the three facets, information systems, group collaboration and the organisation, respectively, even though current research communities focus on only one or, at best, two of them. In the example much attention is given to the organisational aspects of a design project. One of the 'lessons learned' from this project is that the fit between things like the way in which a project is run and the solution it provides, and the organisation are much more important to the success of a design project than the quality of the user interface.

5.6.1 The Problem

In a company which provides IT services, the mainframe-oriented legacy software for system management does not fit the requirements of the Distributed Systems (DS) department which is responsible for the midrange system segment. As a solution, a work practice evolved in which the software is used only for purely administrative purposes and the technical information about computer systems is stored in wordprocessor files. When the department rapidly grows and subgroups start using different document formats, document maintenance becomes a problem. In order to solve the problem, a project is initiated to analyse and describe the problem and formulate solutions.

5.6.2 Establish a Background

The first step in the project concerned acquiring familiarity with the problem, the people and the work of the department, and the role of DS within the organisation. In other methods, this step is often captured under terms like project formation or project team formation. From the analyst's part this does not only involve getting familiar with things but also a sense-making process to understand the values and the issues involved.

As a formal method during this step semi-structured focused interviews were held with key people in the organisation, such as the general manager of the department, managers from sister departments, and representatives from departments that provide or receive services.
The interviews were focused in the sense that key questions were prepared to clarify missing, vague and contradictory information from earlier interviews with other people as well as to acquire insight into the structure and responsibilities within the organisation, the main issues and plans, and the problems and opportunities of the department.
The interviews were semi-structured in the sense that the analyst introduced the topics and checked that all of them were discussed but that the interviewee determined the further course of the interview.
Apart from any formal and well-organised information gathering, establishing a background also profited from a great deal of informal contacts, hearsay and plainly getting involved in the activities of the department.
In order to establish clarity rather than to create tangible results, this step of the analysis process was finished on a pre-set date by presenting a short report with general observations and lessons learned, which also served as a means for the parties involved to check for any gaps and misunderstandings in the analysis.

In brief, the results of the step are as follows: the information management problem is part of a set of more general information and communication problems, both within the department, and between departments and locations. These problems were partly due to the after-effects of a merger, the inward-directed culture within the departments, and the practical problems of a rapidly growing market and organisation in combination with the introduction and roll-out of new system management software.

Some communication problems were caused by the multitude of systems used for information management and communication but also because of the culture of having an organisation with relatively independent departments. The lack of co-operation was stimulated by a reward system that favoured doing work for customers above internal work, even in the face of cost-

savings and quality increases for the department as a whole.
During the initial analysis, it became clear that the old legacy system could not be replaced because of its advantages to other departments and to the global operation of the company. As such, also in a new information management system there would be some form of double bookkeeping.

Both the company as a whole and in particular the system management departments had organised work according to strict business procedures. The main business procedure that DS is involved in starts with a procedure "Contract Acquisition" during which a series of quick scans are performed on the systems of a customer to determine the services and resources that are required. On the basis of this so-called Service Level Agreements (SLA's) are signed which arrange for things like type and quality of services, guaranteed uptimes, service hours, etc. Usually, a contract consists of a number of SLA's which are assigned to different departments on the basis of the hardware, software or services required.
Subsequently, during "Contract Implementation" the system or systems are configured according to the standards and the system management software is installed. Further, the systems and services are documented, and the helpdesk and monitoring facilities are brought in place. During the "Service Delivery process" the actual management of the systems takes place as a cyclic process of resolving problems and implementing requests for changes. Eventually, each instance of the main business procedure ends when the systems or the contract is replaced or otherwise discontinued. See figure 1.

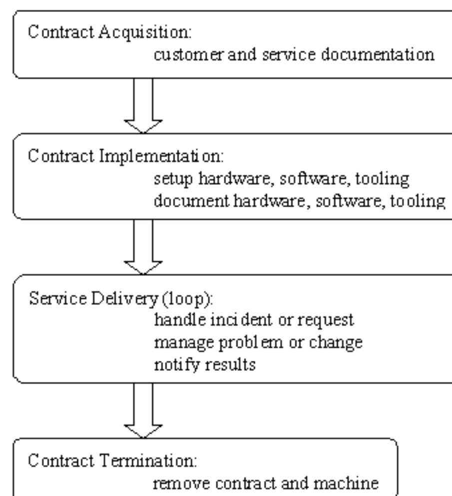

Figure 1. General Business Procedure for System Management.

5.6.3 Delineate the Problem

The second step in the task analysis process is to delineate the problem. The purpose of this step is twofold: first, to clearly identify the main reasons underlying the design project and to determine the most important causes of the problem or problems, and second to determine the constraints that solutions have to meet and to identify in advance the feasibility of alternative

design solutions.
The two purposes aim to delineate the design space beforehand so as to find the best match between the resources that are available and those that are required for the success of the project. Clarifying the design request or the design problem diminishes the chance of a mismatch between the problem and the solution, and clarifying the project constraints increases the chance that a proposed solution may also become the actual solution.

This project concerned a re-design that was not merely restricted to the information system or the user interface. This had a number of methodological implications. The project aimed to solve several known problems in the best possible way rather than as a genuine design project aiming to create a new system. At a methodological level, often no distinction is made between design and re-design. For example, in Muse (Lim and Long, 1994) design starts with an analysis of the so-called extant system, which refers to the old system in re-design projects and to similar types of systems in the case of genuine design projects.
At a practical level design and re-design turn out to be very different. Since this project aimed to solve known problems it was sufficient to have a series of interviews and an analysis of business procedures in combination with observations. There was no need for methods to explore the design space, such as scenarios (Carroll, 1995) or forms of creative design (Monk, 1998; Scholtz and Salvador, 1998).
In addition, because the design problem was well-known and concerned functionality issues rather than user interface details, most of the common design steps, including task- and requirements analysis and interface specification could be done at a relatively superficial level and did not need to be as detailed as would be required for completely new design.

The method used to delineate the problem consisted of a series of semi-structured interviews with stakeholders in combination with a formal analysis of the business procedures. In this case there was a reasonable consensus about the nature of the problem: a multitude of sources for technical documentation that were sometimes inconsistent and difficult to access. The focus of delineating the problem was to establish the a-priori feasibility of possible solutions. In due course this proved to be a right choice when it became clear that office politics between managers and departments had a major impact on the running of the business.

First, a series of semi-structured interviews were held with the manager of each subgroup of the DS department. These interviews asked about the way work was organised in the particular group in terms of roles and responsibilities, and about management issues in relation to information management, such as policies, personnel management, budget, and future plans. The interviews were also used to clarify the questions arising from the analysis of the formal business procedures. With respect to the difference between the methods, formally used and actually practised, it deserves mention that in addition to the data from the interviews with the group managers, also some data were used from a series of interviews held with a system manager from each group. Because these interviews took place in the context of the next step, describing the work situation, they are not further discussed here.

A second method to delineate the design problem consisted of analysing the formal business procedures that are part of the quality system. Like many other IT businesses, the company and the DS department made extensive used of methods to improve the quality of their

services, such as ISO-9000 (ISO, 1987), TQM (Total Quality Management; Roa et al., 1996) and CMM (Capability Maturity Model; Paulk et al., 1993; 1995).
Furthermore, in this company, system management was set up according to a methodological framework called ITIL (IT Infrastructure Library; CCTA, 1989), originally a set of reference books on IT management written under the auspices of the British government for its Central Computers and Telecommunications Agency (CCTA). ITIL divides systems management into a number of services and processes such as Customer support (the helpdesk), Database management, Contingency services (disaster management), and Problem- and Change Management, etc. each of which describes both the service delivered to customers as well as the procedures for delivering the particular service.
Quality documents formally describe at an abstract, conceptual level how an organisation performs its work, ranging from top-level process models down to work instructions, role assignments and responsibilities, and standards. In task analysis quality documents are very useful as a guide to how work is performed even though they are not a straightforward description of what happens on the work floor; for that, they require further interpretation (de Haan, 1998, 1999a). Business processes only describe the work in so far as it must obey rules, such as about the existence of particular roles and the procedures to request that some task is performed, but they do not describe how roles are assigned to real people nor how to perform a particular task.

The results of this step were as follows. The system management responsibilities of the department consisted for most part of the service delivery process. Service delivery is the cyclic process of doing whatever is necessary to have machines work properly. Figure 2 shows the main steps in the DS system management service delivery process.

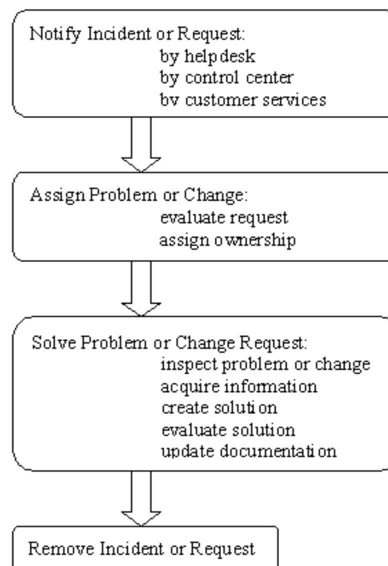

Figure 2. The System Management Service Delivery Process.

Although service delivery is a uniform process, in practise there is a difference between long-term special projects and short-cyclic system management activities. The regular system

management activities of solving problems and implementing requested changes make up the bulk of the work, both in terms of frequency and in the amount of work.
Whenever a request to solve a problem or implement a change occurs, it is first assigned to a specific department or group. The department or group first investigates if they are able to solve it. If not, the request is redirected to another group. When, for instance, a database problem or a general operating system problem requires a configuration change, the request is rephrased into one for the software specialists. When a group accepts a request, it is assigned or taken up by one of their system managers. The system manager investigates the problem and solves it, then documents the changes and solution, and finally removes the request from the queue by notifying it as solved.

The legacy software for system management was ill-adapted to the task demands of system management in non-mainframe environments, both as a toolset to perform tasks, as a resource for technical information and as a system to manage the process or the procedures. In short:

- the toolset was limited to a particular proprietary network protocol
- information-per-machine did not fit dealing with machine types and clusters
- problem and change requests were likewise oriented around single machines
- procedures were not fit to deal with many, small, and frequent problems and changes

The non-mainframe departments used a different set of standard tools for system management. To manage the process and for information management, the legacy software was still required, but most departments had introduced "work-arounds". With respect to managing the process or procedures few problems occurred but in information management it resulted in a multitude of different documentation procedures, systems, sites, formats and software, both between departments and between the groups of a department. Since groups and departments are often responsible for system management of different parts of the same systems and need quick and easy access to each others documentation, this created both content problems such as inconsistencies and communication overhead. The business procedure for document management is shown in figure 3.

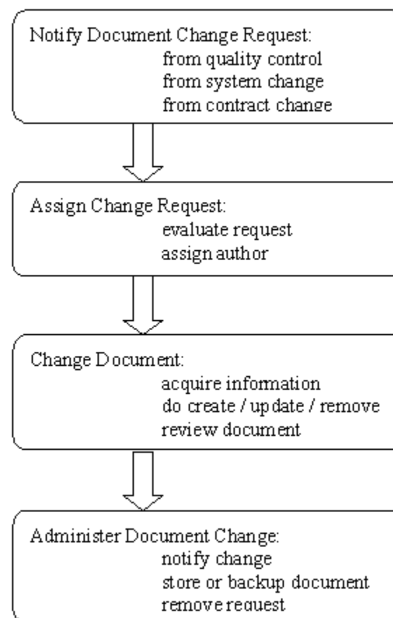

Figure 3. The Business Procedure for Document Management.

The document management procedure starts with a request to check and update some or all documents, either periodically, on the initiative of the quality department or whenever a change in a system or its service level contract requires a corresponding change to the documentation, including the creation or the removal of documents. Following the regular change management procedure, the request arrives at the document owner; generally a group manager, who ensures that the document is updated and reviewed. Finally, the changes are fed back to the requester and the quality department, it is stored or a backup copy is made, and the request is removed.

There had been a number of earlier attempts to resolve the problem. Of these, the enforced top-down attempts had not been generally adopted, leading to the use of a number of different management systems next to each other. The consensus-based bottom-up attempts often resulted in ambitious plans which subsequently faded away because in the reward system working for customers was valued much higher than working on internal projects. Following the experiences with earlier projects, it was decided to restrict the project to the DS department. Furthermore, in line with the use of the method of 'best practises' it was decided to follow an incremental prototyping approach and start with an example system for document management with a limited functionality and user population. In addition to creating a prototype, also a set of procedures or work instructions should be designed to address the ownership and maintenance of document management. Once the example system was successful, other requirements and user groups could be added. Finally, a number of additional decisions were made which delineated the problem, both with respect to characteristics of the project, like duration and resources, and with respect to the functional requirements of the example system, which should, for example, provide facilities for version control, global access and task-appropriate navigation.

5.6.4 Describe the Work Situation

The third step in task analysis is to describe the work situation. Whereas in the two preceding steps the aim is to give meaning to the work situation by describing the context, and to make a distinction between what is considered relevant and what is not by delineating the design problem, this step is concerned with task analysis in the proper sense. Two methods were used in this step. First, a semi-structured interview was held with an experienced system manager from each group. These interviews were guided by the so-called work instructions, the lowest-level business procedures that describe how the work should be managed whilst allowing freedom to decide how to do it. Second, while performing both regular and critical system management tasks, a small number of system managers were observed.

The interviews with the system managers focussed on how the work was actually performed, guided by an example of a hypothetical but common work problem. They were asked about the steps to solve such work problems, the use of information resources, and the common and important problems they met in doing so. Whenever time allowed, they were also asked to describe an ideal system for acquiring technical information.
The purpose of the interviews was to determine how each group translated the work instructions into actual task performance or the way of working in each group, and to determine important differences therein between groups.

The unstructured analysis and observation method included becoming familiar with the software tools and information systems that are used for system management, and identifying their general usability problems. In addition, and particularly informative, was observing how work was performed during a few extensive projects that span a considerable period of time, such as the preparations for upgrading a large number of systems, and observations during several critical incidents, such as so-called "Severity One's" when systems are down for too long or down at the wrong time.
Extensive work projects, when there are no stringent deadlines, provide an opportunity to observe the regular way of working, while the critical incidents show most clearly what makes the regular work procedures break down under tension. In addition, the observations were used to check the interview results.

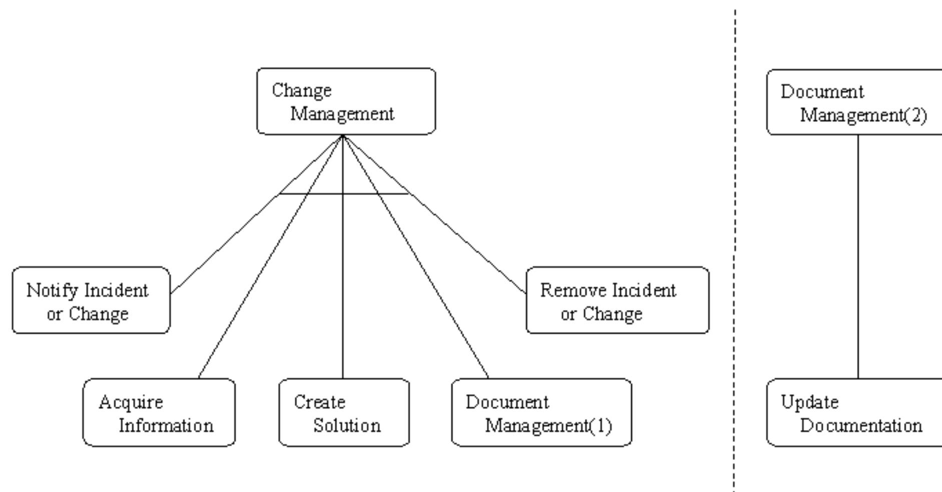

Figure 4. Task Decomposition for Dealing with Problems and Changes.

Here, the results of the task analysis step are discussed with regard to information management. Figure 4 shows the task decomposition tree for solving a problem or implementing a change on a system. With respect to information management, most problems occur at the leaf entitled 'Acquire Information' whereas the greatest differences between groups are found at the 'Document Management' leaf. Needless to say, the most notable problems occur because of human error in creating solutions.

The first information about a problem, a change or about a particular system is acquired from the document containing the problem or change record in the legacy application. This document is shown on the display screen and, as such, the task to "Acquire Information" should only involve reading the information on the screen. In principle, this information should suffice in most cases but often it does not. Due to the limitations of the legacy application, in particular for the groups with less experienced system managers, often, additional information about a problem or change request is required before it can be worked on. As a result of the need for additional information, the task to "Acquire Information" becomes much more complex because of the need to locate, access and open a document in order to read the information in it. Figure 5 shows the top-level task tree for the task to acquire information by reading a document. The boxes at the bottom, marked "Acquire Login" and "Acquire Read-access" are optional and only applicable when access is not or not yet granted.

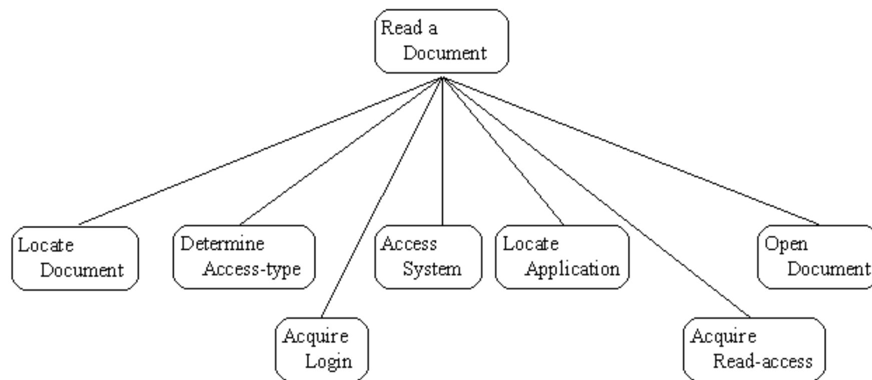

Figure 5. Task Decomposition for Reading a Document.

This information should be readily accessible but often it is not clear where to find it. Technical information resides at different places, depending on the group that is responsible for the system, the architecture or type of system that is of concern, and the type of service. Partly, the inaccessibility of information is accidental; due to the rapid growth of the department, some information is kept at obsolete places until time is available to put it at the place where it should be. In addition, system managers are reluctant to move their group-specific documents to a system different from the one they use most often. System managers clearly recognise the need for good documentation but nevertheless, regard it as a necessary evil.
In some cases, and most notably the information about service contracts, the information is inaccessible on purpose. The formal argument is to prevent information from leaking to the competition. Informally, information access is an asset in negotiations between groups and departments.

Another problem, especially for less experienced system managers, is to determine in what document or database the information should be sought. To manage a system properly requires, in principle, four types of information about, respectively: service contracts (contacts, assets, service levels), the system configuration, the system operations, and history information about for example known problems. For practical reasons, this information is kept in many different files and databases. For example, some information is automatically kept by software tools, several groups placed frequently required information in an online database on a system that they use to manage other systems, and several groups introduced special documents to present technical information in a way that is adapted to the needs and levels of experience of their system managers.
The result of the situation was the creation of a telephone culture. Given that it is always clear who is the owner of a system, the obvious thing to do is to pick up the phone, explain the problem and ask what to do next. As a result, not only much time is wasted by the interruptions themselves, but expert system managers also end up doing the work of their novice counterparts: being interrupted anyway, it is often easier for the experts to quickly solve the problems themselves rather than to explain where to find the information or explain what to do.

A third problem with accessing technical information occurred after locating the relevant information when, due to the use of different computer systems, places to store documents, and applications, it was not always possible to actually consult the document itself. To open and consult a document requires that one has to locate it, choose the way to access the computer system, access the system, locate the software to open the document and, finally, to have read access rights to the document. Generally, access rights to systems proved to be a bottle neck. For group managers it is undesirable to allow system managers from other groups on their systems because of loosing the advantage of keeping information private to their group. An alternative is to work with user privileges that allow outsiders to access parts of specific systems. From a system management point of view, however, it is very inefficient to maintain different user access lists for each system that is used to store documents.

In document management, the greatest differences were in the way of working between groups. In ITIL, document management is just another form of change management or, rather, document management is performed by applying the change management procedure to documents (see figure 3). The document management procedure is triggered by two events. First, there is a periodic trigger to check and update the documentation every once in a while, generally in connection to a quality audit. Document management is also triggered when fulfilling a problem or change request is associated with a change in a service contract or a system attribute such as the configuration or a contact address that is documented.

Part of the document management process is dealt with through the legacy software at the higher levels of problem and change management. During the procedure of removing an incident or change, by indicating it as being solved, a short description or change log is produced for the department in charge of the legacy software to update the information. However, because the information in the legacy application is insufficient for the non-mainframe departments and groups, document management becomes an additional part of their implementation of the change management process.

Within the DS department, the formal document management procedure prescribes that whenever a document needs to be changed because of solving a problem or implementing a change, the responsible system manager informs the document owner of the required change. The document owners themselves or an author assigned by them subsequently creates, changes or removes the document, after which, if the document is approved, the document is administered. In this process, there is no difference between dealing with electronic or paper documents or between genuine documents and information resources such as databases.

Although there is one procedure for dealing with documents, several different versions were in use, and it was not always clear which of these were different implementations of the procedure and which should be classified as violations. Figure 6 shows the task decomposition tree for document maintenance. Although the formal procedure consists of only four steps, the number of steps in the task tree is much higher to account for all the different implementations.

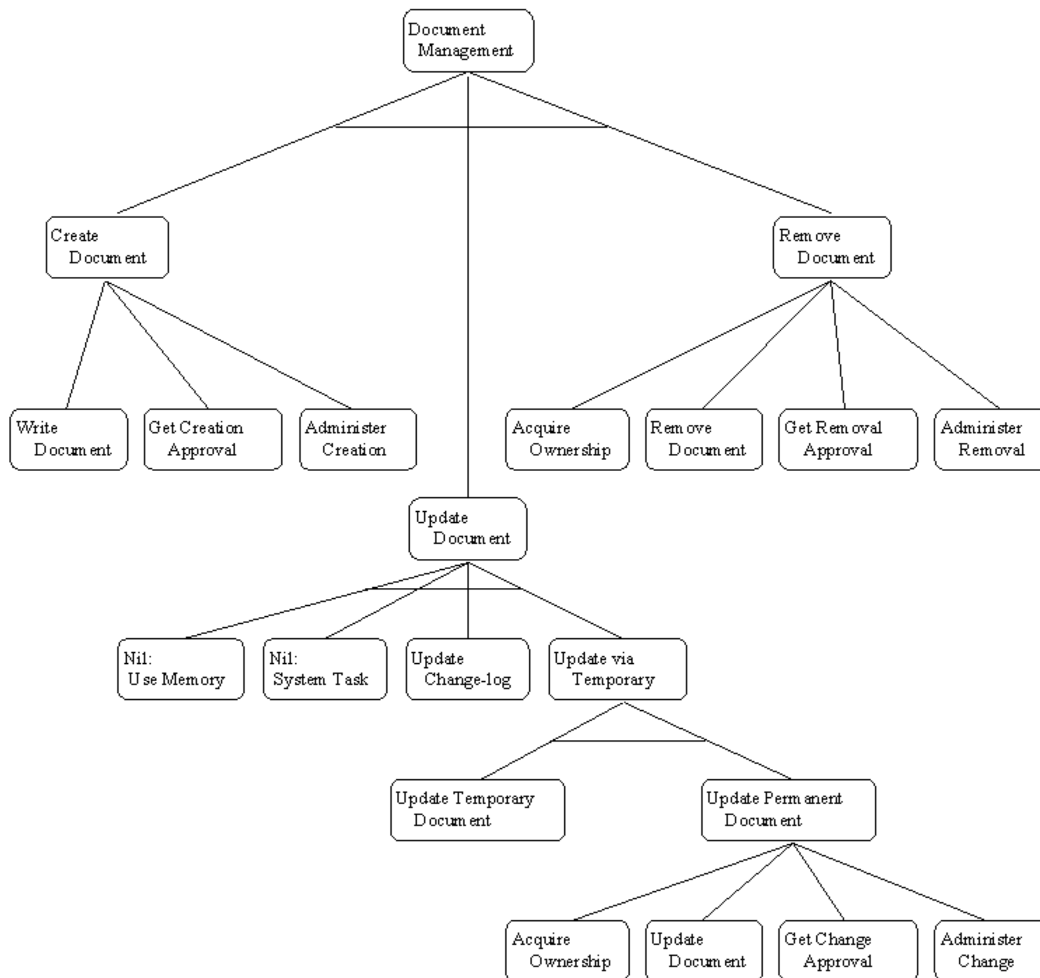

Figure 6. Task Decomposition for Document Maintenance.

First, one group that provided BPR services did not use a document management procedure as part of change management and neither used the legacy software. All their information was automatically kept by the system management tools.
A second, software specialists, group used document management to create new documents but did no document maintenance whatsoever. Awaiting an overall solution for technical information, they only maintained documents on an ad-hoc basis and further relied on their own memory.
A third group also used document management to create new documents and remove obsolete ones, but they restricted maintenance to keeping a change log.
Finally, several groups used the document management procedure but made a distinction between working documents and other information resources such that no approval was needed for changing the working documents and databases whereas changing any other documents, such as database backups and the documents that are also used outside the group, was subjected to the formal procedure.
Both the variety of the ways to maintain technical documentation and also the difference between the way maintenance should be done and was actually performed exemplify that

documentation was seen as overhead that should rather be avoided. This was especially the case among the more experienced system managers.

5.6.5 An ETAG Task Analysis Specification

This section describes an ETAG representation of the example with reference to general tasks involved in problem- and change management, and with specific reference to the tasks involved in document access and in document maintenance. For the sake of brevity, parts that are not required are left out, but in contrast to using ETAG for design purposes, the model is not tweaked to arrive at the most short and simple result but it describes the tasks as they are actually performed to determine how well ETAG may be used for this purpose and more specifically to determine how well the representation does to pinpoint problems and complexities in performing tasks.

Earlier, it was noted that any method for task analysis should be able to specify the relevant aspects of five subjects: context, purposes, activities, people and things, and also, that ETAG's suitability will be influenced by the fact that it is a formal, grammatical and object-oriented representation.
As a formal tool, ETAG will be particularly suitable to describe the formalisable aspects of the work situation and fail where it concerns the non-formalisable aspect, such as the motivations of the users. As a grammatical tool, ETAG will be particularly suitable to describe the elements of a work situation and be less suitable to provide overviews of the relations between elements, like the ordering of tasks, for example. As a object-oriented description tool, ETAG will be most suitable to describe the aspects of a work situation in which distinguishable, discrete and classifiable entities are important, and it will be less suitable for fuzzy, continuous work situations in which few invariants exist, as in verbal communication between people.

5.6.5.1 Object Specification

The object specification lists and describes all the entities that are relevant to understand the work situation. Tasks are also objects in ETAG but here they are described in a separate section. The object specification consists of the object hierarchy which describes the relations between objects of a kind, and the object specification, which, whenever relevant, describes the objects themselves. In order to provide an overview before presenting the details, here the object hierarchy is presented before the object specification but during the actual analysis the object specification is created first.

The object specification includes the organisation, agency within the work system, the system entities themselves, and the external tools that may be used. There are no object specifications for the organisation and the communication tool. The organisation is not a genuine ETAG type of object, but it is described here and in the hierarchy only because of its relevance to the purposes or goals of the work situation; the organisation is what connects all the other objects and tasks.
There is no object specification for the communication tools because such information is only relevant for the description of the task events which is left out of the analysis. Yet, the

communication tools are listed in the hierarchy because they are relevant to understand how particular tasks are and may be performed. As such, the description of the organisation provides a context to the top of the analysis by describing the circumstances in which the system is used and the communication tools provide a context to the bottom of the analysis by describing the technology that implements the system.
With respect to agency, the role concept is excluded because all the agent-types themselves refer to specific roles. Even though there are only a few types of agents, the agent hierarchy and descriptions are somewhat complicated because the Customer agent is not part of the system that is described, but it does play a role in the functioning of the system.

The Object Hierarchy

```
Company ->                          provide IT services
        Department ->                       provide system management services
                Subdepartment ->                    provide DS system management services
                        Group ->                                    provide specific DS services
                                Manager                             manage group resources
                                System_manager                      manage systems
Agent ->
        Tool
        Human ->
                Customer
                Employee ->
                        Manager
                        System_Manager
System ->
        Customer system
        Legacy system ->
                Problem queue ->
                        Request ->
                                Problem
                                Change
        Work system ->
                Directory
                File ->
                        Application
                        Document ->
                                Change_log
                                Temporary_document
                                Permanent_document
Communication_tool ->
        Telephone
        Meeting
        Email
        Notes
        Fax, ...
```

The Object Specification

General Objects

```
        type [System isa object]
        subtypes:       Customer_system, Legacy_system, Work_system
        themes:         File | Directory
```

```
places:         place.ON(System)
attributes:     System_information, System_name, System_owner
end

type [Customer_system isa object]
themes:         Tool_agent
places:         place.ON(System)
end

type [Legacy_system isa object]
themes:         Legacy_application, Problem_queue
places:         place.ON(System)
end

type [Problem_queue isa object]
themes:         Request
places:         place.ON(Problem_queue)
end

type [Request isa object]
subtypes:       Problem_request, Change_request
attributes:     Request_description, Request_type, Request_status
end

type [Work_system isa object]
themes:         File | Directory
places:         place.ON(System)
end

type [Directory isa object]
themes:         File | Directory
places:         place.IN(Directory)
end

type [File isa object]
subtypes:       Document, Application
places:         place.IN(Directory)
end

type [Application isa object]
subtypes:       Application1, Application2, ...
attributes:     Document_type, Application_path, Application_owner, Login_type
end

type [Document isa object]
subtypes:       Change_log, Temporary_document, Permanent_document, Copy
attributes:     Document_information, Document_type, Document_path,
                Document_owner, Login_type
end

type [Communication_tool isa object]
subtypes:       Telephone, Meeting, Email, Notes, Fax, ...
end
```

Agents

```
type [Agent isa concept]
```

```
triggers: Create_request
subtypes:       Tool, Human
end

type [Tool isa agent]
attributes:     Tool_information, System_name
end

type [Human isa agent]
subtypes:       Customer, Employee
end

type [Customer isa agent]
attributes:     Customer_name, Customer_information
end

type [Employee isa agent]
triggers: <all tasks and events>
subtypes:       Manager, System_Manager
attributes:     Employee_name, Employee_information
end
```

5.6.5.2 Task Specification

The Task Hierarchy

Tasks are specified in the task hierarchy and task specification. The task hierarchy is the ETAG equivalent of the task tree and it shows how the tasks are related to one another. The task specification describes each task in terms of the agent or role that triggers the task, and the events or subtasks that make up the task. Since the analysis is concerned with information management, only the tasks (and objects) which deal with document management or which are relevant to that process are included.
In comparison to using ETAG for design specification, the task hierarchy in task analysis represents the actual task decomposition rather than the least complex decomposition. For explanatory purposes, the task hierarchy is presented before the task specification, but in actually performing the analysis, the task hierarchy is created after the task specification is reasonably complete and information is available about which tasks are unique and about how tasks relate to each other.

```
Create_request
Notify a problem
Acquire_information       ->
        Acquire_information_from_Request
        Acquire_information_from_System
        Acquire_information_from_Customer
        Acquire_information_from_Employee
        Acquire_information_from_Document  ->
                Locate_document
                Determine_access_type
                Acquire_login
                Access_system
                Locate_application
```

```
                    Acquire_read_access
                    Open_document
Create_solution
Update_documention  ->
          Create_new_document  ->
                    Write_document
                    Get_creation_approval
                    Administer_creation
          Update_old_document  ->
                    Update_change_log
                    Update_temporary_document
                    Acquire_document_ownership
                    Update_document
                    Get_change_approval
                    Administer_change
          Remove_old_document  ->
                    Acquire_document_ownership
                    Remove_document
                    Get_removal_approval
                    Administer_removal
Remove_request
```

The Task Specification

The task specification describes each task in terms of the agent or role that triggers the task, and the events or subtasks that make up the task. Although it may be helpful to specify the task in greater detail and to list the tools that are used, to describe the exact pre- and postconditions, and to add comments, here they are left out for the sake of brevity.

In comparison to using ETAG for user interface specification, it is necessary to interpret the event- and task-concepts differently. Instead of regarding events as an entity that cannot be decomposed further in subcomponents, in the example, events are interpreted as the smallest meaningful changes to the system. In order to describe a user interface to access a computer system it may be meaningful to describe events at the level of individual keystrokes or providing the userid- and password fields. For a high-level task analysis, specifying keystrokes and userid-fields is not meaningful anymore.

With respect to the depth of the analysis, a task analysis at the level of, for instance, command invocations, requires more effort than would be justified by the gains, and with respect to the detailedness of the analysis, the additional information would add little to a better understanding of the work situation. On the contrary, adding more details would probably only make the representation more confusing, also because of the need to describe each alternative way to perform a subtask and each alternative ordering of subtasks.

In a similar way, the concept of a task cannot be interpreted as a regular ETAG basic task or as the entity that associates a basic event to a user controlled concept (Tauber, 1990). To use ETAG for task analysis purposes it is necessary, first, to assume that tasks may consist of one or more user controlled subtasks, like the concept of a menu task (van der Meer, 1992; de Haan and van der Veer, 1992), and second, to interpret the concept of a task in terms of meaningfulness within the work situation, like the unit task in Card, Moran and Newell (1983) or the simple task in Payne and Green (1986). Both unit tasks and basic tasks are defined in terms of the psychological meaning to the individual task performer. In the example, concern is not even anymore with user tasks, such as specifying userid as part of a

login procedure, but rather with tasks that are or are not meaningful with respect to the organisation, or more specifically to the goals of the department.

Top-level Tasks

```
type [Create_request isa task]
triggered by:     Agent
effect:           event.Create(Request, Request_type)
end

type [Notify a problem isa task]
triggered by:     Employee
effect:           event.Set(Request_owner, Employee_name)
end

type [Acquire_information isa task]
subtasks:         Acquire_information_from [Request, System, Customer, Employee, Document]
triggered by:     Employee
end

type [Create_solution isa task]
triggered by:     Employee
effect:           event.Create( Hypothesis), event.Test(Hypothesis),
                  event.Create(Solution), event.Test(Solution)
end

type [Update_documention isa task]
subtasks:         Create_new_document | Update_old_document | Remove_old_document
triggered by:     Employee
end

type [Remove_request isa task]
triggered by:     Employee
effect:           event.Open(Request), event.Set(Request_status)
end
```

Acquire Information Tasks

```
type [Acquire_information_from_request isa task]
supertype:        Acquire_information
triggered by:     Employee
effect:           event.Open(Request), event.Determine(Request_description)
end

type [Acquire_information_from_system isa task]
supertype:        Acquire_information
triggered by:     Employee
effect:           event.Login(System), event.Determine(System_information)
end

type [Acquire_information_from_employee isa task]
supertype:        Acquire_information
triggered by:     Employee
effect:           event.Contact(Employee), event.Ask(Employee_information)
end
```

```
type [Acquire_information_from_customer isa task]
supertype:      Acquire_information
triggered by:   Employee
effect:         event.Contact(Customer), event.Ask(Customer_information)
end

type [Acquire_information_from_document isa task]
supertype:      Acquire_information
subtasks:       Locate_document, Determine_access_type, Acquire_login, Access_system,
                Locate_application, Acquire_read_access, Open_document
triggered by:   Employee
end
```

Acquire Document Information Subtasks

```
type [Locate_document isa task]
supertype:      Acquire_information_from_document
triggered by:   Employee
effect:         event.Determine(Document_type), event.Determine(Document_path)
end

type [Determine_access_type isa task]
supertype:      Acquire_information_from_document
triggered by:   Employee
effect:         event.Determine(Document_type), event.Determine(Login_type)
end

type [Acquire_login isa task]
supertype:      Acquire_information_from_document
triggered by:   Employee
effect:         event.Create(Request, Access)  |
                event.Contact(Employee), event.Ask(Access)
end

type [Access_system isa task]
supertype:      Acquire_information_from_document
triggered by:   Employee
effect:         event.Login(System)  |  event.Rlogin(System)  |  .....
end

type [Locate_application isa task]
supertype:      Acquire_information_from_document
triggered by:   Employee
effect:         event.Determine(Document_type), event.Determine(Application_path)
end

type [Acquire_read_access isa task]
supertype:      Acquire_information_from_document
triggered by:   Employee
effect:         event.Create(Request, File_access)  |
                event.Contact(Employee), event.Ask(File_access)
end

type [Open_document isa task]
supertype:      Acquire_information_from_document
triggered by:   Employee
```

```
effect:         event.Open(Application), event.Open(Document),
                event.Determine(Document_information)
end
```

Update Documentation Tasks

```
type [Create_new_document isa task]
supertask:      Update_documention
subtasks:       Write_document, Get_creation_approval, Administer_creation
triggered by:   Employee
end

type [Update_old_document isa task]
supertask:      Update_documention
subtasks:       Update_change_log  |  Update_temporary_document  |
                Acquire_document_ownership, Update_document, Get_change_approval,
                Administer_change
triggered by:   Employee
end

type [Remove_old_document isa task]
supertask:      Update_documention
subtasks:       Acquire_document_ownership, Remove_document, Get_removal_approval,
                Administer_removal
triggered by:   Employee
end
```

Update Documentation Subtasks

```
type [Write_document isa task]
supertask:      Create_new_document
triggered by:   Employee
effect:         event.Open(Application), event.Create(Document_information),
                event.Close(Document)
end

type [Get_creation_approval isa task]
supertask:      Create_new_document
triggered by:   Employee
effect:         event.Create(Request, Approval)  |
                event.Contact(Employee), event.Ask(Approval)
end

type [Administer_creation isa task]
supertask:      Create_new_document
triggered by:   Employee
effect:         event.Move(Document), event.Set(Document_owner)
end

type [Update_change_log isa task]
supertask:      Update_old_document
triggered by:   Employee
effect:         event.Open(Application), event.Open(Change_log),
                event.Change(Document_information), event.Close(Change_log)
end
```

```
type [Update_temporary_document isa task]
supertask:      Update_old_document
triggered by:   Employee
effect:         event.Open(Application), event.Open(Temporary_document),
                event.Change(Document_information), event.Close(Temporary_document)
end

type [Acquire_document_ownership isa task]
supertask:      Update_old_document | Remove_old_document
triggered by:   Employee
effect:         event.Create(Request, Ownership) |
                event.Contact(Employee), event.Ask(Ownership)
end

type [Update_document isa task]
supertask:      Update_document
triggered by:   Employee
effect:         event.Copy(Document, Copy), event.Open(Application),
                event.Open(Copy), event.Change(Document_information),
                event.Close(Copy)
end

type [Get_change_approval isa task]
supertask:      Update_document
triggered by:   Employee
effect:         event.Create(Request, Approval) |
                event.Contact(Employee), event.Ask(Approval)
end

type [Administer_change isa task]
supertask:      Update_document
triggered by:   Employee
effect:         event.Move(Copy, Document), event.Set(Document_owner)
end

type [Remove_document isa task]
supertask:      Remove_old_document
triggered by:   Employee
effect:         event.Copy(Document, Copy)
end

type [Get_removal_approval isa task]
supertask:      Remove_old_document
triggered by:   Employee
effect:         event.Create(Request, Approval) |
                event.Contact(Employee), event.Ask(Approval)
end

type [Administer_removal isa task]
supertask:      Remove_old_document
triggered by:   Employee
effect:         event.Delete(Document)
end
```

5.6.5.3 Conclusions

A number of issues derive from the ETAG task analysis example. They are discussed here, grouped according to the categories of issues that task analysis methods should address: context, purposes, activities, people, and things.
Regarding *context* issues, ETAG does not seem to be very helpful here, especially where it concerns the reasons for and causes of the problem. Partly, this is because these reasons lay outside the work situation that is being described and partly because the reasons are neither formal nor formalisable.
Earlier, it was stated that the problem was caused by a number of issues: the reward schema for external versus internal projects, the ill-suitedness of the mainframe tool, and the rapid growth of the department's business. Of these, only the reward schema could have been put into what was called the organisation specification of the model. However, considering that the group managers have to deal with a large number of constraints like these, the choice is between "tweaking" the reward schema into the specification or creating a huge specification that would contain a lot of irrelevant information. In addition, these reasons might not have created the problem without the cultural issues of group independence, regarding documentation as overhead, and co-operation by trading resources.
As such, we may conclude that, indeed, ETAG is not suitable to describe the context of the problem in the work situation, particularly when causes lay outside the formal organisation.

Regarding *purposes*, two conclusions may be drawn. First, as long as there is a one-to-one relation between the (formal) purposes in the organisation and the task activities, ETAG is very well able to model purposes. Second, when the purposes are not directly related to task activities, as in this case, ETAG does less well. The inclination, for example, of the system managers to solve problems but to neglect maintaining the resources required for that cannot be modelled in ETAG. Given the limitations of the model as either a formal model or a model for the generic aspects of a work situation, it does not seem reasonable to blame this on the model, also because informal issues like these are better described in natural language.
Regarding goals and purposes, we conclude that ETAG is well able to model the task-related aspects of goals and purposes but non-task related and high level goals and motivations are better described in prose.

With respect to modelling *task activities*, ETAG seems to do very well, in that the size and the complexity of parts of the representation clearly indicate where the problems occur. From the representation it is clear that there are many tasks to reach the same goal, and that there are complex pre-conditions for several tasks, such as having login-access and knowing the type and whereabouts of a document before it may be consulted. Also, the representation indicated clearly how some of the problems might be solved, such as by unifying the diversity of storage, access, and format types.
On the down side, ETAG does not very well in describing the ordering of tasks when alternative orderings are possible and when a particular task-goal may be reached by different sets of tasks. In the example, there were other ways to determine document and access types and using a strategy to always FTP a copy of a document might render these tasks superfluous. Van Welie et al. (1998a) distinguish between modelling task constituent structure in a task tree and modelling task order in what they call a "task flow specification", and in this respect, in ETAG task ordering is sacrificed for describing task constituents. Finally, the and-or diagrams show task ordering information better than ETAG's task

hierarchy does even though these also rapidly become complicated when there are many different alternative orderings. Regarding modelling tasks, we may indeed conclude that ETAG is very suitable to describe the tasks and actions in the work situation but that it is not very suitable to describe task ordering aspects.

With respect to the *people* and the organisation, ETAG is indeed able to model the aspects of the task situation, although it does not necessarily provide useful information. In the example, there was no need for the role concept because the agents were agent-roles. The agent hierarchy is somewhat complicated because of the mixture of human and software agents, and because of the need to capture the customer agent from outside of the work situation. In addition to the agents in the model, it may be possible to add additional agents to distinguish between the system managers of the different groups, but since the task specification is a separate entity from the agent specification, this would not have made the representation any better or clearer.
The organisation hierarchy could be modelled easily, also because of its clean hierarchical structure. Because the general way of working was roughly similar for each group there was no need to model how the system management services were assigned to groups. Since this assignment is based on different principles, including historical accidents, ETAG would not have been very useful here.
Also with respect to the organisational reasons behind the problem, we conclude that ETAG is restricted to describing the formal characteristics of people and that it is not able to describe the non-formal aspects of the organisation such as motivation and culture.

Finally, regarding modelling the *things* or the objects and tools used, ETAG seems to be quite useful. Similar to modelling tasks, the size and complexity of the object specification indicates where the problems occur, especially when considering that several tools and objects in the specification in the example were only listed and not fully specified. The different types of documents were only listed but their number, and the size of the object specification that they would require, indicates that there are too many of them.
The tools used were only listed in a hierarchy and they were not specified because they only play a role in a type of event, to contact another person, that was left outside the specification. A problem with the tool-objects is that it is difficult to specify in ETAG the conditions for using a particular tool, such as when matters are urgent or when another tool failed. In addition, in ETAG it is not possible to specify an important characteristic of a tool, namely, the degree to which it disturbs the person who is contacted other than by mentioning some kind of disturbance attribute. Note however, also here it does not seem reasonable to demand this from any representation.
With respect to things, the conclusion is that ETAG is suitable to describe the objects and tools that feature in tasks, especially for physical or system things but not with respect to their psychological features.

5.7 Conclusions

This chapter discussed task analysis in the context of ETAG-based design, including key characteristics of task analysis in general, a definition of task analysis, and the suitability of ETAG as a task analysis representation on both theoretical and practical grounds. It described

an eclectic approach to task analysis and, as an example, a high-level task analysis project to test the suitability of the ETAG representation for task analysis purposes.

Regarding the example, the following observations were made regarding the issues that methods for task analysis should address:

- ETAG is not suitable to describe the context of the problem in the work situation, particularly when causes lay outside of the formal organisation.
- ETAG is well able to model the task-related aspects of goals and purposes but non-task related and high level goals and motivations are better described in prose.
- ETAG is very suitable to describe the tasks and actions in the work situation but it is not very suitable to describe task ordering aspects.
- ETAG is restricted to describing the formal characteristics of people and it is not able to describe the non-formal aspects of the organisation such as motivation and culture.
- ETAG is suitable to describe the objects and tools that feature in tasks especially for physical or system things but not with respect to their psychological features.

Apart from the issues related to the appropriateness of using ETAG as a representation for task analysis, two further issues are related to the usability and the usefulness of the ETAG model for task analysis purposes. First, the representation rapidly becomes very large either when the analysis aims to include more low-level details, when the analysis describes higher-level issues of a work situation, and when the analysis is intended to be more complete.
The analysis in the example was delineated with respect to both a pre-given top-level of analysis, in the form of document management within a department, and a pre-given bottom-level, in the form of elementary or meaningful task-events. Subsequently, while performing the analysis, parts of the representation were left out; either because some part would not have been relevant to the problem, or because they were better represented by means of a different kind of representation, like a picture or a text. For the problem at stake, this worked very well, both to keep the focus on the problem, as well as to keep the size of the representation within reasonable limits. Given that the current problem was one at a relatively high level and complicated, we may expect that ETAG-based task analysis is useful for other work situations, including those in which interacting with computer systems does not occur.

A second general issue is the question what has been learned from performing the analysis. Creating a task analysis representation involves a chicken-and-egg problem that to understand a work situation requires a representation of it and to represent a work situation requires and understanding of it. With respect to ETAG the question becomes: is creating the representation helpful to understand the work situation? At the start of the project ETAG could not have been used, because it is necessary to first get an understanding of the organisation and the issues and problems at stake.
When these issues became clear, ETAG's task- and object concepts were most useful to precisely describe the practical aspects of the problem. There is no difference in terms of the

concepts used between ETAG and other current methods for task analysis since, contrary to the more traditional methods like HTA (Shepherd, 1989; Annett and Duncan, 1967), all methods identify task and object categories. There may be a difference between ETAG and other methods in terms of the precision of the analysis. Because ETAG uses a formal model to represent the results of the analysis, it provides direct evidence or proof about where things go wrong. In this respect, the size and complexity of parts of the analysis did not only indicate where problems occurred but also how they should be solved. Had a less rigorous representation method been used, this might not have been evident.
ETAG was useful to describe the practical problems in the work situation, but it was not very useful with regard to the context of the problem, which simply could not be captured in the model. In this respect, methods for task analysis that are able to represent the higher level goals that gave rise to the problem, and the trade-offs between these goals would do better to understand the nature of the problem. However, even though some methods exist to model goals and responsibilities (Kaindl, 1998; Blyth, 1996), we are not aware of any methods that combine this ability with the rigour of a formal model.

For the sake of completeness, the solution proposed to solve the document management problems of the example consisted of storing the documentation as much as possible at one place in a few different formats, adding web interfaces to create task- and experience specific interfaces, as well as to ensure global and uniform access, and a proposal to change the procedure in certain cases from approving changes to periodic validity checking (see: de Haan, 1998).

This chapter is based on the following papers: de Haan, G. (1998). The Politics of Information and Knowledge Sharing for Systems Management. De Haan, G. and van der Veer, G.C. (unpublished document). Task Analysis for User Interface Design. de Haan, G. (1999a). The Design of a Information Infrastructure to support System Managers and Business Procedures: how to catch a guru with quality.

Chapter 6:

How to Create an ETAG Representation

In general, too, what distinguishes the man who has knowledge from who does not is the ability to teach, and this is why we regard art as being more truly knowledge than experience: those who posses art can teach, those who do not cannot

Aristotle (ca. 340 bc).

Abstract

Extended Task-Action Grammar (ETAG) is a formal modelling notation to describe user interfaces for design purposes, in terms of the knowledge a competent user should have to use the interface to perform tasks. This chapter discusses a stepwise procedure to create ETAG representations. The procedure is described in terms of steps which describe what to do, guidelines which describe how things are best to be done, and a number of remarks which describe several "lessons learned" in a less structured way. The procedure is illustrated with two examples from a project to create a wearable computer assistant. The chapter finishes with a brief discussion of ETAG, ETAG-based design, and other modelling approaches to user interface design purposes and it draws several conclusions about things that need to be improved.

6.1 Introduction

ETAG is a formal notational language for specifying user interfaces for the purpose of design. In ETAG user interfaces are represented from the point of view of a perfectly competent user, and represent what has to be known in order to perform tasks through the interface. ETAG, or rather ETAG-based design is also an attempt to bridge the somewhat problematic gap between Human-Computer Interaction (HCI) and Software Engineering (SE) in a radical user-oriented way. On the one hand, even though HCI has delivered examples of usable systems, Software Engineering is still very much concerned with designing systems from a technical point of view. On the other hand, HCI has delivered examples, but it has not offered sufficiently adequate methods and tools to ensure good user interface design. HCI is in many respects still a craft, depending on individual knowledge, expertise and a particular (user-oriented) attitude.

HCI should shift from a (psychological) scientific approach to one with an engineering point of view, and deliver methods and tools that are easy to use for software engineers and other non-expert HCI users to assist in creating usable computer systems. In 1987, Thomas Green (Green, 1990) stated that HCI should provide limited theories, as theories which would solve design problems in a psychologically sound way, without requiring that all main questions in cognitive psychology be answered first. In our view, ETAG as an embodiment of psychological knowledge in the form of a relatively simple but usability enforcing design method for -also- non-experts in HCI is a move in the right direction. This chapter discusses how to create ETAG representations for design purposes, in order to move towards a psychologically sound engineering approach to user interface design.

6.2 How to Cook an ETAG Representation

In this section we will only discuss matters directly related to ETAG models. Psychological issues, such as how to do task analysis, or design issues, such as when to use a command versus a direct manipulation interaction style are left aside. ETAG representations have a lot in common with Object-Oriented Analysis and Design models (OOAD; for a review see Korson and Vaishnavi, 1992). Both types of notations are meaningful, rather than purely formal, both have only a few formal rules, both use hierarchical definition, focussing on objects, and in both, implementing the representation in software is (or rather: should be) trivial once the model is created, but creating the model is not all that easy.

It is important to realise, however, that an ETAG representation is not a Software Engineering Object-Oriented representation, since ETAG is strictly concerned with user knowledge. On the one hand this means that ETAG is more restrictive in the use of features like e.g. implementation classes because these are not meaningful. On the other hand, ETAG is less restrictive because it allows psychological objects and attributes that need not have well-defined equivalents in the domain of the system, such as well-known entities or big and small items (without the need to define how big "big" is). In ETAG, psychological meaningfulness is sufficient. Meaningfulness not only restricts but also frees the designer from arbitrary technical limitations, imposed by e.g. types of inheritance and relations, or the requirement to define methods (ETAG's events) as part of object types. Also with respect to the models point of view there is a difference in that Object-Oriented models abstract the user away or treat the user as another entity within the system, whereas ETAG modelling is only concerned with the user's view of the system.

While creating the ETAG model, there are two -usually related- things to keep in mind: the intended scope of the model, and the level of detail. The scope has to do with what has to be modelled. If the model is merely created to acquire ideas about the user's task world, the model does not need to be as complete, detailed and correct, as it should be when the purpose is to create a design representation. For example, if the interface is to have a command to execute operating system commands, it is generally a waste of effort to try to model the operating system. Similarly, when the computer system is meant for a standard graphical user interface style, there may not be a need to model the production rules.

The criterion of the level of detail determines the level of design decisions left to the implementation. It is a "stopping criterion", similar to the criterion used in task analysis to decide whether to go into more detail or stop the analysis (Diaper, 1989a, 1989b). ETAG-based design is concerned with creating usable systems, which is not necessarily the same as pixel-precise modelling. For example, in representing a form-filling application, we found that modelling exact cursor movement within each field did not add much information while it made the model extremely large.

Where to put the stopping criterion is not always clear: it depends on a combination of modelling considerations and practical restrictions and opportunities. As a rule of the thumb, modelling details ends with the boundary between system events and basic tasks on the one hand, and simple user actions on the other. In this respect, cursor movement is likely to be only a part of an event to enter or change an attribute, whereas, from a software point of view, cursor movement is generally handled by a user interface module or widget, and not part of the application's functionality.

Given the considerations regarding the scope and the level of detail of the model, what remains to be done is actually creating the model. In this respect, a study by Yap (1990) proved to be of key importance to the development of ETAG. The primary purpose of the study was to explore and extend the descriptive validity of ETAG by analysing the process of applying it to two functionally comparable applications: the page tools "pg" and "less". Page tools are used to browse through computer files by displaying a single page of information at a time.
Since methods were not yet available for creating the ETAG representation, the problem was approached by listing all information about the programs which seemed relevant and attempting to structure the information according to ETAG's requirements. Simultaneously to creating the representations, problems occurring in doing so were recorded, as well as inabilities of ETAG to represent aspects of the programs. The problems related to the descriptive abilities of ETAG were used to improve the theory behind the method. These aspects are described in more detail in chapter 7. Yap derived a set of guidelines for creating ETAG representations on the basis of his experiences with respect to the modelling process with the aim of general applicability. Most of the guidelines emphasise the structure of the process to do things, such as:

> Make a list of Objects
> Construct a hierarchy of Objects definitions

A smaller number of guidelines provides directions about how to identify entities, such as:

> An Object is an entity manipulated directly by an Event
> Introduce a Place when an Event explicitly creates an Object at a Place

It was not possible to design a deterministic algorithm to do ETAG modelling but the set of guidelines proved successful as a starting procedure for creating ETAG representations. Experiences with respect to the modelling process in more recent studies have been used to elaborate Yap's guideline approach and overcome the main limitations that follow from modelling readily available computer programs.
First, when modelling existing systems the universe of discourse has been specified beforehand and need not be delineated. As a consequence, the information required to set the scope and level of detail of the model is readily available or may be probed for, and, the elements of the ETAG model only need to be recognised and put in place.
Secondly, although there is no principal difference with respect to the results of the modelling process between modelling new designs and modelling existing systems, in practical terms there are. Modelling existing system allows for a significant bottom-up component in which the system serves as a baseline to check the model. Modelling for new designs lacks a baseline and requires that the specification and iteration of the model proceeds according to a design approach in order to keep the process within manageable limits (see: chapter 7). The procedure that will be described below explains ETAG modelling in a form that is suitable for both new designs and descriptions of existing systems.

In terms of procedural steps, an ETAG model is created by gathering information, structuring it to make it understandable and fit the ETAG notation, creating (part of) the model, and only

if the model is (locally) complete, iterate to make it better (more consistent, smaller, etc.). Within the iterative procedure, ETAG models are loosely created top-down. We speak about loosely top-down modelling to indicate that the general direction of specification is top-down, but that locally "anything goes", including bottom-up abstraction and 'opportunistic' design (Guindon and Curtis, 1988).
After a first delimitation of the area to be modelled, in terms of preliminary specifications of the canonical basis and the user's tasks, the ETAG model is filled in and refined starting with the top-most concepts and working downward. The complete specification of the Dictionary of Basic Tasks sets a clear border between the User's Virtual Machine and the production rules. The specification of the UVM and the production rules are relatively independent of each other, without many iterative cycles passing over the Dictionary Tasks. In other words, once there is a clear understanding of the objects, attributes and tasks (either as basic tasks or as events), only at the level of details will iteration be required between the three main parts of the ETAG model.

Two examples will be provided about how to make an ETAG representation. Both deal with a system that is developed in the Comris project, one of the thirteen I3 (Intelligent Information Interfaces) Esprit Long Term Research projects that were launched in 1997. Comris (Cohabited Mixed Reality Information Spaces; van der Velde, 1997) is a research and development project that seeks to develop a wearable assistant for conference and workshop visitors. On the basis of a personal interest profile and an active badge system, conference visitors receive context-sensitive information about interesting persons and events in their spatio-temporal vicinity. For example, a person who has indicated a high interest in wearable computers on her home-page or profile form may receive a message that "a demonstration of the parrot, a wearable conference assistant, is about to start in 5 minutes at the Comris booth".
The first example is derived from de Haan (1999b, 2000) and describes (in the paragraphs 6.2.1 - 6.2.5) part of a simplified ETAG representation of the Comris system as a whole and this focuses on the higher levels of the representation. The interesting point about this example is what to include in the representation and what to leave out.
The second example is derived from de Haan (submitted) and describes (in paragraph 6.2.6) one of the prototypes for Comris' wearable component ("the parrot") that allows the user to browse through the received messages. This example focuses on the lower levels of the ETAG representation.
The examples will be presented as much as possible as if the Comris project is still in its early stages even though the real Comris system is a working prototype.

6.2.1 Step 1: A Raw List of Tasks, Objects, Attributes, etc.

The first step in creating an ETAG notation is to gather information about the context in which the interface will be used, which tasks should -at least- be supported, the task structure and functionality of an extant or comparable computer system, etc. The purpose of this step is to acquire data about the would-be design, so unstructured methods like open interviews, brainstorms and drawing sessions can be applied as well as (semi-) structured ones, such as collecting wish lists, requirements and e.g. 'mental' prototypes.
In full-blown ETAG-based design (de Haan, 1994, 1996; chapter 4), we presuppose the creation of a task model 1, completely representing the extant task situation, if any, and a task model 2, representing the new task situation at a high level of abstraction, without a specification of the user interface details. In such a case, both task models can be analysed in terms of which tasks, objects, etc. are required or may be useful.
The result of this step should be a number of (unstructured) lists of user tasks and selectable options, objects or data types, object features or attributes, and possible object states. As it is

easier to delete items than to come up afterwards with forgotten items, one should rather be overcomplete than too restrictive.
If it is not completely clear what exactly the objects, etc. are, one may wish to use general heuristics mentioned in the OOAD literature. Examples of such heuristics to identify objects (or classes for that matter) are to look for nouns in textual descriptions of the user interface (Coad and Yourdon, 1990; Greatbatch, et al., 1995), or look for the static entities that operations act upon (Rumbaugh, 1991). Yap (1990) lists a number of specific guidelines regarding identification of ETAG elements and steps to create an ETAG model, which will be used below. He suggested, among others, to look for 'things' which provide space for each other, are acted upon by tasks, or play a role in task conditions in order to identify objects.

Example: Comris is a system that is meant to support visitors to large conferences in planning and keeping track of the activities by means of a small wearable computer. Comris users specify a number of interests, such as attending events around a particular topic, making appointments with particular people or meeting people who share a particular interest. Users provide each interest with an interest-value.
When users start to move around, agents in the background start making matches between the interests, people and events using information about the whereabouts of the user, the user's agenda, and the conference schedule. A personal watchdog agent selects the most competitive match and if one surpasses a user-settable threshold, it is passed on to the parrot which signals its availability to the user and presents it in spoken form via a headphone or in textual form in a small wrist-worn display. After presentation, the messages remain available to the user until they are responded to, deleted, or have lost their relevance.

The Comris system is built around four different types of messages:

- *Proximity alert* - A presentation on usability engineering, involving John Peterson as a speaker, is about to start, in the lecture room, in 5 minutes.
- *Meeting proposal* - Mrs. Susan Bowen, from Computer Soft, proposes to meet you, to discuss augmented reality, on Friday at 3 o'clock.
- *Commitment reminder* - Don't forget you have an appointment, with Olivia Rogers, on groupware systems, at 3 o'clock.
- *Suggestion* - You might be interested in the presentation to be held at 3 o'clock, on Friday in room 22, on artificial intelligence, by Peter Roberts.

Each message has a modular form that corresponds to the event it refers to, and events consist of a type of event, a person, a timeslot, a place, a topic, and a hidden topic value. In Comris, the modular structure derives from the mechanism to generate the messages (Geldof, and van de Velde, 1997) but it is to the modeller to decide whether or not this information is relevant to the user. To understand the structure of each message as well as to create the software, they are relevant but to understand the workings of the system it might be argued that only the matching of interests is relevant. For example, since Comris keeps track of the user's agenda, in principle, users do not have to "understand" time aspects or the concept of an empty timeslot.
More in general, Comris can be modelled as black box that merely presents messages to the user, it can be modelled as a semi-transparent box that includes a description about the contents of the messages, and it can be modelled as a glass box that includes a description about how the messages are brought about. The last model is too detailed. The process of matching interests and the agents involved are not relevant to the user, since this could also have been done by the personal watchdog agent or some monolithic system.
In order to keep the explanation within reasonable limits, the example will only deal with messages that inform the user about the formal event of a conference and leave everything out that relates to the interactive uses of the system such as agenda keeping and meetings proposals.

The first step in creating the ETAG model is to produce a raw list of all the entities that users must know about: Actors, Objects, Attributes, Values, Places, Paths, States, Tasks, and Events. The derivation is along these lines: there are two entities that start tasks or events, persons and their agents. A conference consists of a schedule with a number of events like demonstrations and presentations, and each of these has a topic like "usability" which is derived from a general list of topics. Conference events take up timeslots and are held at a location.

Users also have topics that they are interested in and each has a particular interest value. The topics and the topic values are used in combination with a message threshold to generate advisory messages. These messages, finally, are stored in a user-specific message list. The raw list of entities is as follows:

person	agent
schedule	conference_event
event_topic	topic_list
timeslot	start_time
end_time	location
person_topic	topic-value
person_location	threshold
message	message_list

Users are involved in three types of tasks. First, tasks related to specifying the topics of their interest, their interest values. Secondly, users specify how often they want to receive advise or, stated differently, how relevant messages should be in order to be presented. Thirdly, once messages have been presented, users are able to read or listen to them, and to remove the messages that they consider obsolete. As such, there are six user tasks:

specify_interest_topic	specify_topic_value
set_threshold	play_message
browse_message	remove_message

The system or rather the user's system agent deals with two tasks only: to present a newly created message and to remove from the user's list of messages those that the system considers obsolete. As such, there are two system or agent tasks:

create_message	remove_old_messages

6.2.2 Step 2: Select Concepts for the Canonical Basis

The second and third step are concerned with delimiting the design space, by establishing the preliminary canonical basis and dictionary of tasks. Step two consists of picking the canonical basis, which is largely a matter of selecting from among the standard canonical concepts (Actor, Object, Attribute, Value, Path, Place, State, Task and Event). The Actor concept is only necessary if there is another 'intelligent' agent besides the user. The Path concept was introduced in ETAG to address file paths and movement routes. However, it is advised not to use it for mere file identification. Paths easily clutter the model, apart from not belonging to the genuine application knowledge. At this stage it may not be wise to add non-standard canonical concepts; not only because of lacking psychological generality, but especially to avoid the danger of adapting a different view on the task world than the user's.

Guidelines:

ACTORs are required when events do not originate from the user only:
(network, message presentation, traffic light, ...).

OBJECTs are the entities that are changed by tasks:
(file, string, window, car, message, ...).

ATTRIBUTEs represent the properties of objects, except for location and existence:
(name, number, amount, colour, ...).

VALUEs describe the values or range of values that attributes can take:
("foobar", "ETAG", "deleted", "dev/null", 1 - 100, ...).

PLACEs are where objects live, possibly within other objects:
(ON, ON-TOP, ON-POSITION(i), LEFT-OF(i), ...).

PATHs are needed when reference to object locations or movements are relative:

(TO, FROM, VIA, ...).
STATEs identify specific objects by attribute values, places and existence:
(BE, HAS-VAL, IS-AT, IS-CURRENT, ...).
TASKs are executed by actors to change the state of objects:
(delete-message, copy-file, cut, eat-icecream, ...).
EVENTs actually change objects, and their attributes, places and states:
(KILL-ON, MOVE-TO, SET-VAL, CREATE, ...).

Example: From the raw list of concepts that play a role in modelling the Comris application it follows that it is necessary to include certain concepts in the canonical basis. These are derived by taking ETAG's standard canonical basis concepts (e.g. Actors, Objects, etc.) and asking for each if there are concepts in the raw list (e.g. person, agent, schedule, etc.) that require them or by doing this vise versa. Apart from analytic derivations it is a matter of experience that certain canonical concepts occur always (e.g. Object, Attribute), often (e.g. State), or only sometimes (e.g. Path) in ETAG models. The following concepts are at least necessary for the canonical basis (between brackets, the application concepts that require their presence in the canonical basis):

Actor	(person, agent)
Object	(message)
Attribute	(topic)
Value	(threshold)
State	(at-time)
Task	(specify_interest_topic)
Event	(create_message)

At this stage it is not completely clear whether the Place concept is required. Places occur as the whereabouts of users and conference events but within the Comris system there are no objects that move between places. As such, places are only necessary when the contents of the messages is modelled. The Path concept is certainly not required.

6.2.3 Step 3: The Preliminary List of Tasks and Basic Tasks

In the third step, a list of the user's tasks or commands is created to acquire a first approximation of the contents of the dictionary of basic tasks. At first, there is no need to be precise about how 'basic' tasks are, as interest is only in delimiting the lower boundary of the UVM in the form of an as complete as possible list of so-called primary tasks. Only during the refinement step (see below), secondary tasks will be added as tasks which enable the user to invoke primary tasks within the context of the specific UVM chosen. As an example, in filing systems the concept of a directory requires a "set current directory" command to use it, even though it does not add any new -primary- functionality.
When the task list is satisfactorily complete, the dictionary of basic tasks can be found by grouping functionally similar commands and abstracting away details related to how the functionality is offered to the user. Examples of such details are: whether or not user confirmation is required to continue, and optional repetition (do for all, repeat unless cancel) of the command. Each group may now be represented as a single basic task, with variations for the different object types and attributes, similar to the example of Task-Action Grammar (see: paragraph 3.3.1) in which several different ways to move a cursor are reduced to one rule and two features to represent the direction and amount of movement.
A special case of abstracting away details is the identification of menu tasks that refer to the optional repetition or cancellation of commands and require user decisions while executing. As such, "do-for-all" or "repeat-until-cancel" tasks cannot be described in ETAG-proper as a

pure competence model. Menu-tasks are common in user interfaces and therefore, instead of leaving them out of the model, they are specified on top of the ETAG model or included as a special kind of task, as menu tasks. Tasks that may be interrupted by hitting special key combinations or tasks that require the user to respond to a "Continue/Abort" request are treated as special basic tasks, since describing them without loops and interruptions would only clutter the specification and not add useful information.

Guidelines:

Make a list of available or required commands:
(set interest topic, play message, ...).
Describe in words what each command does:
(enter a topic word, present a message on the earphone, ...).
Group functionally similar commands:
(play a message/browse a message, ...).
Dissect Menu-tasks into Basic tasks by analysing user control:
(lower the threshold/raise the threshold, ...).
Distinguish system-controlled task elements from user-controlled elements:
(user remove message/system remove message, ...).

Example: In the model of the Comris system there are three sets of tasks. First, there are the user's tasks related to specifying topics-of-interest:

- specify_topic — enter the topic for a new interest
- specify_topic_value — enter a value between 1 and 5 for the particular interest

To the user, specifying a topic and specifying its interest value are presumably two parts of a single task unit because they are related to specifying one's interest and because they are performed together and using the same interface. In a complete ETAG specification of the Comris system there would be other "housekeeping" tasks connected to this interface, such as logon, logoff, specify a userid and select a current or a new topic to work on. In the example only the two listed tasks are included to explain in functional terms where the concepts come from that are required to understand the workings of the wristwatch interface.
Second, there are the tasks of the system or the system agent in presenting messages to the user and to remove the messages that are no longer relevant:

- create_message — a magic menu task to present a new message to the user
- remove_old_messages — a task to scan messages and remove the outdated ones

Only the required system tasks are included which is, as far as ETAG modelling is concerned, perfectly legal because users do not need to understand how, for example, the speech output of a message is actually generated. In the menu task to create and present a message to the user the word "magic" is used because the principles of the task will be described but not how it will actually work, because in an agent system that would require a complete moment-to-moment description of the system. The task to remove the old messages is required to explain why, every now and then, messages may also disappear without the user having issued a command to remove them. This happens, for instance, upon answering a question or when a particular event has finished.
Third, there are a number of user tasks that relate to specifying the settings for message presentation and managing the messages after their initial presentation:

- set_threshold — a menu task to raise and lower how often messages are presented
- set_volume — a menu task to raise or lower the volume of the earphone
- set_current_message — a menu task to select the next or previous message if there is
- play_message — present a message in spoken form or a text on the display
- browse_message — present a message in textual form on the display

- remove_message delete the current message

The user's "set_something" tasks are menu tasks because they combine two simpler commands to increase and decrease a value. The tasks to play and browse messages are an example of why it may be necessary to iterate between the specification of the perceptual interface, and more specifically, the rewrite rules and the dictionary of basic tasks. The task to adjust the volume has not been listed earlier in the raw list of concepts because it was simply overlooked or because the detailed conditions of using speech output had not been thought out before.
Another example of iteration due to design detail is that during the design of the Comris wristwatch, "playing with prototypes" inspired the idea that there could be a single task to play a message or that there could be two different commands to listen to and to read a message, respectively. At the time during the design process when the dictionary of basic tasks is first specified such choices may not yet be made. In addition, it may be noted that when the dictionary of basic tasks would not have identified a task to browse a message, users might have invented or discovered it by turning down the speech output volume.

6.2.4 Step 4: A First Specification of the Elements of the UVM

The fourth step is concerned with establishing a first specification of the UVM as a starting point for iterative refinement. From the dictionary of basic tasks follows which events should be defined in the type definition, or, when these can be defined using only the canonical concepts, in the canonical basis. This may, for instance, be the case with Copy and Move commands which can be defined in terms of Objects and Places in general.
The list of object types that needs to be defined is -analytically- derived from the available basic tasks and events. If a task may be defined using either object or attribute types, it is advised to attempt object(type)-based definitions first. Object-based definitions generally lead to more concise models, and although attributes provide general purpose glue (perhaps: semantic sugar), they increase the risk of creating a model which is invalid as a user conceptual model.
There are now enough starting points to approach the chicken-and-egg problem of defining the UVM as an object-oriented structure, but it may help to create the draft object, space and -if possible- attribute hierarchies first. When all the task, event, object and attribute lists or hierarchies are laid down and all the ingredients to model the primary functionality are available, there is an opportunity to reason about (and improve) the user interface as a whole, before adding the specification details.
The end-product of this stage is the specification of the UVM which should include, at least, the definitions of the objects and events. It may be possible to lay down the set of values of some of the attributes at this stage, but the aim is to provide an overview rather than a specification that is as detailed as possible.

Object guidelines:
Make a precise list of objects:
-Object attributes and states are set by events and tasks:
(set current-message, set topic-name, ...).
-Objects provide places for other objects:
(place.on(message, message-list, ...).
List the objects in a type containment hierarchy:
(message-list -> message -> event-type, ...).
Determine for each object how many instances there are, and if it is a constant:
(1 message-list, static; n messages, dynamic; each 1 topic, static, ...).

Attribute guidelines:
- For each object, list the attributes that uniquely distinguish it from others:
  - (conference-event -> attribute: time, location, ...).
- List the value type of each attribute:
  - (interest-topic: string; interest-value: digit, ...).
- List the attributes in a type hierarchy:
  - (message-list -> message -> conference-event, ...).

Place guidelines:
- List the places provided by each object:
  - -Objects are created, deleted and moved between places:
    - (create.on(message-list, message), ...).
  - -Objects may need to be identified by a place:
    - (previous-message, current-message, ...).

Path guideline:
- List the paths, objects are on or move along (not applicable).

State guidelines:
- List all states that express relations between objects, attribute-values and places:
  - (exists(current_message), volume < 10, ...).
- Remove any states that are not meaningful or never used:
  - (at-position(message, message-list, 1), ...).

Event guidelines:
- List all tasks as events with the same name:
  - (task remove-message -> event remove-message, ...).
- Dissect each event into primitive or 'general' events:
  - -Dynamic object instances are created and deleted:
    - (kill-on(current-message, message-list), ...).
  - -Objects are moved, copied, have attributes and places set by primitive events:
    - (person-topic-name = string, ...).
- For each event list which objects, attributes, etc. are manipulated and how they are:
  - (copy-to(message, display) -> copy-to(object, place), ...).

Example: Specifying the user's virtual machine is the most fuzzy process in ETAG modelling because of the yo-yo nature of specifying things that are not yet there, and specification involves adding low-level details and verification requires a clear top-down overview of the model. Because changing the model at one place inevitably requires changes elsewhere it is good practice to split the model into independent modules that are easily overviewed and to split the process of creating the model into different specification/verification steps that are easily managed.

To model the example, the specification is split in two parts, one part around the specification of the conference in general, and one around specifying the messaging part. Furthermore, the process is split in steps to create a draft specification to provide general overviews and a final specification that provides full details.

The specification of each part (the conference and the messaging) starts with a draft specification of the object and object-attribute hierarchy which are mutually verified and adapted. For each part a draft is created of the type specification which is first locally verified and adapted to the type hierarchy and then globally verified and

adapted to the specification of the other part. After the draft step, the final specifications of the conference and messaging parts are created and verified independently (leaving aside the odd ad-hoc adaptation), after which the final specifications of the two parts are mutually adapted to each other.
Verification and adaptation proceeds in iterative cycles driven by identifying notation problems (e.g. syntax errors and inconsistencies) and by identifying problems in the relation between the specification and the intended design (e.g. semantic errors and omissions). After each of the two parts was specified, the parts themselves were mutually adapted.

A core object in the example of the Comris system is the conference schedule, as the place where all the events reside and from where information about the events is selected and copied into user messages. Each such event is characterised by an event type, a topic, data about a person, a time, and a location. The time of an event is a time-slot that consists of a start-time and an end time. Note that the end-time concept is not used in user messages even though it is required to generate sensible advise.

Part 1 - Conference concepts - Object hierarchy

```
conference
        person_list
                person
                        person_name
                        person_affiliation
        location_list
                location
                        location_name
        topic_list
                topic
                        topic_name
        schedule
                conference_event
                        event_type_name
                        event_topic_name
                        person_name
                        timeslot
                        location_name
```

Part 1 - Conference concepts - Object specification

```
type [conference isa object]
        themes:   person_list, location_list, topic_list, schedule
        remarks:  a conference isa event but not like its own conference_events

type [schedule isa object]
        themes:   conference_event (*x)
        places:   place.on (schedule)
end

type [conference_event isa object]
        attributes:     event_type_name, event_topic_name, person_name,
                        timeslot (start_time, end_time), location_name
end

type [event_type_name isa attribute]
        values:   one-of ["Presentation", "Meeting", "Appointment", "Demonstration"]
end

type [event_topic_name isa attribute]
```

```
        values:   one-of  ["usability engineering", "augmented reality", "groupware", ... ]
end

type [person_name isa attribute]
        values:   one-of  ["John Peterson", "Mrs Susan Bowen", "Olivia Rogers", ... ]
end

type [timeslot isa attribute]
        themes:  start_time, end_time
        values:   day-hour-minute
end

type [location_name isa attribute]
        values:   one-of  ["the lecture room", "room 22", "", "the commons", ... ]
end
```

Messages in the Comris example present information about conference events. Each message is created and presented to the user by the system agent, and they are kept in a user specific list to which both the system agent and the user have access. Messages are created on the basis of information about the interest topics and their interest values of the particular user. Messages are only presented to users when they pass some user-settable threshold that is related to the frequency of the messages and the minimum relevance of the events about which users wish to be informed. Frequency requires that there is a general concept of the current time. The concept of relevance is obviously related to user's interest values but because it is also related to the time and whereabouts of the user, there must be a user specific current location.

Part 2 - Messaging concepts - Object hierarchy

```
agent
        owner

current_time

person
        message_list
                message
                        event_type_announcement
                        event_specification
        person_interest_list
                person_topic
                        topic_name
                        topic_value
        current_location
                location_name
        threshold
        volume
```

In ETAG modelling there is a distinction between the person type or the user and the user's system agent because as agents, they are both responsible for executing particular tasks, and because ETAG models the system from the user's point of view, only the system agent needs to be specified as a type. In modelling for software engineering purposes the user's point of view is not adequately represented: it is either reduced to a tunnel view on the individual tasks of the user or it is hidden in a model of the whole system viewed from the outside. In UML modelling (Jacobson, 1995; UML, 1999), for instance, the user is either represented as just one of the external agents who is involved in a number of use-cases or as an entity that completely coincides with and is represented by the system agent.
The type hierarchy as specified corresponds to the UML idea of the user as a particular agent. To re-create the user's point of view in the ETAG model, below, the "person agent" is removed as a supertype while retaining

entries like message_list and threshold as entries that the user must know about. One step further is to model the tasks of the personal agent in creating and removing messages as embedded events (van der Meer, 1992), as event that just happen to happen.

Part 2 - Messaging concepts - Object specification

```
type [agent isa actor]
end

type [current_time isa object]
        values:   day-hour-minute
end

type [message_list isa object]
        themes:  message
        places:   place_pos(x)(message_list)
end

type [message isa object]
        attributes: event_type_announcement, event_specification
end

type [event_type_announcement isa attribute]
        values:   one-of ["Proximity", "Proposal", "Reminder", "Suggestion"]
end

type [event_specification isa conference_event]
        supertype: conference_event
end

type [person_interest_list isa object]
        themes:  person_topic
        places:   place_on(person_interest_list)
end

type [person_topic isa event_topic]
        attributes:           topic_name, topic_value
end

type [topic_value isa attribute]
        values:   one-of  [1 ... 5]
end

type [current_location isa location]
end

type [threshold isa attribute]
        values:   one-of  [-4 ... 4]
end

type [volume isa attribute]
        values:   one-of  [0 ... 10]
end
```

The entries in the dictionary of the basic tasks were listed earlier. At this stage they are specified in detail and linked to the type specification to enable the specification of the events, and as a starting point for the completion of the UVM specification with the entries that relate to the secondary tasks.

Part 3 - Task and event concepts - Task specification

```
type [specify_topic isa task]
    effect:   [create_person_topic isa event][set_person_topic_name isa event]
    T1 [person_topic_name: string(s)]
    remark: enter the topic for a new interest

type [specify_topic_value isa task]
    effect:   [set_person_topic_value isa event]
    T2 [person_topic_value: value(v)]
    remark: enter a value between 1 and 5 for the particular interest

type [create_message isa embedded task]
    effect:   [create_message isa event][play_message isa task]
    remark: a magic menu task to present a new message to the user

type [remove_old_messages isa embedded task]
    effect:   [remove_old_messages isa event]
    remark: a task to scan messages and remove the outdated ones

type [set_threshold isa task]
    effect:   [set_threshold(direction) isa event]
    T5 [set_threshold: event(e), direction: boolean(b)]
    remark: a menu task to raise and lower how often messages are presented

type [set_volume isa task]
    effect:   [set_volume(direction) isa event]
    T6 [set_volume: event(e), direction: boolean(b)]
    remark: a menu task to raise and lower how often messages are presented

type [set_current_message isa task]
    effect:   [set _current_message(direction) isa event]
    T7 [set_current_message: event(e), direction: boolean(b)]
    remark: a menu task to raise or lower the volume of the earphone

type [play_message isa task]
    effect:   [play_message isa event]
    T8 [play_messsage: event(e)]
    remark: present a message in spoken form or a text on the display

type [browse_message isa task]
    effect:   [browse_message isa event]
    T9 [browse_message: event(e)]
    remark: present a message in textual form on the display

type [remove_message isa task]
    effect:   [remove_message isa event][set_current_message]
    T10 [remove_message: event(e)]
    remark: delete the current message
```

Part 3 - Task and event concepts - Event specification

```
type [create_person_topic isa event]
      preconditions:    exists(person_interest_list) && length(person_interest_list) < max
      postconditions:   length(person_interest_list) += 1
      effect:           create_on(person_interest_list, person_topic(p))

type [set_person_topic_name isa event]
      parameters:       person_topic_name(s)
      preconditions:    exists(person_topic(p))
      effect:           person_topic(p).name = s

type [set_person_topic_value isa event]
      parameters:       person_topic_value(v)
      preconditions:    exists(person_topic(p))
      effect:           person_topic_value(p) = v

type [create_message isa event]
      parameters:       person_topic_list, schedule, current_time, threshold
      preconditions:    exists(person_topic_list)
      effect:           select conference_event(c) from schedule for relevance
                                    if (relevance > threshold) do
                                                create_on(person_message_list, message)
                                                copy_to(c, message)
                                                current_message = message
                                                play_message
                                    od

type [remove_old_messages isa event]
      parameters:       person_message_list
      preconditions:    exists(person_message_list)
      effect:           for message(m) in person_message_list do
                                    if (relevance(m) < relevance_min) remove(message(m))

type [set_threshold(direction) isa event]
      parameters:       threshold, direction(b)
      preconditions:    (direction = plus && threshold <  4) ||
                        (direction = min && threshold > -4)
      effect:           threshold += b

type [set_volume(direction) isa event]
      parameters:       volume, direction(b)
      preconditions:    (direction = plus && volume <  10) ||
                        (direction = min && volume > 0)
      effect:           volume += b

type [set _current_message(direction) isa event]
      parameters:       current_message, direction(b)
      effect:           if ((direction = plus && exists(current_message+1) ||
                                    (direction = min && exists (current_message-1)
                        then current_message += b

type [play_message isa event]
      parameters:       current_message
      preconditions:    exists(current_message)
      effect:           copy_to(currrent_message, speech_output, volume)
```

```
type [browse_message isa event]
      parameters:        current_message
      preconditions:     exists(current_message)
      effect:            copy_to(currrent_message, display_output)

type [remove_message isa event]
      parameters:        current_message
      preconditions:     exists(current_message)
      postconditions:    if (exists(current_message+1)) current_message += 1
                         elseif (exists(current_message-1)) current_message -= 1
                         else current_message = nil
      effect:            kill_on(message_list, current_message)
```

6.2.5 Step 5: Refine the Specification of the UVM until Completion

The fifth step, or iterative refinement of the UVM repeats the above steps from (almost) the beginning to complete the interface design, using the preliminary specification of the UVM as a base-line. The refinement process consists of three parts: the addition of secondary functionality, the completion of the UVM in terms of consistency, coherence, etc. and the filling in of remaining details. Adding secondary functionality consists of creating the tasks and their UVM specifications required to enable the user to make use of the primary functionality. Just like a pencil device requires a piece of paper, a command to save a piece of work requires the ability to specify a storage space such as a current disk. Likewise, to use an editor or an email system generally requires a command to enter (and exit from) the specific task environment.

After completely specifying the functionality, it may be useful to (e.g. formally) analyse the consistency, uniformity or orthogonality (read: the complexity and size) of the UVM. It may be interesting to determine if, for example, certain file and directory operations cannot be rewritten as general object operations.

Filling in the missing details is just what it says: adding any lacking definitions of Objects, Attributes, etc. and taking care of the syntactic specifics. Note that, although the refinement process as described may suggest a linear sequence of substeps, one should expect a process of iteration.

With this step, the most important part of the ETAG specification, the User's Virtual Machine should be completed, and now that the users' view of the application semantics has been defined, the design and implementation of the functionality of the computer system can take place, as well as the specification of the perceptual interface.

Example: Upon completing the first specification of the elements of the UVM (paragraph 6.2.4) a number of things may be noted. First, the event specification refers to items that have not been specified, such as the events to kill-on and copy-to and the events to set the current time and location. Such canonical events do not need to be further specified because it may be assumed that they belong to the general world knowledge of the user, but they still should have been listen in the canonical basis.

Secondly, details in the task and event specification indicate the need for minor changes and adaptations in the UVM. To describe the tasks to create and remove messages requires a specification in the UVM that messages take up a certain position in the message list and can be referred to by their index position.

Finally, a number of elements and events is merely listen but not specified in any way. There is no description of what speech-output is and how it is influenced by the volume setting. Likewise in the create_message and remove_old_messages events there is no specification that describes what the user should know about their workings. It may be assumed that a concept like relevance simply belongs to the canonical basis but, from design considerations , it may be more appropriate to indicate in more detail how it relates to e.g. time-aspects and the

values of the user's interest topics. In that case, the specification may easily blow up and the trade-off between the size and clarity of the specification as a design tool may become acute.
Because concern in this chapter is with describing the process of creating an ETAG representation, there is little reason to complete it beyond the current state which is sufficient for the specification of the perceptual interface. For a complete example of an ETAG user virtual machine, see chapter 7.

6.2.6 Step 6: The Specification of the Perceptual Interface

The design of the perceptual interface that is concerned with the specification of the presentation interface and interaction language, is a relatively straightforward process in which the UVM functions as an upper delimiter and characteristics of the i/o devices set the lower boundaries. It is largely a matter of taste whether or not to use ETAG for this part or do this on a trial-and-error basis. For the presentation interface, (drawing) tools and intuition have to be used, since no ETAG specification method has yet been developed for this.
However, since a complete conceptual specification of the user interface is available in the ETAG model, it would be a waste not to make use of it, as a useful starting point to guide perceptual interface design or as a way to check if the perceptual interface specification is appropriate and complete. For automatic generation of the perceptual interface, a number of approaches and tools has been proposed (e.g. Wilson et al., 1993; Bodart et al., 1994; Balzert, 1995; Lauridsen, 1995), but it remains to be seen how easily ETAG representations may be used within these systems. A less automatic approach to presentation interface design is also possible, guided by the same rules and questions that have been built into the automatic generation tools. For a WIMP like interface one might ask:

- if there is more than one task environment, how many task windows are needed?
- for each object (type) and place: if it should be presented, and how?
- for attributes and relations: how to present and manipulate them.

Such a non-automatic generation of the presentation interface is essentially similar to generating the interaction language from the UVM, except that there is no visual-grammar readily available to make a neat distinction between the various levels of information (semantic, syntactic, etc.). Another point that is important in this context is that most user interface builders and toolkits are concerned with the general graphical elements of interaction: they provide the material to set up an interactive environment (interact through) but offer little or no assistance when it comes to accessing the tasks and information the user wants to access (interact with). Toolkits provide the windows, but not what to display in them; they provide the menus and buttons, but not what tasks to provide them for. Since ETAG is all about the user's tasks and task environment, it nicely complements such interface builders and toolkits.

To specify the interaction language it is, in our opinion, best to use the ETAG format to ensure compatibility with the UVM specification and a neatly layered specification. Because ETAG's propositions do not allow an easy grasp of dialogue sequences, but are a useful tool to show structural (in-) consistencies, we suggest a combination of ETAG with a user interface builder in the traditional sense. The specification of the interaction language consists of specifying the following four levels, in -again- a loosely top-down order:

The **command syntax**, or the order of specifying commands, options and command elements, on the basis of what is most appropriate for each individual task and overall uniformity.
The **reference style** or interaction style of addressing commands, setting options, etc. For simple pc-type applications this may be the starting point for specifying the interaction language.
The **naming** or visual labelling comes down to choosing a name, menu-label or icon for each command, etc. that suits the experience, language, etc. of the intended users.
The **behavioural actions** specify how interaction takes place in terms of acting upon the hardware.

The uniformity (consistency, coherence, etc.) of the interaction language between any of these levels can be inferred from the similarity of the rewrite rules, and in very general terms, an overall uniform interaction language will have few rules at the top levels and many, as many as there are different tasks, at the bottom levels. Yet again, except when designing the most trivial interaction language, several iterations should be expected within the interaction language specification. It may also be required to iterate back to the UVM specification once or twice, for example, when it shows that small UVM changes would lead to a much more uniform interaction language.

Example: Modelling the perceptual interface of the Comris example is relatively easy because, early in the project, detailed requirements have been stated about the wristwatch interface, in relation to the point that, as a wearable interface, it is a source of important design constraints.
With respect to the output side, the parrot interface should be equipped with an audio output device like a headphone and with a display screen providing room for 2 lines of 16 characters (or icons) for the presentation of messages in spoken and textual form.
With respect to the input and feedback, the requirements that there should be a slider or dial and four or five buttons to:

- adjust the volume
- repeat a message
- interrupt or postpone a message
- suppress or delete a message
- raise or lower the context threshold
- move 'up' to the next message
- move 'down' to the previous message
- give a positive or "yes" response
- give a negative or "no" response
- elaborate a message to present additional information

When designing the prototypes of the Comris parrot, it was not certain whether the agent system could provide for the last function, to elaborate a message. Instead, it was decided to add an option to show a message in abbreviated form or to have the complete message scrolled by on the display and/or spoken via the audio channel. Finally, since it is clear from the list that there are more functions than there are buttons (five) available, and alternative function-to-button mappings are required, a mode function was added.
To investigate the usability aspects of the parrot, four different interfaces were designed and implemented in Tcl/Tk, each according to a general principle to deal with situations in which there are more functions than there is room for, including the use of a mode button to change sets of available commands, a control key modifier button or double clicking to modify the meaning of the buttons and, finally, the use of multiple dialogue steps for a most efficient or a most consistent dialogue, respectively (de Haan, submitted). Here, the mode interface will be described as an example of how to create a specification of the perceptual interface.

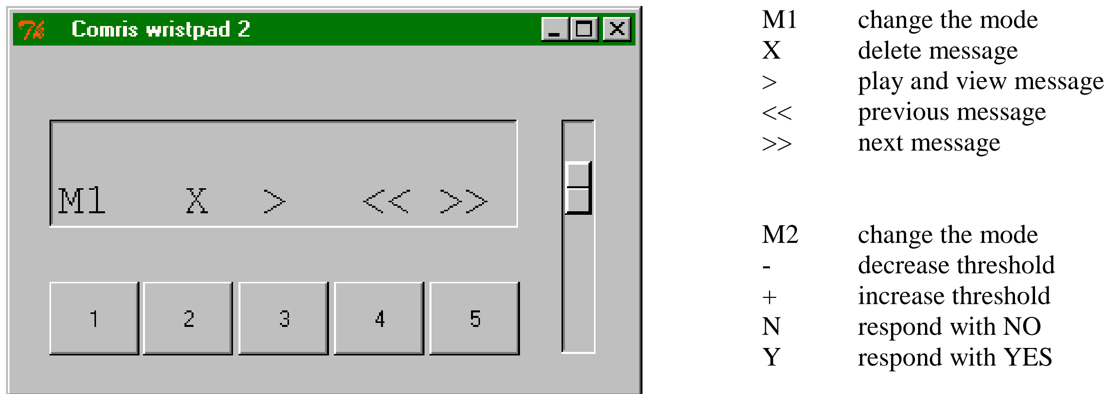

Figure 1. The Mode Interface prototype in Comris.

The design of the prototypes of the parrot interface concentrated on two main issues: the presentation of text on the display and the button assignments. Starting with the requirements about the display size and the synchronisation of text and speech, the design of the text presentation was a relatively easy matter that was solved by trial and error. The message text had to scroll over the two lines of the display to allow for the synchronisation between text and speech. After considering a number of scroll-options (e.g. by letter, by word or by phrase), the choice was made for letter-by-letter scrolling with line-breaks at word-endings, and scrolling speed was adapted to speech rate. In order to display abbreviated messages, a command to show a message was added and a set of abbreviated messages was created such that each fitted on the display without the need to scroll.

In the design of the user interaction with the different prototypes it was given that a slider would be used to set the volume and that there would be five buttons. The starting point for the button-interaction was to use a general principle to help overcome the problem of having fewer buttons than commands. In the mode-interface, one button would be used to set the mode. Assigning commands to the other buttons was done on the basis of safety and positional constancy (in each different mode, similar commands use similar buttons), the expected frequency of use (frequently used commands are placed in the default mode).
Given that modes are considered harmful, the use of a third mode was avoided by leaving out the command to browse a message. Messages can be browsed by playing them with the volume turned down. The command to show a message in abbreviated form was combined with selecting the next or previous message. Finally, to avoid that users have to learn the command-to-button assignments, the buttons were labelled by letter-icons on the display which appeared upon pressing the button or following a command to show a message, after a period of inactivity.

In the Comris project, the prototypes were used for usability experiments (see: de Haan, submitted). The moded interface prototype is used to exemplify the specification and analysis of an interface in ETAG. The preliminary dictionary of basic tasks does not correspond to the tasks that the interface provides. Two tasks are lacking. In the UVM part of the example, the interactive use of the system was left out to save effort and trees. For a complete specification of the perceptual interface it is necessary to add another task for answering questions:

type [answer_question isa task]
    effect: [answer_question isa event]
    T12 [answer_question: event(e), direction: boolean(b)]
    remark: answer a question (e.g. to an invitation) with yes or no

An additional basic task is required to set the command-mode:

type [set_mode isa task]
    effect: [set_mode isa event]
    T11 [set_mode: event(e)]

remark: change the command mode

The requirement to first select a particular mode before selecting a task would imply that, from a theoretical point of view, all the tasks like playing message, etc. would change from basic tasks into menu tasks because they require that the user makes a decision while invoking the associated task commands. In practical terms, the task as individual entities may still be regarded as basic tasks. The same reasoning applied to the task to browse a message. In the mode interface, reading the text of a message without it being spoken requires that the user first turns down the volume and then plays the message. As a non-basic task, browsing a message can be left out from the specification. When analysing the usability of different interfaces this could create problems, but in the current example it does not matter to leave the task out.

From the task and event specifications, it follows that there are only two different entities that are referred to in the interaction language of the mode interface: specifying an event and specifying a direction, making the description of the command syntax with specification level production rules very simple:

T5 [set_threshold: event(e), value: boolean(v)] ::= specify[event:e, value:v]
T6 [set_volume: event(e), value: boolean(v)] ::= specify[event:e, value:v]
T7 [set_current_message: event(e), value: boolean(v)] ::= specify[event:e, value:v]
T8 [play_messsage: event(e)] ::= specify[event:e]
T10 [remove_message: event(e)] ::= specify[event:e]
T11 [set_mode: event(e)] ::= specify[event:e]
T12 [answer_question: event(e), value: boolean(v) ::= specify[event:e, value:v]

Similar to all mouse-based interfaces (see: e.g. chapter 7), the description of the reference style with the reference level production rules is also very simple and only "complicated" because referring to the volume slider differs from selecting buttons by their labels:

Specify [event <> set_volume] ::= Icon(label[identifier: i])
Specify [event == set_volume] ::= Form([direction: i])

The full description of the command and argument names at the lexical level of the production rules requires eleven rules:

Icon [event: set_threshold, value: up] ::= [label, "+"]
Icon [event: set_threshold, value: down] ::= [label, "-"]
Icon [event: set_current, value: next] ::= [label, ">>"]
Icon [event: set_current, value: previous] ::= [label, "<<"]
Icon [event: play_message] ::= [label, ">"]
Icon [event: remove_message] ::= [label, "X"]
Icon [event: set_mode] ::= [label, "M?"]
Icon [event: answer_question, value: yes] ::= [label, "Y"]
Icon [event: answer_question, value: no] ::= [label, "N"]
Form [event: set_volume, value: up] ::= ["up"]
Form [event: set_volume, value: down] ::= ["down"]

In this example, the ETAG representation was created as a specification for design purposes. For design purposes it is sufficient to leave the eleven rules as they are. If the specification had been made to analyse the usability of the interface, the lexical rules could have been replaced by one high-level rule that uses the concept of a "known item" from Task-Action Grammar (Green et al., 1988) and lower level rules to distinguish the items, to reflect that icon-labels are so general that they do not require users to learn what they mean.
At the keystroke level of the production rules it is necessary to distinguish between the two forms of physically interacting with the interface: clicking on a button and dragging a slider, and there are two keystroke-level rules:

Icon [label, l] ::= press_button(l)
Form [l] ::= drag_slider(l)

6.3 ETAG-based User Interface Design

The method of user interface design on the basis of the ETAG notation has been developed to summarise some of the knowledge and experience of our group regarding Software Engineering, cognitive psychology and ergonomics by means of a teachable and practically applicable design method. In ETAG-based design, design is regarded as the progressive specification of a competent user's knowledge about a computer system in order to perform tasks with it. As such, the focus is on users, tasks and task-knowledge, rather than on the creation of the artefact. Figure 2 graphically presents the structure of the design process. Detailed information about ETAG-based design can be found in de Haan (1994, 1997, chapter 4).

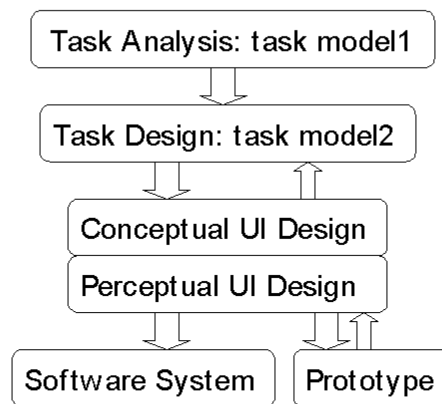

Figure 2. The Main Steps in ETAG-based Design.

In short, ETAG-based design is structured as follows: during task analysis, information about users, their tasks and the work environment is collected, from which the task structure of a job or function is represented in a Task-model 1. Task-model 1 refers to all the results from analysing the old task situation, and which is centred around an abstracted or idealised task model that corresponds to what Wilson et al. (1993) call the Extant Task Description, and Lim and Long (1994) refer to as the Extant Task Model. On the basis of task-model 1, improvements may be suggested and represented in Task-model 2. In terms of Wilson et al. (1993), Task model 2 is the Designed Task Model. On the basis of the functionality of the user's computerised tasks a new user interface can be designed, according to the steps described in the previous sections. During the design process, the user interface specification can be fine-tuned with the (changing) user wishes and task environment by means of various kinds of prototypes. After the user interface has reached stability, it is ready to be put (further) to work through computer code, and installed at the work environment.

The interesting point about ETAG-based design is not the particular sequence of design phases. In de Haan (1997, chapter 4) we also show that ETAG actually follows a standard design model. ETAG is neither special in focussing on user knowledge. For example, ADEPT (Wilson et al., 1993) is also concerned with knowledge. What really sets ETAG-based design

apart is that user interface design is regarded as the specification of (an idealised form of) a user's mental model. In this respect, the ETAG notation is comparable to Moran's Command Language Grammar (CLG; Moran, 1981), although CLG has never been turned into a design method.

Because of the notation, and to some extent the particular steps in the design method, it is relatively difficult not to consider the user in ETAG-based user interface design, even as a non-expert in HCI. This is a consequence of the early adoption of the idea that Software Engineering and HCI (or cognitive engineering, for that matter) have a different view on user interface design and that, as such, it is better to keep responsibilities separate (van der Veer et al., 1988). In ETAG-based design, the 'strategy', if that is the appropriate word here, for co-operation between HCI and Software Engineering is to provide a strong HCI method, and opportunities to connect both types of specifications. Using a strong HCI method with the usability of a design as its major aim also avoids the traditional role of HCI to do the polishing after the main decisions have long been taken.

Other methods which, in our view, run a greater risk of not delivering usable systems are, for example, the ADEPT method (Wilson et al., 1993), the framework of Lim and Long (1994) to facilitate HCI use in system design, and at the other extreme, the extension of SSADM to incorporate HCI concerns (Summersgill and Browne, 1989). ADEPT comes relatively close to ETAG in specifying user knowledge, but it is explicitly meant for use next to software engineering methods.

The strategy of Lim and Long is to persuade system designers to not forget HCI through using a SE notation (JSD) within a design framework with opportunities for using HCI methods. In this case, using the framework and HCI methods is up to system designers, but, once used, a usable system can be expected.

Summersgill and Browne suggest that usable systems can be designed through offering additionally required activities and reports within the -already huge- SSADM approach. Here, not only adhering to the HCI steps depends on system designers, but also the way in which these steps are 'filled in' depends on the attitude and expertise of the system's engineer. In our humble opinion, we cannot expect very much of such a way to create usable systems.

6.4 Conclusions

As a most general conclusion, this chapter shows that it is feasible to take a representation language that allows for psychologically valid formal representations of user interfaces (chapter 3) and use it as a "limited theory" to solve the practical problem of providing a model that is useful for design, without solving the basic questions in cognitive psychology. This does not mean that an applied cognitive psychology is feasible, but rather that particular psychological methods and concepts can gain practical validity. ETAG does not solve cognitive psychology, because it leaves questions about human attention, memory and information processing unanswered. It neither proves the feasibility of an applied cognitive psychology, because ETAG is based on a deliberate, and perhaps pragmatic, choice of psychological theories and concepts, rather than *the* common understanding in cognitive psychology. ETAG is among the approaches that show how cognitive psychological concerns, such as mental models, can lead to applicable tools that go well beyond the -otherwise erroneous- seven plus or minus two menu items type of guidelines.

As to user interface design, ETAG exemplifies that cognitive psychology should not merely be seen as a resource for "shoulds", which may and may not be effectuated, but rather as a resource for theory-based tools, whose utility may be proved in, and improve practice. In this respect, it is promising to note that ETAG is only one among other HCI design methods that consider user knowledge; together with ADEPT (Wilson et al., 1993) and Lim and Long's (1994) approach it may exemplify a trend away from user interface design as the user interface program specification, towards user knowledge requirements specification.

As a tool, ETAG is not finished. A relatively minor problem is that it is cumbersome to change ETAG representations once they are (fairly) complete. User interface design is highly iterative, and when change is difficult, improvements may not be sought. It was mentioned before that the propositional representation of the interaction language in ETAG is not particularly suitable to overview dialogue structures. For this, tool-based solutions are sought, using graphic representation and manipulation tools to increase the usability of the representation by hiding its details (e.g. Bomsdorf and Szwillus, 1999).
A larger problem in our view is the presentation interface. For practical purposes, this gap is filled by using drawing tools and interpreted languages like Visual Basic and Tcl/Tk to design and prototype small applications. As a matter of fact, in prototyping the Comris application, it turned out that there is a remarkable resemblance between the ETAG mode and the Tcl/Tk programs. When the presentation interface is regarded as -merely- the visualisation layer of the higher level concepts of a user interface, this is a perfectly acceptable solution. However, when the "visual layer" is regarded as the primary information source about the opportunities to interact with and about the hidden higher level concepts of the user interface, it will be clear that it requires a more methodological approach than trial-and-error.
This corresponds to the idea of Payne (1991) that the display is not just the medium to show feedback messages, but also a resource to inform users about the actions to take and to persuade users into following the proper dialogue paths.

This chapter is adapted from: de Haan, G. (1997). How to Cook ETAG and Related Dishes: uses of a notational language for user-interface design. The examples are derived from: de Haan, G. (1999b). The Usability of Interacting with the Virtual and the Real in COMRIS. De Haan, G. (2000). Presenting Spoken Advice: Information Pull or Push? De Haan, G. (submitted). Interacting with a Personal Wearable Device - Getting Disturbed and How to Control it.

Chapter 7:

Analyzing User Interfaces: ETAG Validation Studies

We are entitled to have faith in our procedure just as long as it does the work which it is designed to do – that is, enables us to predict future experience, and so to control our environment. [...] What justifies scientific procedure, to the extent to which it is capable of being justified, is the success of the predictions to which it gives rise: and this can be determined only in the actual experience. By itself, the analysis of a synthetic principle tells us nothing whatsoever about its truth.

Ayer, A.J. (1936).

Abstract

This chapter discusses using Extended Task-Action Grammar (ETAG) as a tool to analyse user interfaces. It reports and discusses the first investigations we have undertaken to determine if ETAG fulfils two requirements for use as a design model: if ETAG allows for a reasonably complete and accurate description of a variety of different computer systems and, to a lesser degree, if ETAG is able to draw attention to usability problems in the systems modelled.

Several studies are reviewed in which ETAG is applied to different types of computer systems, and to variants of the same system, in order to exemplify its use and reach conclusions about the validity and the general applicability of ETAG, and to determine which aspects of ETAG should be improved, and how this may be done. A secondary purpose of these studies was to collect material to enable development of a methodology to create ETAG representations that was described in the preceding chapters.

7.1 Introduction

From the point of view of Cognitive Ergonomics three different but closely related types of analysis should be performed in designing new user interfaces or redesigning existing ones:

Task analysis is the main method to obtain a systematic overview of the user's tasks, and to determine which elements of the user's tasks can or should be supported by the computer. An analysis of the design problem, or the problem space of an application can be a part of task analysis, or can be used to complement it.

Problem space analysis is used to enable a mapping between the user's problem space (sometimes called the external task space) and an internal or computerised representation of it. This kind of analysis is used to model what users should know or should do to specify their tasks in terms of the interaction language of the computer system.

Analysis of the design representation is a recurring activity, taking place during the design of a user interface, to determine which questions are relevant at each particular design stage, and which questions should be answered next. It is meant to progressively reduce the freedom in representing the problem space, until a full fledged computer system is established, including the design of user documentation, training strategies and support.

Several formal modelling techniques have been proposed to model aspects of the analyses in human-computer interaction (see chapter 2; de Haan et al., 1991). Most of these methods fall into the category of problem space analysis, and are used to generate evaluative and predictive statements about usability aspects of computer systems. Some methods (e.g. CLG and ETAG) are primarily meant for the third kind of analysis: analysis for design purposes. Based on a preliminary specification of the users' tasks and task elements, and their representation in the computer domain, a method like ETAG can be used for progressive specification of the user interface and other user-related parts (e.g. training materials) of the computer system.
Our main research purpose is to develop a structured method to design user interfaces, considering the user's knowledge. It would be considered advantageous when such a method is also useful for usability evaluation purposes, and when it fits the methods for task analysis and software development. Because ETAG describes both declarative and procedural aspects of the user's knowledge, it offers possibilities to adapt to methods for task analysis. The neat structuring of ETAG descriptions, in combination with its use of objects and events, offers possibilities to adapt to the (e.g. object-oriented) methods of software development.

A problem with ETAG is that it has not been applied to a wide variety of different user interfaces and fields of application. The main example in Tauber (1990) deals with a simplified electronic mail system, and in Tauber (1988) a subset of MacWrite is used. Note that this situation is not unique to ETAG. For example, to our knowledge, ETIT has only been applied to two toy text editors in the original publication by Moran (1983). Likewise, Reisner (e.g. 1981, 1984) keeps repeating the same example of Action Language. Both van der Veer (1990) and Brazier (1991) report on using ETAG succesfully for systems design but they do not provide detailed accounts about the advantages and disadvantages of doing so.
The lack of published examples of formal methods makes it difficult to estimate the validity and applicability of methods to anyone not directly involved. Similarly, applying a model to only a small number or a small variety of computer systems may hide potential problems. As such, validation studies are needed, and models may be regarded more valid when they have been successfully tried on a range of different applications.

In this chapter we will report on our efforts in applying ETAG, both to different types of systems and to variants of the same type of system, in order to come up with conclusions about the validity and the general applicability of ETAG, and to determine which aspects of the method may be improved, and how this may be done. Note that we do not intend to produce any hard numbers on e.g. the predictive validity of ETAG, nor are we able to do so on the basis of the studies reviewed here. Instead we intend to review the descriptive validity of ETAG, which according to our view should be a first prerequisite for using formal models for design purposes, and perhaps, for using them at all. The studies reviewed here have helped to shape ETAG into the form that is described in the earlier chapters of this thesis. As such, most of the ETAG models in this chapter have not been created using the methodology described in the chapters 4 to 6, but rather by a more trial-and-error alike approach.
The studies which will be reviewed deal with two variants of the UNIX page tool "more", namely "pg" and "less", several variants of data correction in a spreadsheet application, and two variants of a telephone-based email system for personal computers of the Dutch PTT.

Section 2 discusses three studies in which ETAG had been applied to several different types of application. In section 3 the results of applying ETAG will be presented, including several conclusions.

7.2 Studies in Analysing User Interfaces

The general structure of ETAG may look rather simple, but our own research in applying ETAG to various user interfaces seems to justify one overall conclusion in the opposite direction. Creating ETAG descriptions is a complex and sometimes difficult process. Difficult, because it is not always clear what the most appropriate objects and attributes are, and because it may be difficult to oversee the consequences of choosing particular objects and attributes. Creating an ETAG representation can be complex because ETAG models are generally large. Using the method will be facilitated by clear and easily applicable directions on how to proceed.

A similar conclusion can be derived from a more theoretical consideration proclaiming the need to avoid value judgements in applying formal models. Value judgements are required when the analyst has the freedom to choose from among alternatives on how to represent a feature of a design in terms of the model used. An example in Task-Action Grammar (Payne and Green, 1986) is the choice for a particular semantic feature the user is presupposed to employ in distinguishing between tasks. In GOMS (Card, Moran and Newell, 1983), deciding on what counts as a routine task for the user requires value judgements. Card, Moran and Newell often used empirical data to decide upon unit tasks, but in designing real systems there are often no opportunities to do so (Reisner, 1984). Both value judgements as well as the requirement of collecting empirical data in advance degrade the practical usefulness of formal representations.

In ETAG, value judgements regarding identifying primitive tasks are excluded by simply defining these as elementary (e.g. irreducible) tasks, provided by the computer system. Nevertheless, also in ETAG the analyst still has some freedom of judgement. For instance, in identifying the objects the competent user knows about, or the ways of representing what happens when events occur.

What needs to be done first is to develop a method such that performing an ETAG analysis is made much easier, and that the need for value judgements is taken away by the availability of clear and enforcing judgement criteria. As a first step towards such a method, several studies will be reviewed in which ETAG is applied, in order to identify shortcomings and possible improvements of the method.

In this section, three studies are reviewed in which ETAG is applied to different types of computer systems, and to different versions of the same system. Its main purpose is to report difficulties in applying ETAG to determine where and how the available methods to create ETAG descriptions may be improved. The results of the particular studies are of secondary interest.

In section 7.2.1, a study of two page tools is reviewed. Section 7.2.2 is about analysing variants of a spreadsheet application that will serve as a more detailed example of problems involved in creating ETAG representations. Section 7.2.3 will discuss applying ETAG to two electronic mail systems.

7.2.1 Analysis of Two Page Tools

The primary purpose of a study by Yap (1990) was to explore and extend the descriptive validity of ETAG by analysing the process of applying it to two functionally comparable applications: the page tools PG and LESS. Page tools like PG and LESS are mainly used to browse through text files, one page of text on the display screen at a time. However, page tools also provide more powerful features, such as searching for strings, the choice for display options like line length, and binary code filters. The reason for studying page tools is that ETAG has no special facilities to deal with the presentation of information on the screen: the presentation interface. This study was meant to investigate how well ETAG could be applied to applications which put a heavy weight on information display.

The problem was approached by listing all information about the programs which seemed relevant and attempting to structure the information according to ETAG's requirements. Simultaneous to creating the ETAG representations, problems occurring in doing so were recorded, as well as inabilities of ETAG to represent aspects of the programs. The problems related to the descriptive abilities of ETAG were used to improve the theory behind the method. Based on experience and aiming at general applicability, a set of guidelines was derived for applying ETAG, which was subsequently used to create thorough representations of the programs.

The guideline part of Yap's (1990) study was briefly discussed in section 4.4. Here the results of the analyses and particular difficulties with ETAG will be looked at.

The programs PG and LESS are successors of the UNIX MORE utility, but offer additional functionality by e.g. allowing backward paging, pattern or string searching and line scrolling. Earlier ETAG studies dealt with programs in which presentation of information is only a small part of the functionality and not included in the analyses. In this study, page tools were investigated as programs whose main purpose is information display. Visual presentation of information is a feature which was not covered by ETAG at the time.

This study showed how ETAG could be adapted to address the semantical relation between the functionality of the computer system and (the content of) the information displayed. The exact visual layout of the display screen is still not addressed by ETAG. Provided that there are tools to facilitate the design of layout, it is not really necessary and it would only increase the size and complexity of ETAG representations superfluously.

To address visual information display ETAG had to be extended with more advanced concepts to indicate the organisation and placement functions of information on the display screen. In the following type hierarchy, the Visible Object at the topmost level is an abstract concept, only meant as a class label:

```
VISIBLE_OBJECT
        SCREEN
            WINDOW
            COMMAND_LINE
```

There is a static object of the SCREEN type, which is to provide place for the WINDOW and COMMAND_LINE. The COMMAND_LINE is the object type to provide place at the bottom line of the screen where the user's commands to the system are echoed. The WINDOW is the object type providing the remaining place on the SCREEN to visualise the

contents of a file. Note that within windowing systems, the SCREEN object would correspond to a window. Having defined the additional concepts to address the visible objects, what is further needed is a function:

event.MAP([PLACE], [PLACE])

which maps an active or current place in a file to the place on the WINDOW. This mapping function can be invoked explicitly by a user's basic task, such as by telling the system to go to the next page of data, or to move to the xx-th line of a file. More in general, the mapping event will be automatically invoked by the system as part of handling one of the user's basic tasks, such as searching for a particular word, or to go to a next page of data when that page would be partially beyond the end of the screen.
Most of the functionality of the programs could be represented straightforwardly as events, invoked by the user's commands which change the active or current place in a file, and a subsequent invocation of the mapping event. The ETAG representations of the two page tools were neither very different, nor very complex (about 15 pages). There are three points worth being mentioned:

First, part of the analyses involves aspects of the file system structure. Normally, this falls outside the scope of an ETAG analysis. The link with the context of the operating system may need to be addressed when command line arguments influence the program being invoked, or a program requires a specific type of file. For example, PG does not examine binary files, whereas LESS does.

Second, an aspect of both programs which causes some inelegance in the ETAG representations is how to describe the information on the display when a string to be searched is already on the screen, at one of its borders, or not yet on the screen. After a successful search, pagers will generally position files on the screen in such a way that the search word is placed on a xx-th line, but may not do so when the search word is already in the vicinity thereof. Not describing these effects would make the ETAG representation incomplete, but describing it does require the analyst to be occupied with a level of detail which does not seem to be very relevant for user interface design.

A **third** point is that ETAG does not address the exact layout of the display screen. Using the concepts developed in this study, only the relation between a system's functionality and the presentation of information can be described. ETAG may be extended to cover additional aspects of the presentation interface. ETAG representations are already large and complex, and this would increase the representations even more, whereas the precision of the specifications is only slightly increased. Also in terms of efficiency it seems better to cover these relatively 'low-level' aspects by means of the available interactive graphical interface builders like e.g. InterViews' Ibuild (Vlissides and Tang, 1991).

7.2.2 Variants of a Spreadsheet for Currency Exchange

Muradin (1991) and, de Haan and Muradin (1992) performed a study in the context of a small contest between models in human-computer interaction, a research initiative of the MacInter

group. Given specifications of variants of a currency exchange program, the contest aimed at determining which aspects of the specifications could (not) be modelled very well, and to arrive at conclusions about the usability, complexity, validity, etc. of the models. This analysis is presented at a fairly detailed level, not only to enable a close look at the strong and weak sides of ETAG, for example, in comparison to TAG, but more in particular to present a complete ETAG example of a 'real world' system.

The money exchange program is a spreadsheet program in which the user, a bank clerk, enters the requested amounts of various foreign currencies. The program calculates the appropriate equivalent amount of money in the national currency, taking care of transfer costs, exchange rates, etc. In addition to entering numeric currency data, the clerk also has to specify alpha-numeric data, such as the client's name and address.

Name: --
Address: ---------------------------------- -----
Zip / City: ------- --------------------------------
Id-no: -------------------------

Country	Cash	Cheque	....
....	0.00	0.00	0.00
....	0.00	0.00	0.00
....	0.00	0.00	0.00
....	0.00	0.00	0.00
....	0.00	0.00	0.00

Total amount: -----------.---
Fee à --.-- %: -----------.---
To pay: -----------.---

Figure 1. The Currency Exchange Program.

The differences between variants consists of the way in which cell entries can be changed and errors can be corrected. The most simple version (A) requires the user to re-enter the whole field, regardless of its length, which may be up to 40 positions. A more sophisticated variant (B) allows single character editing within a field, introducing insert and overwrite modes, as well as the need for a correction command. The third version (C) combines the editing facilities of versions A and B, offering both a field and a single character editing mode. Versions A and C each consist of two subversions.
Since it would not add essential data, the versions of the design problem will not be described in detail. For a detailed description of the design problem see: Wandke (1992, section 3). The purpose of the contest was to acquire information about the ability of modelling approaches to fit a hypothetical system, and, if possible, arrive at conclusions about the usability of the different versions of the money exchange program. For this purpose, many of ETAG's syntactic constructs were not necessary, and were left out to facilitate understanding of the structure of the representation.

The differences between variants were specified at a very low level, in terms of command keystrokes and the use of visual cues to indicate the currently active field or character position. The low level of the specification of the system variants caused some difficulty in creating ETAG descriptions of the system variants.
In design, ETAG is meant for top-down specification of a user interface, but here the systems were described at the lowest possible level of specification, i.e. physical actions. As such, the representations had to be created in a bottom-up fashion starting with the command keystrokes, inventing conceptual object types, attributes and events, in order to explain, or at least not contradict, the original specifications. As such, the most interesting parts of ETAG (e.g. the task semantics) are not given, and because the analyst still has to do the whole analysis, using ETAG to address keystroke differences seems a bit exaggerated, beforehand.
Certain parts of the design cannot be represented in ETAG, nor in most other formal models. These include the representation of the visual aspects and an evaluation of the way in which this presentation supports the user's tasks. A similar problem occurs with evaluating the usability aspects of the different design alternatives, because the design alternatives provide different functionality, whereas most, if not all, formal methods can only predict differences applied under the assumption of equal functionality.

Since ETAG is a competence model and does not address user performance aspects of interfaces, the usage data that was provided in terms of average field lengths, error rates, usage frequencies of field categories, etc. could not be used.
Finally, the specifications of the systems are incomplete. Therefore, it is not possible to generate full ETAG descriptions of their behaviour. The systems are described in terms of what users should do and what will be the reaction to that, but what is missing is a description of what will happen when the user acts differently. For instance, what will happen when too much data is entered to fit in a field, or a user presses the arrow keys before confirming the field being inserted into. Is it automatically confirmed, is rubbish entered as data, etc.

Prior to the actual analysis, the available information was ordered by listing all the commands of the systems, and deriving their preconditions and effects. In order to acquire a clear overview, transition diagrams were created to represent the systems' states and changes between them.

7.2.2.1 The User's Virtual Machine

The first step in creating an ETAG representation of is to determine which objects are relevant to the user, and in what way they provide locations for each other. The spatial structure of the design problem is relatively straightforward: the user has to provide data by means of a form, containing fields ("strings"), which consist of zero or more characters. Version A of the design problem does not need the CHAR object, because the user is only concerned with FIELDs (strings). The Space hierarchy is as follows:

```
FORM
    FIELD
        CHAR        ! not in version A
```

To determine the type hierarchy requires various kinds of decisions about which types can be identified, and which types are required to describe the computer system. Standard ETAG types like CONCEPT, OBJECT, EVENT, etc. are common to any computer system, and can be taken directly from the canonical basis. Types such as FIELD, CHAR and POSITION depend on the particular (type of) computer program.
The need for types at the lower levels of the hierarchy is generally more difficult to establish. Here, the main question is not which types are required, but which choice of types yields the most clear, parsimonious and psychologically realistic representation. The choice for the attribute types CONFIRMED and MODE, for instance, requires considering the consequences for (and partial creation of) the type specification, the dictionary of basic tasks, and the event specification.

Since concern is with describing existing computer systems, the bottom-up component in creating the ETAG descriptions of the design problem is especially large. Generally, the design new systems allows for a larger top-down component. The type hierarchy, which lists the concepts the user should be familiar with in order to understand the particular currency exchange program is as follows:

```
CONCEPT
        OBJECT
                FORM
                FIELD
                CHAR            ! not in version A
        PLACE
        POSITION
        EVENT
        STATE
        ATTRIBUTE
                STATUS
                CONFIRMED
                MODE            ! not in version A
                VALUE
                LENGTH
                MAX_LENGTH
```

The type specification describes the types in relation to the particular system. Most differences between the versions will be found in terms of the tasks provided for, and, as such: event types. Few, but major differences will be found in the specification of data objects CHAR and FIELD, with the more simple version A on the one hand, and the more complex versions B and C on the other. To enable easy comparison between the three versions, Figure 2 presents a generalised type specification. The type hierarchy is represented by the level of indentation.

```
type OBJECT isa CONCEPT
        value set:      FORM | FIELD | CHAR
end  OBJECT

        type FORM isa OBJECT
                theme:          FIELD
                places:         place.IN [FORM]
        end  FORM
```

```
        type FIELD isa OBJECT
                theme:          CHAR                    ! not in version A
                places:         place.IN [FIELD]
                                place.ON_TOP [FIELD]
                                place.ON_END [FIELD]
                positions:      pos.NEXT  [FIELD]       ! generally left
                                pos.PREV  [FIELD]       ! generally right
                                pos.BELOW [FIELD]
                                pos.ABOVE [FIELD]
                attributes:     STATUS
                                CONFIRMED
                                MODE                    ! not in version A
                                VALUE                   ! only in version A
                                LENGTH
                                MAXLENGTH
        end  FIELD

        type CHAR isa OBJECT                            ! not in version A
                positions:      pos.NEXT [CHAR]         ! generally left
                                pos.PREV [CHAR]         ! generally right
                attributes:     STATUS
                                VALUE
        end  CHAR

type PLACE isa CONCEPT
        object type:    FIELD | CHAR
        value set:      place.IN [OBJECT]
                        place.ON_TOP [OBJECT]
                        place_ON_END [OBJECT]
end  PLACE

type POSITION isa CONCEPT
        object type:    FIELD | CHAR
        value set:      pos.NEXT  [OBJECT] |            ! generally left
                        pos.PREV  [OBJECT] |            ! generally right
                        pos.BELOW [OBJECT] |
                        pos.ABOVE [OBJECT]
 end  POSITION

 type STATE isa CONCEPT
        object set:     FIELD | CHAR
        value set:      state.IS_IN   [OBJECT]
                        state.IS_ON   [OBJECT]
                        state.HAS_VAL [ATTRIBUTE]
 end  STATE

 type EVENT isa CONCEPT
        object set:     FIELD | CHAR
        value set:      event.SET_VAL [ATTRIBUTE]
                        event.MOVE_NEXT [OBJECT]
                        .....
                        /* the set of events to address all basic tasks */
 end  EVENT

 type ATTRIBUTE isa CONCEPT
```

```
        object type:    FIELD | CHAR
        value set:      STATUS | CONFIRMED | MODE | VALUE  | MAX_LENGTH
end  ATTRIBUTE

        type STATUS isa ATTRIBUTE
                object type:    FIELD | CHAR
                value set:      "active" | "inactive"
        end STATUS

        type CONFIRMED isa ATTRIBUTE
                object type:    FIELD
                value set:      "yes" | "no"
                comment:        "default values are regarded confirmed"
          end  CONFIRMED

        type MODE isa ATTRIBUTE                 ! not in version A
                object type:    FIELD
                value set:      "insert" | "replace"
        end  CONFIRMED

        type VALUE isa ATTRIBUTE
                object type:    FIELD | CHAR
                value set:      /* regular expression for characters */
        end  VALUE

        type LENGTH isa ATTRIBUTE
                object type:    FIELD
                value set:      /* regular expression for numerals */
        end  LENGTH

        type MAXLENGTH isa ATTRIBUTE
                object type:    FIELD
                value set:      /* regular expression for numerals */
        end  MAXLENGTH
```

Figure 2. The Type Specification of the Currency Exchange Programs.

Creating the type specification is in many ways similar to specifying the type hierarchy. Part of it is concerned with merely filling the slots of a type, whereas other parts require inspection of the dictionary of basic tasks and the event specification. Creating a type specification requires careful consideration of the consequences of decisions for the lower level parts of the ETAG description. During the analysis, the 'cursor' was erroneously identified as an object instead of the visual representation of the active character position. Repairing this mistake required extensively changing the ETAG representation.

The use of the concept of POSITION instead of the more commonly used ETAG concept of PLACE is an example of psychological parsimony: if PLACE were used, then Previous and Next would have been different attribute types of FIELD and CHAR, respectively. For users, however, a Next FIELD will be like a Next CHAR, presumably, except that they apply to a different Object.

7.2.2.2 The Basic Tasks

In ETAG, basic tasks are defined for each unique command, the program provides to the user. In this design problem, the number of basic tasks varies between 7 in version A and 15 in version C. Figure 3 presents all possible basic tasks.

```
ENTRY 1:       task:    DELETE_FIELD          ! only in version A and C
               event:   DELETE_FIELD
               object:  FIELD = f
               T1 [event: DELETE_FIELD = e]
               "make the active field (string) empty"

ENTRY 2:       task:    INSERT_CHAR
               event:   INSERT_CHAR
               object:  FIELD = f, CHAR = c
               T2 [attribute: VALUE = v]
               "insert a character value into the active field"

ENTRY 3-6:     task:    MOVE_UP, DOWN, LEFT, RIGHT
               event:   MOVE_UP, etc.
               object:  FIELD = f
               T3-6 [event: MOVE_UP = e (etc.)]
               "make the field above, below, left or right active"

ENTRY 7:       task:    CONFIRM_FIELD
               event:   CONFIRM_FIELD
               object:  FIELD = f
               T7 [event: CONFIRM_FIELD = e]
               "confirm the active field, move to next field"

ENTRY 8:       task:    REPLACE_FIELD                 ! only in version A and C
               event:   REPLACE_FIELD
               object:  FIELD = f
               T8 [attribute: VALUE = v]
               "replace the active field by a character value"

ENTRY 9:       task:    DELETE_CHAR           ! only in version B and C
               event:   DELETE_CHAR
               object:  FIELD = f, CHAR = c
               T9 [event: DELETE_CHAR = e]
               "remove the active character from the active field"

ENTRY 10:      task:    REPLACE_CHAR                  ! only in version B and C
               event:   REPLACE_CHAR
               object:  FIELD = f, CHAR = c
               T10 [attribute: VALUE = v]
               "replace the active character by another"

ENTRY 11:      task:    SET_EDIT              ! only in version B and C
               event:   SET_EDIT
               object:  FIELD = f, CHAR = c
               T11: [event: SET_EDIT = e]
               "make the active field editable and it's first CHAR active"

ENTRY 12-13:   task:    MOVE_LEFT, RIGHT      ! only in version B and C
               event:   MOVE_LEFT, MOVE_RIGHT
```

object: FIELD = f, CHAR = c
T12-13 [event: MOVE_LEFT = e (MOVE_RIGHT)]
"make the character on the left/right active"

ENTRY 14-15: task: SET_INSERT, REPLACE ! only in version B and C
event: SET_INSERT, SET_REPLACE
object: FIELD = f
T14-15 [event: SET_INSERT = e (SET_REPLACE)]
"set insert (replace) mode when in replace (insert) mode"

Figure 3. The Generalised Dictionary of Basic Tasks.

Describing the dictionary of basic tasks is relatively straightforward. It requires identifying the unique commands by means of the user's actions that invoke them, determining which concept instances (e.g. a value, etc.) are specified by the user's actions and implicitly by the system, and providing the name of the related basic event.

In the task dictionaries of the design problem there are many cases where entering a character value may be interpreted as the specification of a command, as data entry, or both. This means that the systems use modes, both explicitly set by the user, and implicitly (e.g. by default) set by the system. These modes are represented by means of flags or ATTRIBUTES. Modes do not add (much) to the structural complexity of the computer system, but they increase the memory load of users. In the event specification this is expressed by the number preconditions that test Attribute values.

7.2.2.3 The Event Type Specification

In the representation of the basic tasks there are as many basic tasks as there are events. There are, however, fewer "basic events", i.e. events that may not be decomposed into more primitive events because several events can be rewritten as a combination of other events. For example, REPLACE_FIELD can be decomposed into DELETE_FIELD and INSERT_CHAR. The event descriptions contain an indication of which field (f) and/or character (c) are active. The notation can be read as follows:

thisisa.FUNCTION (OBJECT, object specification) ATTRIBUTES

Here, the this-is-a-function does something with the attributes of type OBJECT, of which the particular instance is specified by the object specification part.

The Event Specification for Version A

In version A there is no need for the character object. The attribute VALUE of the FIELD object is used to hold a character string. The attribute CONFIRMED is used to indicate that a field is being edited, and that it is neither provided by the system (default) nor confirmed by the user.

1: type DELETE_FIELD isa EVENT
parameter: FIELD = f
user: EVENT = e
effect: event.SET_VAL (FIELD = f)
CONFIRMED = "no", LENGTH = 0, VALUE = ""

"make the active field (string) empty"

2: type INSERT_CHAR isa EVENT
- parameter: FIELD = f, CHAR = c
- user: VALUE = v
- pre_cond: state.HAS_VAL (FIELD = f) LENGTH <= MAXLENGTH
- effect: event.SET_VAL (FIELD = f)
  CONFIRMED = "no", LENGTH = LENGTH + 1, VALUE = VALUE + v

"insert (add) a character to the active field"

3-6: type MOVE_RIGHT isa EVENT ! likewise: LEFT, UP, DOWN
- parameter: FIELD = f
- user: Event = e
- pre_cond: state.IS_IN (FIELD = f, pos.NEXT) FORM
  state.HAS_VAL (FIELD = f) CONFIRMED = "yes"
- effect: event.SET_VAL (FIELD = f) STATUS = "inactive"
  event.SET_VAL (FIELD, pos.NEXT) STATUS = "active"

"if there is one, make the next field active"

Note that moving to another field is in our solution only possible when the active field is confirmed, which is expressed as a precondition. The formulation of the design problem is not conclusive about whether this is the case or not.

7: type CONFIRM_FIELD isa EVENT
- parameter: FIELD = f
- user: EVENT = e
- effect: event.SET_VAL (FIELD = f) CONFIRMED = "yes"
  event.MOVE_RIGHT

"set the active field confirmed, move to the next one"

The next event in the event specification of version A concerns replacing the contents of a field by a character. This task or event is only possible when the field is confirmed by the user or contains the default value. Note that this is only an extra task because of the precondition; the body or the effect of the event could have been rewritten as a combination of deleting a field and inserting a character.

8: type REPLACE_FIELD isa EVENT
- parameter: FIELD = f
- user: VALUE = v
- pre_cond: state.HAS_VAL (FIELD = f) CONFIRMED = "yes"
- effect: event.SET_VAL (FIELD = f)
  CONFIRMED = "no", LENGTH = 1, VALUE = v

"replace the active field (string) by a character"

The Event Specification for Version B

In this version the CHAR object type is introduced to address the possibility to move within a field and to insert, replace or delete single characters. A flag (attribute) MODE is required to indicate the difference between insertion and replacement of characters. To facilitate comparing between versions, the numbering of events from version A is adapted.

2: type INSERT_CHAR isa EVENT
- parameter: FIELD = f, CHAR = c

```
        user:           VALUE = v
        pre_cond:       state.HAS_VAL (FIELD = f)
                        CONFIRMED = "no", MODE = "insert", LENGTH <= MAXLENGTH
        effect:         for (CHAR, place.ON_END) downto (CHAR = c)
                                event.SET_VAL (CHAR) VALUE = (CHAR, pos.PREV) VALUE
                        event.SET_VAL (CHAR = c)        VALUE = v
                        event.SET_VAL (FIELD = f) LENGTH = LENGTH + 1
                        event.MOVE_RIGHT
        "insert a character c into the active field"

3-6: type MOVE_RIGHT isa EVENT                  ! likewise: LEFT, UP, DOWN
        parameter:      FIELD = f, CHAR = c
        user:           Event = e
        pre_cond:       state.IS_IN (FIELD = f, pos.NEXT) FORM
                        state.HAS_VAL (FIELD = f) CONFIRMED = "yes"
        effect:         event.SET_VAL (FIELD = f) STATUS = "inactive"
                        event.SET_VAL (CHAR = c)  STATUS = "inactive"
                        event.SET_VAL (FIELD, pos.NEXT)
                        STATUS = "active", CONFIRMED = "yes"
        "if there is one, make the next field active"

7: type CONFIRM_FIELD isa EVENT
        parameter:      FIELD = f
        user:           EVENT = e
        effect:         event.SET_VAL (FIELD = f) CONFIRMED = "yes"
                        event.MOVE_RIGHT
        "set the active field confirmed, move to the next one"

9: type DELETE_CHAR isa EVENT
        parameter:      FIELD = f, CHAR = c
        user:           EVENT = e
        pre_cond:       state.HAS_VAL (FIELD = f) CONFIRMED = "no"
        effect:         for (CHAR = c) upto (CHAR, place.ON_END[FIELD = f])
                                event.SET_VAL (CHAR) VALUE = (CHAR, pos.NEXT) VALUE
                        event.SET_VAL (CHAR, place.ON_END) VALUE = ""
                        event.SET_VAL (FIELD = f) LENGTH = LENGTH - 1
        "remove the active character from the active field"

10: type REPLACE_CHAR isa EVENT
        parameter:      FIELD = f, CHAR = c
        user:           VALUE = v
        pre_cond:       state.HAS_VAL (FIELD = f) MODE = "replace", CONFIRMED = "no"
        effect:         event.SET_VAL (CHAR = c) VALUE = v
                        event.MOVE_RIGHT
        "replace the active character by another"

11: type SET_EDIT isa EVENT
        parameter:      FIELD = f
        user:           EVENT = e
        effect:         event.SET_VAL (FIELD = f) MODE = "replace", CONFIRMED = "no"
                        event.SET_VAL (CHAR, place.ON_TOP[FIELD = f]) STATUS = "active"
        "make the active field editable and the 1st char active"

12-13: type MOVE_LEFT isa EVENT                 ! likewise: MOVE_RIGHT
        parameter:      FIELD = f, CHAR = c
        user:           EVENT = e
```

```
        pre_cond:       state.HAS_VAL (FIELD = f) CONFIRMED = "no"
                        state.IS_IN (CHAR = c, pos.PREV) FIELD
        effect:         event.SET_VAL (CHAR = c) STATUS = "inactive"
                        event.SET_VAL (CHAR, pos.PREV) STATUS = "active"
        "make the character on the right active"

14-15: type SET_INSERT isa EVENT                ! likewise: SET_REPLACE
        parameter:      FIELD = f
        user:           EVENT = e
        pre_cond :      state.HAS_VAL (FIELD = f) MODE = "replace"
        effect:         event.SET_VAL (FIELD = f) MODE = "insert"
```

The most remarkable thing about this representation is the fact that some events (for instance the event of inserting a character) are specified at a very low level of detail, almost in program code. Previous applications of ETAG mostly addressed systems where the "basic unit" of interaction (e.g. a message in an electronic mail system) is much more abstract and larger than the manipulation of single characters in a spreadsheet.

The Event Specification for Version C

Version C is a combination of version A and B. In comparison to version B there is one extra task to delete all the characters of a field. This event cannot be directly taken from the event specification of the A version but need to be rewritten in terms of single character manipulation.

```
1: type DELETE_FIELD isa EVENT
        parameter:      FIELD = f
        user:           EVENT = e
        effect:         event.SET_VAL (CHAR, place.ON_TOP[FIELD = f]) STATUS = "active"
                        while not (state.HAS_VAL (CHAR = c) VALUE = "")
                                event.DELETE_CHAR
                        event.SET_VAL (FIELD = f) CONFIRMED = "no"
        "make the active field (string) empty"
```

Creating the event specification is not a particularly difficult task although, it is rather time-consuming. To complete the type hierarchy and type specification for an existing system, the choice for (not) using certain types, and especially attribute types, depends on details of the event specification. Because the opposite also holds, a number of top-down and bottom-up trial and error iterations are required to arrive at a satisfactory solution.

The reason why the event specification in this case took some time is also due to the fact that ETAG is not meant to describe low level details such as single character manipulation. To compare the different versions of the currency exchange program it was necessary, however, to create very detailed ETAG models. As a result, the event specification looks a bit like a computer program that hides, rather than explicates what is semantically most important to the user. Trying to arrive at a representation that was as clear as possible took careful deliberation.

7.2.2.4 The Production Rules

The specification level is concerned with the order of specifying conceptual command elements. Apart from the presence or absence of particular task entries, it is similar in form for each version:

T1:	[event: DELETE_FIELD]	::=	specify [event: e]
T2:	[event: INSERT_CHAR][attribute: VALUE]	::=	specify [value: v]
T3-6:	[event: MOVE_UP, etc.]	::=	specify [event: e]
T7:	[event: CONFIRM_FIELD]	::=	specify [event: e]
T8:	[event: REPLACE_FIELD][attribute: VALUE]	::=	specify [value: v]
T9:	[event: DELETE_CHAR]	::=	specify [event: e]
T10:	[event: REPLACE_CHAR][attribute: VALUE]	::=	specify [value: v]
T11:	[event: SET_EDIT]	::=	specify [event: e]
T12-13:	[event: MOVE_LEFT, MOVE_RIGHT]	::=	specify [event: e]
T14-15:	[event: SET_INSERT, SET_REPLACE]	::=	specify [event: e]

The reference level specifies the way conceptual entities of the system are referred to in the interaction language (e.g. pointing vs. naming). In the systems of the design problem, each event and each value is specified by a named lexical item. Consequently, there is only one reference-level rule:

specify [identifier: i] ::= name 'symbol [identifier: i]'

In a sense, one may regard entering a character value, and perhaps indicating the direction of movement as selection operations rather than naming operations. Here, it was decided not to do so, because it concerns selecting on the input device -the keyboard-, and not on the visual display.

The lexical level is meant to represent the semantic expressiveness of the names of command elements. For this reason, the reference level utilises quoted string (e.g. 'symbol [SOMETHING]') to indicate which part of the right hand side of a reference level rule should be replaced by the right hand side of a lexical level rule.
In the list of lexical rules the names or values of the function keys are enclosed in quotes, to indicate that they have to be learned and retrieved from memory. No such learning is required to generate character values.

symbol [value: VALUE]	::=	[%char_value%: v]
symbol [event: DELETE_FIELD]	::=	[%commandkey%: "F7"]
symbol [event: MOVE_UP]	::=	[%commandkey%: "_"]
symbol [event: MOVE_DOWN]	::=	[%commandkey%: "_"]
symbol [event: MOVE_LEFT]	::=	[%commandkey%: "_"]
symbol [event: MOVE_RIGHT]	::=	[%commandkey%: "_"]
symbol [event: CONFIRM_FIELD]	::=	[%commandkey%: "ret"]
symbol [event: DELETE_CHAR]	::=	[%commandkey%: "del"]
symbol [event: SET_EDIT]	::=	[%commandkey%: "F8"]
symbol [event: SET_INSERT, REPLACE]	::=	[%commandkey%: "ins"]

Similar to entering character values, using the arrow keys to express the target position of an intended movement may also be regarded as not involving memory retrieval. In that case, the four rules on movement could be replaced by one rule. This would also simplify the UVM, basic tasks, and type specification. The "ins" command is different in that it acts as a toggle between insert and overwrite mode. Toggle commands like Caps Lock save keystrokes, and the "ins" command may also save some memory for key assignments, but at the higher-level cost of having to remember or perceive the current mode.

The keystroke names represented in the list do not correspond to the particular commands for all versions of the computer system, but the idea will be clear. In all versions of the computer system the names of the keystrokes correspond to the keystrokes themselves. Therefore, there is only one rule at the keystroke-level:

```
name [%identifier: %char_value% | %commandkey%] ::= KEY_IN [%identifier%]
```

7.2.2.5 General Comments on Modelling the Currency System

Regarding the effort required to create the ETAG representation, the dictionary of basic tasks could be specified with little effort. The type specification required many iterations, especially where it concerned events. Because only the physical actions were specified, it was often not clear what exactly were the pre and post conditions and the effects of events. Consulting the author of the original specifications of the design problem brought some help, but no complete success.

In creating the representation, a mistake of the analysts required a complete rewriting of the type specification (see below). However, once the correct type specification was complete, filling in the missing details in the spatial and type hierarchies, and specifying the production rules was straightforward.

The primary target of the modelling contest was to determine for which purposes each of the participating models could be useful, and to what extent. Elsewhere (chapter 2; de Haan et al., 1991), we have identified analysis, evaluation, prediction and description as the main purposes of formal modelling techniques. ETAG was originally proposed as a model to describe computer systems for design purposes (Tauber, 1988), and in this respect we may conclude that ETAG is indeed a useful tool for describing the design problem of the modelling contest rather precisely and complete.

Notwithstanding the success of applying ETAG as a description method, it must be stated that, presently, ETAG is far from perfect. There are gaps in the descriptive power of ETAG, it is not easy to create and manipulate (e.g. change) ETAG descriptions, and it is unclear how ETAG can be used to evaluate usability aspects of design proposals.

Regarding how the ETAG representations can be used to measure the complexity of systems, the following points were noted. In the specification of the computer systems much attention was paid to providing the user with visual information about what the active cell and character are, and about which cells have been confirmed. In ETAG it is not possible to address such visual support of the user's tasks. Versions B and C of the currency exchange system provided more commands than version A, but the form of the commands (usually single keystrokes) is similar. Consequently, the number of rewrite lines per level of the production rules is somewhat larger in versions B and C, but the production rules are not more complex.

There is, however, a large difference in the length and complexity of the type specifications, and especially of the event descriptions, between version A, and versions B and C. This clearly exemplifies that the complexity of user interfaces depends only partly on the complexity of the interaction language of computer systems. In this case, versions B and C

provide for single character editing, in addition to moving between, entering and deleting fields in version A.
Apart from differences in functionality, versions B and C are also more complex, because they require the user to keep track of the specific editing modes, such as field versus character editing, and insert versus replace mode. In figure 2, for each of the tasks, the number between brackets indicates the number of additional preconditions that apply to the task in the more advanced versions of the program. Predicting the complexity of a device, as opposed to the interaction language, is only possible when a model can address aspects of the user's "how it works" knowledge and task semantics, in addition to the knowledge on "how to do it".

In a study by Attema and van der Veer (1992), ETAG's predecessor TAG is applied to the same currency exchange problem as described above. TAG is interesting in this context, because it does not address "how it works" knowledge, but does use semantic features to distinguish between tasks, and a feature grammar to describe the derivation of the commands to perform the tasks. From the TAG analysis it can be concluded that TAG is not able to address the differences in device complexity. However, by its use of semantic features, TAG was able to predict complexity differences, caused by the mode-dependent interpretation of the user's actions. A second difference between ETAG and TAG is that creating TAG models is faster and easier than creating ETAG representations. Given roughly the same amount of time and effort available to analyse the problem, TAG turned out to even provide the opportunity to make suggestions to significantly improve the interaction languages.

An important aspect of the money exchange system of the modelling contest is hardly ever mentioned in this study, because ETAG is unable to address it. Like most other models, ETAG is not able to describe the "presentation component" of computer systems: the way in which the system supports the user's task performance by visually presenting information about the history, the current state and future possibilities of the problem space. In general, there is an overwhelming lack of theoretical work around this question. Only a few propositions exist which attempt to describe, for instance, how highlighting of fields may support the user's task performance (Polson and Lewis, 1990; Payne, Squibb and Howes, 1990). It may be possible to extent ETAG by describing how events change the visual state of systems, but this would increase the already problematic extent of ETAG descriptions even further. As a more promising solution, we are currently considering to use software tools, such as user interface management and design systems to complement the ETAG approach.

A problem which may look like a shortcoming of ETAG, but is in fact a shortcoming in the design specification, had to do with the level of detail of the event specification. To describe the event of inserting a character into a field, for instance, requires describing that the characters originally in the field will shift one place to the right. For the sake of describing the design proposals as completely as possible, a control construct like a for-loop was used to address this implementation level detail. As such, we consider this foremost as an example of what happens when a model is used for the wrong purpose, or, stated differently, what happens with a description of the conceptual knowledge of users when the input, the design, is specified in terms of implementation solutions.

When analysing the design problem, the cursor was first erroneously identified as an object of the UVM, instead of regarding it as the visual representation of the active character position. When this mistake showed up at a late stage of the event description, part of the UVM and most of the event specification had to be rewritten, requiring considerable effort. This mistake points at two related problems regarding ETAG. First, because ETAG descriptions are relatively complete, they tend to be large, and because they are large, they become complex and difficult to manipulate.
Secondly, creating an ETAG representation sometimes requires careful consideration, because the method leaves the designer quite some freedom of choice and uncertainty. There is no easy remedy to these problems. At least partially, we expect to find some relief in the use of tools to automate and/or facilitate parts of the ETAG analysis, like intelligent editors, consistency checkers, etc. We also intend to strengthen the methodology and narrow down the uncertainty in applying ETAG by providing, among other, guidelines for identifying objects, etc., and a "how to do an ETAG analysis" manual.

At the start of applying ETAG to the design problem, three expectations were formulated about the possibility to evaluate systems providing different functionality, the low level of the specification which seemed to stress the least interesting parts of ETAG, and the absence of full specifications of the behaviour of the systems, respectively.
In the absence of usability criteria for ETAG representations, it remains unclear what ETAG can do regarding the evaluation of the systems. At face value, it may be clear from the representations that complexity increases from version A to B, and C, although the latter version is still quite simple.

Secondly, the expectation was formulated that the low level of the specification of the design problem would hide the most interesting parts of ETAG. It was indeed the case that the dictionary of basic tasks and the production rules merely listed what was specified in a straightforward manner. However, it was also shown that adding only a few extra keystrokes or commands can lead to major changes in the required "how it works" knowledge of the user. The description of the events became much more complicated when switching from the first to the second and third versions of the computer system. Similar to Payne et al. (1990) it may be concluded that details of the user interface may have important consequences for the complexity of learning to use the computer system.
As a third point, it was mentioned that the specification of the systems is incomplete. During the ETAG analysis, several omissions in the problem specification became clear, which could not all be resolved by consulting the author of the specification. It would be interesting to see whether all participants did indeed describe the same computer system. This matter points at an important aspect of design; namely that it is difficult to create a full specification of a computer system in natural language (Meyer, 1985). Natural language specifications easily lead to problems being formulated in terms of partial solutions, rather than in complete but more abstract terms. In our view, computer systems design should not start with the way in which the status of fields is visually represented. Instead, computer systems design should start with the semantics of the problem domain the user has to cope with, and in this respect we expect that the use of formal models like ETAG will prove to be most successful.

7.2.3 Analysis of Two Electronic Mail Systems

A study by van der Meer (1992) involved ETAG analyses of two electronic mail systems of the Dutch PTT Telecom, memostart and memocom. The systems were meant to provide personal computer owners the possibility to exchange messages by means of a standard telephone line, a modem, and an account on a Dutch PTT Telecom central computer system. The systems of the study were prototypes of commercial products. Both systems provide electronic mail facilities to personal computer owners, by means of a modem, a standard telephone line, and a central computer which acts as a mail server.

A manual was provided for only one of the systems, and this was used to get a first impression of its possibilities. The next step was to work with the system in order to explore its workings further. The provision of an on-line help system facilitated this stage. The second system was received without a manual, and it did not provide such extensive help facilities as the first one. However, based on the experience with the first system and trial and error, the full functionality of this system could be determined. The results of the orientation phase consisted of a list of important differences between the systems, and of flow charts showing the command structure of each system in terms of available menus and menu options.

The second stage consisted of creating the ETAG type specifications. For this purpose, a preliminary listing was made of all the objects and their attributes which were present in the interfaces. Subsequently, for each of the objects and attributes, the relations to other objects and attributes were examined in detail, and a relatively complete type hierarchy was created. During this stage, several duplicate objects and attributes could be discarded.

In the next step, a closer look was taken at the tasks the system supported. This consisted of describing the available tasks in combination with the user and system supplied arguments and the associated events. As an event may be associated with more than one task, in a further refinement events that were almost similar were taken together, and details were added or changed. During this stage, several mutual adjustments were required between the dictionary of basic tasks and the type description. In refining the descriptions, ETAG showed to be particularly valuable by enforcing completeness and by focussing attention on the correctness of the results.

As a final step, the production rules and the detailed contents of each event should be described. This was only partly done, because it was already clear that problems with these systems did not come from interaction language complexities or from problems understanding the detailed workings of tasks and events.

Regarding ETAG as a formal description tool, the conclusion could be drawn that using it enforces, or at least stimulates, the analyst to generate complete and correct descriptions of the system.
In this study, the problem that ETAG presently can not address the presentation component of user interfaces also showed up. In the study of the money exchange system this was not important, because each of the systems presented the same visual information. In this study, however, there is a difference in both the amount and the form of visual task support.

In the representation of the memocom system, it was not always possible to represent almost similar tasks and events similarly because of differences in context. For example, in one environment the user is able to invoke a set of (sub)tasks by means of one menu choice. In another environment, the same menu choice exists, except that here the task-set might have some extra or fewer tasks. In order to deal with this, a distinction was made between menu tasks as higher level tasks and embedded tasks as lower level tasks. Note that an embedded task may also be a composite -non basic- task. Figure 3 presents part of the task-hierarchy of memocom. This figure shows that the two Menu Tasks, EDIT-NEW-MESSAGE and EDIT-MESSAGE-AND-MAKE-READY-TO-SEND are almost, but not exactly, identical, because each is composed of a slightly different set of Embedded Tasks. The difficulty of dealing with almost similar tasks is not necessarily a deficiency of ETAG, as it may also say something about the user complexity of memocom.
Another problem regarding the use of ETAG was that it is sometimes difficult to represent the opportunity for users to interrupt activities. This problem is not unique to ETAG, but has been noticed to apply to various other formal modelling techniques as well. In figure 3, the user may interrupt the ongoing task after each "task" or "event" at a lower level has been performed.

```
18: EDIT-NEW-MESSAGE is a MENU-TASK
        user-arguments:
        effect:   event.CREATE-MESSAGE (send-tray, message: o),
                  task.EDIT-MESSAGE (send-tray, message: o),
                  task.DETERMINE-MESSAGE-ATTRIBUTES (message : o)
        "a new message in the send-tray is supplied with text and send-attibutes"

21: EDIT-MESSAGE-AND-MAKE-READY-TO-SEND is a MENU-TASK
        user-arguments:  tray:  o1,  message:  o2
        effect:   event.COPY-MESSAGE-TO-TRAY (tray: o1, message: o2, send-tray),
                  task.EDIT-MESSAGE (send-tray, message: o3),
                  task.DETERMINE-MESSAGE-ATTRIBUTES (message : o3)
        "the message is copied into the send-tray as a new message, where it may
          be changed and provide with new send-attibutes"

37: DETERMINE-MESSAGE-ATTRIBUTES is a EMBEDDED-TASK
        user-arguments:   message: o
        effect:   task.DETERMINE-SUBJECT (message : o),
                  task.DETERMINE-SENDOPTIONS (message: o),
                  task.DETERMINE-ADDRESS (message : o)
        "set or change the attributes for sending a message"

40: DETERMINE-ADDRESS is a EMBEDDED-TASK
        user-arguments: message: o, <address>string : v
        effect:   event.ASSIGN-MESSAGE-ATTRIBUTE (message : o), <address>string : v)
        "set or change the address the message is to be send to"
```

Figure 3. A Hierarchy of Menu- and Basic Tasks in the Memocom System.

Regarding the learnability and usability of the two systems, the complexity of both the ETAG representations and the command structure flowcharts provided evidence for the intuitive conclusion that, in comparison to the memostart system, memocom is more complex. Memocom provides additional tasks, like storing messages in different archives, and it

provides more complex tasks, such as storing and archiving sets of messages in one turn instead of one by one.

7.3 An Evaluation of ETAG as a Tool for Analysis

7.3.1 What Has Been Learned From These Studies

ETAG is able to capture essential differences among computer systems. Regarding the studies reviewed above, it may be concluded that ETAG is able to represent different types of computer applications and to highlight important differences between variants of similar systems. Several problems came forward when actually creating ETAG models. Creating ETAG representations is difficult. It may be difficult to choose the most appropriate elements (e.g. objects, attributes) to create ETAG representations.

ETAG Representations tend to be large and thereby complex and resisting change. When less appropriate elements have been chosen, changing the representations requires much work. To facilitate creating ETAG representations, Yap (1990) has suggested a number of guidelines for both how to approach the problem in general, and how to choose specific objects, tasks, etc. Most of these guidelines proved to be very useful. Nevertheless, one of his conclusions states that they are not enough.
More directives for creating ETAG representations are needed. There is no guaranteed algorithm for performing an ETAG analysis. Although ETAG generally provokes a top-down hierarchy of design decisions, the process always requires iterative steps.

ETAG does not adequately address the presentation interface. ETAG is unable to satisfactorily address the problem of the presentation interface. The model lacks an adequate structure and adequate concepts in this respect. In the study of the currency exchange system this was no problem because the presentation interface was similar in each version of the system. In the study of the electronic mail systems it did matter, because both the amount and the content of visual task support differed.
At a cost, ETAG may be extended to address object presentation. It was shown that ETAG representations can be extended with screen objects, for instance in modelling page tools.

Tools are needed to complement ETAG and facilitate its use. Because ETAG representation already tend to be large, the use of tools may be a better approach to solve incompleteness. Elsewhere (chapter 2; de Haan, van der Veer and van Vliet, 1991) we proposed tools for e.g. consistency checking and intelligent editing to deal with the size problem. Likewise, it may be possible to employ tools like user interface management systems and interface builders, instead of extending the model.

More advanced control structures are needed for the event descriptions. A problem referring to the notation of ETAG is the difficulty of describing asynchronous, (almost) identical user actions, and control structures in event descriptions. Describing asynchronous tasks, such as the opportunity to interrupt ongoing activities, requires additional information to be captured

in the representations, making them much more complex. It is not clear how this problem may be solved in ETAG or, more generally, in any of the present models for analysis.
To describe how similar user tasks may occur in different contexts, van der Meer (1992) proposed menu tasks as higher level tasks to deal with context differences. Basically, this problem is similar to the problem of the event descriptions, as the place where pre- and postconditions are handled. This problem also showed up in the analysis of the money exchange systems. In order to resolve it, a powerful yet uniform notation for the description of events needs to be developed.

ETAG is widely applicable and yields relatively complete representations. All studies reviewed above were primarily directed at establishing the weak and strong points of ETAG as a tool for describing computer systems. In this respect, we think that ETAG is quite strong. Although the descriptive power of ETAG was not directly compared to that of other formal models, we consider it justified to conclude that it yields more complete representations and that it is applicable to a wider variety of computer systems.
Both, "how to do it" and "how it works" knowledge are captured by ETAG. In comparison to formal models which are restricted to the user's knowledge of command element specification, ETAG is also able to address the (device) complexity in the "how it works" knowledge.

ETAG may be applied to evaluate usability and learnability. It was also shown that ETAG has potential for use as an instrument to evaluate aspects of the usability and learnability of user interfaces.
There is a need to validate ETAG for evaluation purposes. Unfortunately, which features of ETAG representations can be used for evaluation purposes remain unclear and need further investigation.

7.3.2 Conclusions

In the introduction of this chapter, it was stated that formal methods should be evaluated. Without proper evaluation studies and clear insights into their applicability formal methods will remain mere notations. Up to now evaluation studies have been scarce, and often restricted to evaluating specific features of models. In this chapter ETAG was illustrated as a method to analyse user interfaces, and aspects of its applicability were evaluated by applying it to analyse different computer systems and different versions of the same system. In our view it is more important to determine in advance how well a formal design method can be used to describe a diversity of applications, than to extensively evaluate a few predictions to "prove" its validity. In doing so, we have identified several points about ETAG which need to be improved, such as coverage of the presentation interface, directions on applying the method, the size and complexity of the representations and the derivation of usability predictions.

This chapter is based on: de Haan, G. and Van der Veer, G.C. (1992a). Analyzing User Interfaces: ETAG validation studies. The spreadsheet study in section 7.2.2 is based on: de Haan, G. and Muradin, N. (1992). A Case Study on Applying Extended Task-Action Grammar (ETAG) to the Design of a Human-Computer Interface.

Chapter 8:

Etag as the Basis for Intelligent Help Systems

If anyone should ever hit on the idea of testing the vocabulary most parents use when talking to their children, he would find that it makes the vocabulary of the comic strip look like a complete dictionary.
Böll, H. (1965).

Abstract
An ETAG representation is a conceptual model which contains all information a user might want to have of a computer system. As such, ETAG representations may serve as the basis for intelligent help facilities. A number of studies will be reviewed in which ETAG was used to build non-interactive and interactive facilities providing help information about computer systems. In order to provide static -non changing- information about the computer system, the ETAG representations contain sufficient information. To provide dynamic information about interacting with the system as well, the information in ETAG representations needs to be supplemented with history information. In order to cope with dynamically changing information, it proved to be necessary and more widely advantageous to use an intermediate (PROLOG) representation of ETAG and history information.

8.1 Introduction

Although ETAG is originally developed for the purpose of design specification (Tauber; 1988, 1990), its ability to represent much of the relevant computer system knowledge on behalf of the user should make it more widely applicable. Formal models can be used for task-analysis, user knowledge analysis, user performance prediction and specification for design (de Haan, van der Veer, and van Vliet; 1991). ETAG can be applied to specify and analyse the knowledge of users of various computer systems, which is required, among other, to predict the places where performance difficulties might arise (de Haan and van der Veer, 1992).

The basic principle that ETAG represents all the knowledge of a system that a user should and might have in memory makes ETAG a gratified candidate for use in an even more applied role: the development of intelligent help systems. An ETAG model consists of all system knowledge a user might need, and with the addition of a means to interrogate the model it should be possible to build systems to supply the user with this knowledge. Depending on the extent to which such a help system can adapt to the circumstances in which help is requested, it may behave more or less intelligently. In this chapter systems will be discussed which use ETAG to provide help information interactively or non-interactively, and to provide static help information about the system and dynamic help information about previous interaction with the system.

The main purpose of this chapter is to investigate to what extent ETAG, as a representation of the knowledge that users should have about a user interface, can actually be used for the purpose of generating user help information. There is ample evidence that models, and especially conceptual or design models, can be used for this purpose (Foley and Sukaviriya,

1994; Forbrig and Schlungbaum, 1999). The purpose of this chapter is not to determine what is the best approach but to determine how well ETAG fares in generating user help information.
Section 8.2 discusses why to use a formal model like ETAG as the basis for help systems and describes ETAG based help systems in general terms. Subsequently, in section 8.3 several ETAG based systems will be reviewed which provide the user with either static or dynamic help information. Finally, in section 8.4 and 8.5 the conclusions on ETAG based intelligent help systems will be summarised and requirements for future research will be formulated.

8.2 Etag as the Basis for Help Systems

In designing a help system for a computer system there are three main points of concern: (a) ensure that the user is supplied with the help that is needed in the particular circumstances; (b) ensure that the help conforms to the characteristics and the behaviour of the system, and, (c) ensure that there are no internal inconsistencies in the help information.

The question of how to ensure that the information provided is appropriate can not really be answered by ETAG. We have restricted ourselves to user-initiated help requests. What needs to be done is to define the set of possible or 'allowable' user questions and determine whether the answers to them are appropriate. The formality of ETAG will facilitate to do this, at least in comparison to using a non-formal system representation.

Earlier, it was argued that a formal model of the knowledge of a competent user should be used as the basis for the design of computer systems. There are good reasons to use the same formal model for designing help information for such systems: when the system and the help facilities share the same source, their mutual integrity is guaranteed: the help will automatically keep up with (changes in) the system. A uniform formal basis for the help information will also ensure the internal integrity of help information. ETAG can in principle be used as the basis for help, and especially in comparison to using non-formal representations it will be advantageous to do so.

8.2.1 The General Structure of an ETAG Help System

A general structure of ETAG based help systems is developed by van der Veer (1990). The main ideas behind this structure are to insure that the help system is independent of the target system and follows a modular structure. Target system independence is meant to insure a wide applicability of help systems, including usage for research and prototyping purposes. Apart from usual considerations, modularity is important for practical reasons (the extent of a student project) and to enable construction of special purpose systems for, e.g. user help, tutoring, etc. using ready-made building blocks. The role of an (e.g. ETAG-based) help system within a general architecture for modular user interfaces is presented in Figure 1.

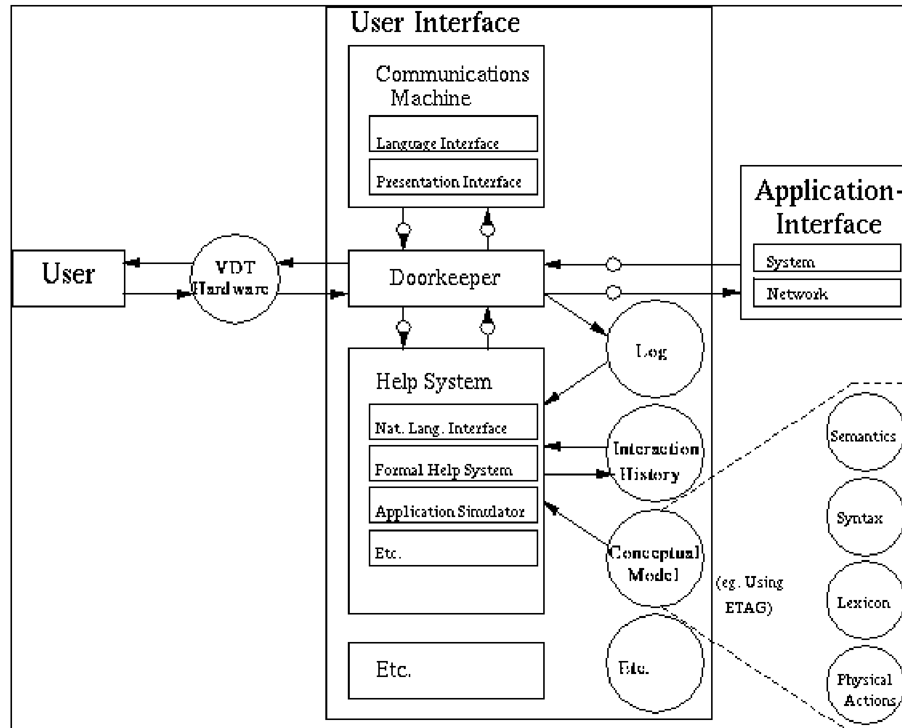

Figure 1. The Help System as part of a General UI Architecture (van der Veer, 1990).

Part of any (static or dynamic) ETAG based help system is a Natural Language Interface, to translate between the user's natural language and the formal language used internally by the help system. Internally (not shown in figure 1), the Natural Language Interface consists of a natural language back end, which translates formal information into more or less natural English, and a natural language front end, which is used to formalise the questions of users that it receives from the doorkeeper machine. The Doorkeeper distinguishes help requests from commands directed to the system. The Formal Help System (or: the Formal Question Answering Machine) is the module which actually retrieves the requested information. A module to translate an ETAG representation into an internal conceptual model that suitable format for question answering is called the ETAG Translator.

In help systems, static information is the information about a computer system which does not change by interacting with it, such as knowing what the concept file means. On the contrary, dynamic information may change during interaction, such as knowing if a particular file is still there. To provide dynamic information several modules are required, in addition to those of a static help system. In ETAG-based help systems, independence of the target system is reached by simulating the target system in terms of ETAG's basic tasks in an ETAG Simulator module.

During a session, the actions of a user are collected in a User Log file. This file is parsed by a Log Parser module to determine which basic task and task arguments the user has -correctly- specified. Basic tasks change the state of the system, which is kept by the System State Manager. The System State Manager communicates with the Database Manager to store and

retrieve facts about the interaction history in the History Database. The real or target system could be another part of the history system. However, to ensure the application independence of the help system, it was decided to simulate the computer system in terms of ETAG's Basic Tasks.

The general structure just described may be extended with additional modules to make it suitable for different purposes. For example, to build intelligent teaching systems, modules might be added, like user monitors, process inference modules and coaching machines (see: van der Veer, 1990 for further details).

8.3 Examples of ETAG-based Help Systems

In this section we will report about research efforts involving the use of ETAG representations as the basis for systems to provide users with various kinds of help. The information about computer systems was either generated solemnly on the basis of ETAG, or generated using ETAG in combination with additional sources. Information was provided by means of non-interactive on-line manuals or by means of interactive help systems. In the latter case, the system could either provide static information about the system, or dynamic information about interacting with the system.

8.3.1 Help Systems to Provide On-line Manuals

A first project on ETAG based help systems, reported in Broos (1989), involved creating a system to automatically generate natural language help text on the basis of a description of the UVM of a computer system. The system is a non-interactive Natural Language Back End. A Natural Language Front End was outside the scope of the project. Instead, a limited set of questions is assumed to be specified by means of, for instance, a simple menu interface listing all possible questions. Task level help is not provided, but semantic ("what is ... ?"), syntactic ("how do I ... a task ?"), and keystroke level ("how do I ... an action ?") is. Figure 2 graphically presents the structure of the translator module.

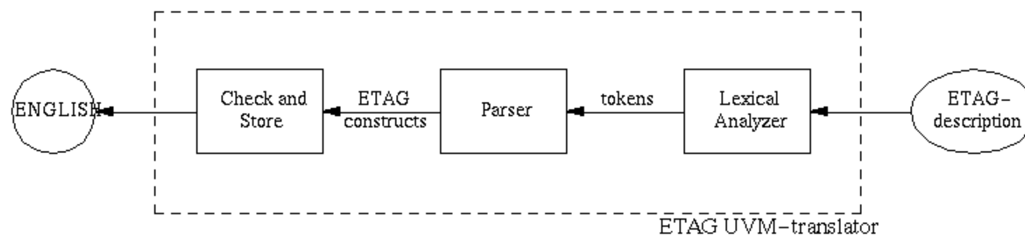

Figure 2. The Structure of the ETAG UVM Translator.

The following example presents a clipboard as it would appear in the type specification part of an ETAG representation of a typical Macintosh application.

```
type  [OBJECT  =  CLIPBOARD]
        supertype:          [SINGLE_OBJECT_BOARD] ;
        themes:             [STRING: *s] | [WORD: *w] | [PICTURE: *p] | [RULER: *r] ;
        instances:          [CLIPBOARD: #clipboard] ;
end
```

In essence, this specification indicates that there is one clipboard, called "clipboard", which may contain either a string, a word, etc. The help system would translate this specification into something readable like the following:

Here is some information about the type CLIPBOARD
It is a special kind of SINGLE OBJECT BOARD
It can contain a STRING, a WORD, a RULER, or a PICTURE
There is a CLIPBOARD, indicated with "clipboard"

The help system is implemented using the lexical analyser Lex (Lesk, 1975), and the specification language for user interface management systems SYNICS (Guest, 1982). Semantic, or rather, strategic information about task procedures is not supported. Also, the natural language output of the system is readable but not very 'natural'. This project showed that ETAG-based help systems are feasible. Furthermore, the help provided may easily be extended or changed by using different ETAG representations.

A practical application of creating an on-line manual involved the design and implementation of a help function for an experimental electronic mail system, based on an ETAG description of it (Tamboer, 1991). The electronic mail system was developed as part of a European research project of the COST-11-ter working group (van der Veer et al., 1988). Due to some radical changes of the ETAG notation, it was no longer possible to automatically translate ETAG information into help messages so it had to be done by hand. Nevertheless, this project showed that it is possible to use help messages created on the basis of an ETAG description to build a help function as an integral part of a computer application.

8.3.2 Help Systems to Provide Static Information

The goal of this project was to investigate possibilities to create an interactive help machine, as a step further than facilities to translate information, formally represented in ETAG, into a kind of natural language. Part of this project involved building a help system around a formal help machine (Fokke, 1990). An automatic ETAG Translator transforms a given ETAG description into a Prolog database of clauses or 'facts' representing the static information contained in the ETAG model. This part involved choosing an appropriate representation format for the database. As an example, in some electronic mail systems there is a task "mark for deletion" to discard one or more messages upon leaving the mail environment. In the Dictionary of Basic Tasks of an e-mail system it can be described as shown below. The ETAG description tells that this task is identified by the arbitrary number 9, and that it invokes an associated basic event with the same name. Upon execution of the task, the system will supply the name of the so-called message file, whereas the user should specify the name of the task and the messages to be marked for deletion:

```
ENTRY 9:
        [MARK_FOR_DELETION isa TASK],
        [MARK_FOR_DELETION isa EVENT],
        [MESSAGE_FILE: *y],
T9 [MARK_FOR_DELETION isa EVENT] [MESSAGE isa OBJECT: {*}]
```

The output of the Translator is a number of declarative Prolog clauses:

```
task( 9, mark_for_deletion,
        [concept( isa( mark_for_deletion, event ), generic )],
        [concept( message_file, generic( y ) )],
          t9,
        [concept( isa( mark_for_deletion, event ), generic ),
          concept( isa( message, object ), set( [generic], [range( 0, inf )] ) )
        ],
).
```

Using a specially designed formal query language, an Interpreter may consult the database to generate a formal answer to a formally specified question. Figure 3 presents the architecture of such a formal help machine.

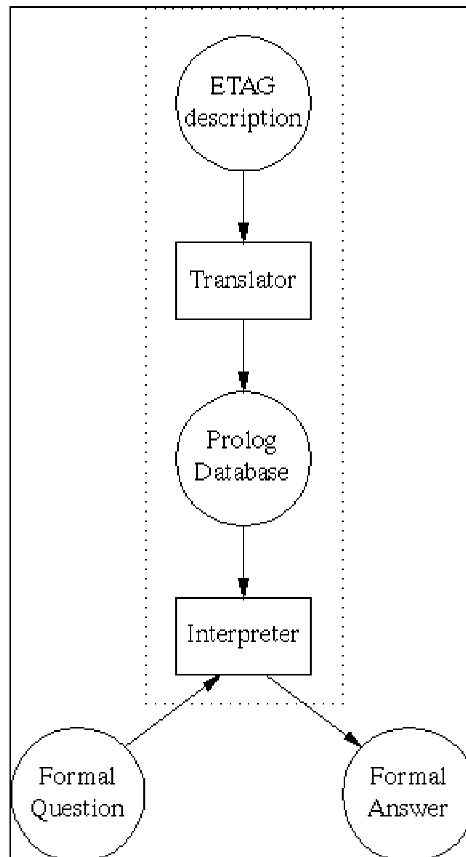

Figure 3. The Architecture of the Formal Help Machine.

In addition to using the database to answer questions, it may also be used for different purposes, such as checking the consistency of the ETAG representation. The translator is a Prolog program generator which is the result of applying Lex (Lesk 1975) and the compiler-compiler Yacc (Johnson, 1975) to a Backus-Nauer Form (BNF; Backus et al., 1964) representation of the ETAG notation, and compiling the resulting C program. To capture changes in the ETAG notation, and consequently its BNF representation, only requires one to re-generate the Translator program.

Recently, a simple but ingenious natural language front end has been designed and implemented by Go (1992). It is a simple tool, because it restricts the user in asking questions. However, it employs an intelligent mechanism to find out what the user's questions are about. Interaction is simplified by presenting the user with pre-defined menu-choices to select particular types of questions, like asking for the procedure to perform a task or explaining a concept.

How can I	**<task keystrokes or syntax>**
What happens if I	**<task effect>**
Does the system ..... if I	**<task event>**
What is	**<concept explanation>**

After specifying the type of question, the user still has considerable freedom to ask questions. Using standard language processing procedures (lexical and other analyses, a general purpose lexicon, syntax rules, etc.), a number of parse trees is created, representing alternative interpretations of what the user might have meant to ask. To select one of the interpretations as best, the system makes use of a encyclopaedia which contains knowledge rules about how ETAG concepts are referred to in the formal Prolog representation. From among the alternative questions, one is selected for further formalisation that comes closest to being answerable, or "unification", in Prolog terms. A minor drawback of this project is, that changing the (kind of) ETAG representation requires some effort to build a new encyclopaedia. Given that there is an ETAG representation and a general purpose lexicon, the rest of the process is automatic. Figure 4 presents the architecture of the Natural Language front-end of the static help machine.

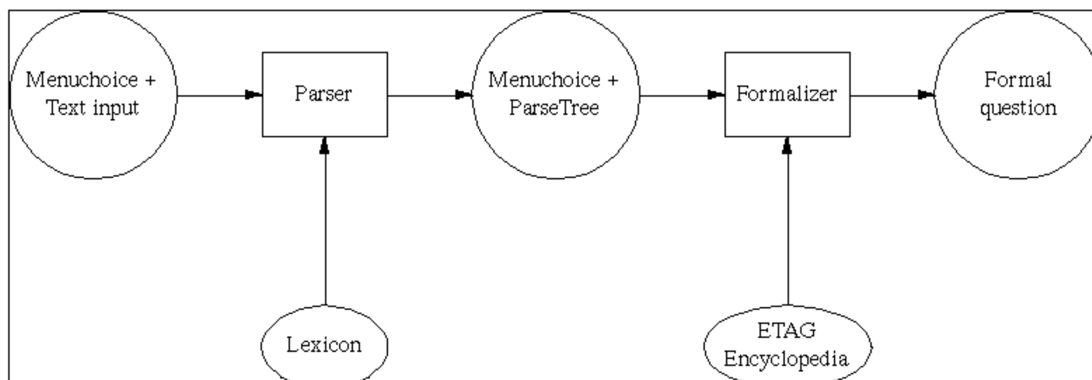

Figure 4. The Natural Language Front-End.

A practical example of how to use ETAG in combination with a Prolog database comes from a project by Smit (1991). Instead of directly translating from ETAG to natural language, a Natural Language Back End (NL-BE) to the database was built, which could produce a manual of the computer system in readable but rather imperfect English. Using an intermediate Prolog representation does not change the actual content or quality of the output of the help system. Provided that ETAG representations exist, manuals for different computer systems can be generated by running the ETAG translator and the Natural Language Back End.

Both of these projects show that it is feasible to use ETAG to create interactive help systems for static help to provide information that does not change during interaction with a system. In addition, using the Prolog database as an intermediate representation creates opportunities to use the information which is statically represented in ETAG in a more active way, by manipulating it. Example of this are: consistency checking of ETAG representations, and adding history information to the database in order to provide dynamic information about interacting with a system.

8.3.3 A Help Systems for Dynamic Information based on an ETAG Simulator

An extension of using an intermediate representation of ETAG information to provide static help information is to construct a help machine which also stores information about the interaction. Such information can be derived by having the (target) system report all its events into a log file, for further processing by a help system. However, this would limit the flexibility of the target and help system as a whole, because the help system would need to know what is happening within the target system, and the target system should be adapted to supply this information. In addition, it would require that the target system should actually be used in order to work with the help system.

To circumvent such limitations, and to allow the help and target systems to be used independently (including simultaneously), it was decided to simulate how the "virtual" target system should work on the basis of only the ETAG description of it. In principle, the user will not be able to differentiate between using the target system, the help system, or both of them simultaneously, because the system responses will be exactly similar. Except, of course, that in the first case, no help will be available. Similarly, because the input will be the same, the target system will not know the difference, although a help system might have grabbed all available input for its own processing, because the help system will pass on all command input that was originally directed to the target system.

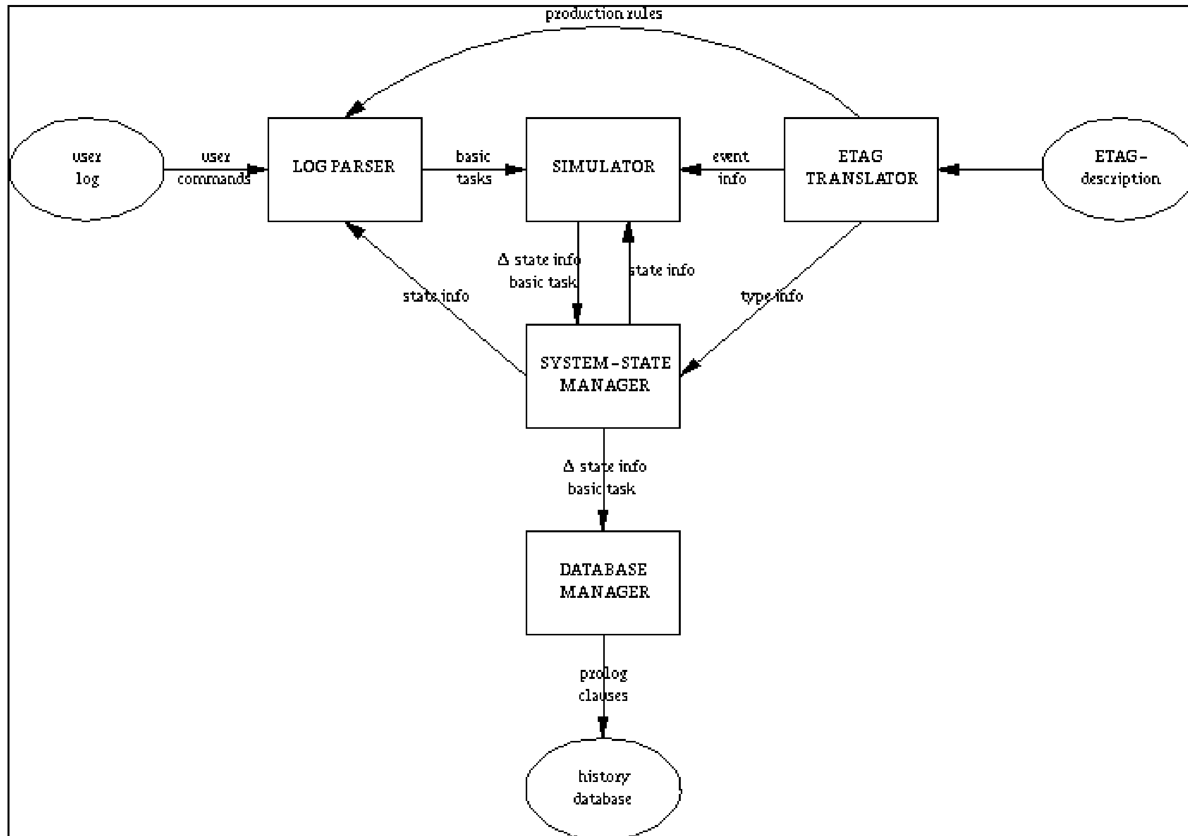

Figure 5. the Structure of the History System.

To build an ETAG-based help system for dynamic information, a History System is needed, which simulates the target system using an ETAG description of it, a log of user activities and a History Database of system states. A first project (Bakker and van Wetering, 1991) involved the design and implementation of the main software modules of the history system, and (the format of) the History Database. The format of the previously used Prolog database had to be extended in order to store 'facts' about past system states, starting from the so-called Initial State. The Initial State is the state the system is in when the application environment is entered for the first time. Whenever a user tries to execute a basic task, simulated basic events will happen. These event occurrences serve as matching points in the History Database, in whose terms facts about the interaction are stored, and in whose terms questions of the user all have to be reformulated. The structure of the history system is presented in figure 5.
The kernel of the history system is the System State Manager, which keeps track of, and provides information about the system state, and which passes state information to the Database Manager for storage purposes. The activities in the User's Log are parsed by a Log Parser to identify basic tasks and their arguments, guided by information from the History Database. Information about basic tasks is passed to the ETAG Simulator, where they are simulated as basic events. To simulate basic events, state information is provided by the State Manager, which also receives information about the system state (changes) to have it stored in the database by the Database Manager. Finally, information about the content and requirements of the basic tasks is provided by the ETAG Translator.

An interesting aspect of the project is the following. Earlier it was noted that the ETAG Translator could be used outside the context of help systems, to check the syntax and various semantic aspects of the ETAG representation. Here, in a similar vein, parts of the History System, and particularly, the ETAG Simulator can be used in a wider context, to inspect time-dependent aspects of computer systems, and even more promising, for prototyping purposes. The opportunities to use these modules for quite different purposes than what they were meant for seems to justify the modularity of the architecture.

Woudstra and Kohli (1992) designed and implemented the Log Parser. This module has a threefold function. The first function is to parse the user's actions to identify basic task invocations, on the basis of information about the production rule part of an ETAG representation, which is provided by the ETAG translator. The second function of the Log Parser is to identify the type and number of concepts (tasks, objects, places, etc.) in the user's task specifications, and to check if they are in accordance with the type information in the ETAG representation. The third function is to identify the existence of the type instances (files, folders, etc.) themselves, on the basis of information from the System State Manager and the History Database.

Two other projects concerning dynamic ETAG based help systems focussed on consulting the History Database to answer user questions. In these projects, the focus was on using the History Database, instead of feeding it. Figure 6 presents the overall structure of the Dynamic help system and its main modules, in which the parts at the top are concerned with feeding the History Database, and the parts at the bottom are concerned with using it to answer the questions of users.

In one project, Schep (1992) worked on the formal parts of a question answering system and, more specifically, the design of an FQL, a formal query language to interrogate the history database. This project started with an analysis of which kinds of questions involving dynamic information may logically be answered on the basis of the information in the History Database. Eventually, this system was able to provide dynamic information to (formal equivalents of) questions such as: "when did the file MyFile exist?" or "which files are currently owned by MyName?".

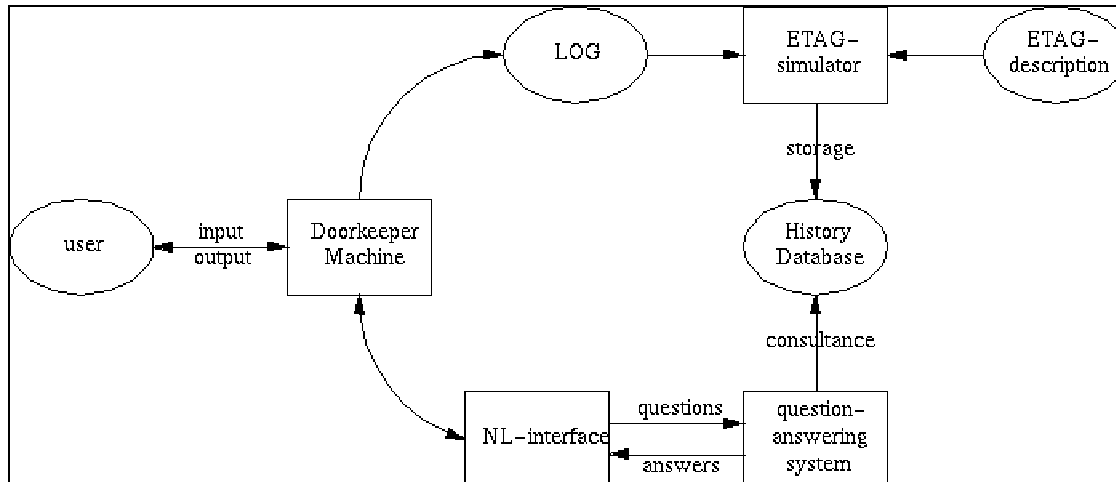

Figure 6. the Structure of the Dynamic Help System.

Users interact with the formal question answering system by means of the Natural Language interface in which two submodules address the translation of information between natural language and formal query language: the question module and the answer module. The purpose of Van Hoof's (1992) dynamic help project was to deliver a question module for asking dynamic information. In general terms, this would involve a lexical, syntactic and semantic analysis of a question of a user, and presenting the formal result to the question answering machine. However, for practical purposes, the process is turned upside down.
Instead of building a huge natural language processing subsystem, and a system to determine whether the questions of the user can be answered indeed, it was considered more appropriate to build a question module asking the users to provide the necessary and sufficient information. In this way, after the user has indicated that help is needed, the question module will ask particular questions until enough information is collected to formulate a question that can be meaningfully answered by the formal question-answering machine.

The dynamic help system is not complete. All the formal machinery is in place and, as such, the concept of ETAG-based help systems has been proven. A main omission is the natural language back end. Without a module to translate the formal -Prolog- answers into natural language, only closed questions yield acceptable "yes" or "no" answers. In addition, the complete independence of the static and dynamic help systems, and the most unnatural question and answer dialogue style of the dynamic help system prohibit practical usage of ETAG-based help. Creating a complete help system was considered unnecessary because it would move the aim of the investigations away from ETAG and, too much into solving practical problems.

To give an idea about the capabilities of the dynamic help system, the following is an example of a part of the interaction between a user and the dynamic help system. In order to save (much) space and to show the main points, it has been extensively shortened, and somewhat simplified. Suppose a user wants to ask a question like:

"Is there a file called MyFile [now]?"

To answer this question, it has to be presented to the formal question-answering system in one of its question formats with the appropriate variable substitutions. In formal terms, the question can be rephrased as asking whether there is currently ("during now") a file called "MyFile" that existed "during now":

```
query( [state, exists, true,
        [[object, file, ref1, the,
                [[[state, has_val, true,
                        [ref1, [attribute, name], [value, string, 'MyFile']]
                        [context, ref2, the, point, [[[during, now]]]]
                ]]]
        ]],
        [context, ref3, the, point, [[[during, now]]]]
]).
```

The natural language front end, or rather the question asking module, is to build the Prolog clause above, by systematically asking questions, such as:

do you want to Check something, or to Solve a problem [C/S] ? :
do you want to know something about an Object or a Task [O/T] ? :
do you want to know about a Value, a State or a Place [V/S/P] ? :

The formal answer to closed questions like the "Is there a file called ..." question will be a simple "yes" or "no". Open questions like: "which files belong to MyName" will produce a series of "VAR = MyFileName" Prolog clauses. Eventually, the natural language back end, or answer module will use part of the original question to present the answers to the user in a more natural form.

8.4 Summary and Future Outlook about ETAG-Based Help Systems

The studies reviewed above focus on using ETAG as the basis for static and dynamic help systems. It was shown that static help can be provided solemnly on the basis of the information contained in an ETAG representation of a computer system. Using ETAG as the single source of information also proved useful to assure the consistency among help information, and between help information and the workings of the computer system.

To design dynamic help systems, additional facilities are required, such as a History Database, and tools to manipulate that. Note that the advantage of using ETAG as the single source of information is not lost when introducing a database representation, as it only involves a direct and automatic translation from one formal representation to another, without any loss or addition.

The work reported above mainly involves implementing help systems. A typical and most important implementation problem encountered was how to deal with the -seemingly- ever

changing ETAG notation, and build systems flexible enough to avoid having to rebuild modules. Another problem is that 'local' changes in software modules, motivated by the need to solve specific implementation problems in one project, did sometimes lead to problems elsewhere. With the increasing number of student projects, and hence software modules, the urge for something like system administration increases.

A first move towards flexibility was to use general purpose tools, such as Lex, Yacc and the C programming language, instead of dedicated tools like SYNICS. Using these tools, changing the ETAG notation would in most cases only require having to create a new or changed BNF representation of ETAG, and a recompilation of the software.
A second decision to increase flexibility was the introduction of a kind of intermediate representation, the Prolog database to represent information apart from, and instead of, the ETAG model. This change was also required to allow dealing with dynamic help information.
Using a database and, in addition, Prolog's facilities to make logical inferences also provided the possibility to use the information in multiple ways. Not only to create help systems, but also, and this may prove important in the long run, to create tools for consistency checking and 'debugging' ETAG representations, and for using ETAG as a language for prototyping purposes.
Independent of the work on ETAG-based help systems, a number of approaches to automatically create help information have been developed in the context of model-based approaches to user interface design and implementation (Forbrig and Schlungbaum, 1999). Foley's UIDE (User Interface Design Environment; Foley and Sukaviriya, 1994) is an example of a model-based User Interface Management System (UIMS) which uses an (extensive) object-oriented conceptual design model as a knowledge base for generating both the user interface and the associated help information.
Even though the nature of the underlying conceptual model is different between ETAG and UIDE, the generation mechanisms are sufficiently similar to investigate the possibility of ETAG-based user interface generation.

An important problem facing ETAG-based help systems is that they are not able to provide information which is not already present in either the history of interaction or the ETAG representation. This problem concerns strategic help information. Tasks in ETAG are elementary tasks, which are presented to the user as indivisible units. Users however may also be interested in asking questions about tasks at a higher level, such as GOMS's unit tasks (Card, et al., 1983) and TAG's simple tasks (Payne and Green, 1986). These psychological unit tasks involve sequences of basic tasks and strategic decisions about how to combine them. Such decisions, however, fall outside the scope of ETAG, at least at present. This information has to added from a non-formal source, although it may be possible to use (parts of) the ETAG representation to do so in a systematic manner.

A final problem concerns the fact that ETAG representations do not seem to lend themselves easily for translation into a really natural English. It remains to be seen if it is worth the effort to translate a quasi English message, like "the task CHOWN changed the attribute OWNER of the object YOURFILE of type file", which is directly translated from ETAG information, into a more natural English. To create a more natural English would require far more powerful natural language modules than the ones which merely transform natural language

sentences to and from their formal equivalents. As such, projects like this may be feasible only for the longer term.

Finally, and for the longer term, it may also be worthwhile to consider extending the ETAG Simulator towards a real prototyping environment, and to create tools for analysing ETAG representations and/or interaction histories as a first step towards building systems in which users do not need extensive help facilities. Taking into consideration that well-designed user interfaces are transparent, and would not need extensive help, it would be more natural to develop the Simulator into a prototyping tool rather than using a help system for prototyping purposes.

8.5 Conclusions

In this chapter, it was argued that formal representation techniques can and should be used as the basis for intelligent help systems. We think that the examples provided justify this argument. The examples also provide evidence supporting our second argument: that is possible to use a specific formal model, an ETAG representation of the competent user's knowledge of a computer system as the basis for help systems, and moreover, as the single source of information. ETAG was originally only meant to be a specification method for computer systems, and not, at least, not directly meant for the purpose of creating help systems.

However, the information contained in an ETAG description showed to be useful and sufficient to create help systems to provide all kinds of help. Strategic information cannot be provided by a model for mere descriptive purposes. But information can be provided about both, the computer system as well as the interaction with it. Elsewhere, it was shown that ETAG can be used fruitfully to specify computer systems in various application areas (chapter 7; de Haan and van der Veer, 1992).

In this chapter, it was shown that ETAG can also be used successfully for a quite different purpose. Thus, increasing the scope of its applicability and showing the validity of the approach.

We also argued that a modular structure of help systems would be advantageous. Regarding this point, it was shown that modules which were developed as elements of help systems can be useful for quite different purposes. Finally, the help systems created on the basis of ETAG are strictly application independent. To provide help information about a system only requires an ETAG description of the particular system.

This chapter is based on: de Haan, G. and van der Veer, G.C. (1992b). Etag as the Basis for Intelligent Help Systems.

Chapter 9:

Conclusions

Why did you think that I am wasting my precious time to put ideas into words which I let pass through my mind before. I have rather better things to do: daydreaming for example... loafing around... mooning about... listening to my beard growing... more important matters than these tiring and vain attempts to acquaint mankind of my thinkings in the foolish hope that it would improve from that.

de Bie, W. (1988).

9.1 Summary of Findings

The central theme of this thesis is how to design usable human-computer systems. The thesis starts with a description of the background of the theme in cognitive ergonomics in chapter 1. Cognitive ergonomics is defined as the study of human-computer interaction that seeks to enhance the usability of computer systems by studying and applying knowledge about the cognitive aspects of HCI. In order to create better user interfaces it is necessary to follow a science and engineering approach and seek the creation of a general knowledge base and general theories to replace the multitude of individual facts, research fashions, and individual expertise.

It is further necessary to replace the software-oriented view of user interfaces by taking a cognitive stance towards user interfaces and regard them in terms of the knowledge that users must have to use them to perform their tasks.

Finally, it is necessary to develop design methods for user interfaces as a vehicle to bring about the required changes and to improve the usability of computer systems in everyday practice.

In order to further the development of cognitive ergonomics as a science and engineering practice that is concerned with the design of usable human-computer systems two main research questions are raised:

- What is a good representation to model usable human-computer designs for design purposes?
- How to create a design method based on a good representation of HCI design?

To answers these questions, each was divided into four subquestions. To determine a good representation for HCI design, the following subquestions were asked:

1. What criteria should be used to assess and evaluate models for design representation?
2. Which models are available or may be developed into models for design representation?
3. Which model is most suitable or promising with respect to the assessment criteria?

4. How well does the selected model fare to answer different design questions and how can it be improved?

To answer the question of how to create a design method on the basis of the selected representation (ETAG), the following four questions were asked:

5. Is there a general or 'standard' method for user interface design and how does it fare?
6. What is the appropriate view or definition of the user interface as a concept?
7. What does ETAG-based design look like?
8. How can ETAG be used in the different design stages and how should it be improved or complemented?

9.1.1 Assessment Criteria for Design Representation Models

Models to represent user interfaces for design purposes are artefacts that specify things about the users of a computer system and are meant to help designers perform the task of representing user interfaces for design purposes. In chapter 2, a number of formal modelling techniques in human-computer interaction is described and criteria are developed to assess the techniques and select a most promising one to represent user interfaces for design purposes.

Four criteria were developed to assess the usefulness of the selected models:

- *Completeness* with respect to description of the different levels of abstraction of the system. Given that the user's knowledge, user-system interaction, and designs specifications are layered, a model should provide a complete and accurate representation at the different levels of abstraction of the design.
- A *wide applicability* with respect to a variety of different kinds of users, interaction styles, and types of tasks. In order to create a general purpose notation for user interface representation, formal modelling techniques should be applicable to a wide variety of different kinds of users, styles of interaction, and types of tasks.
- *Validity* with respect to using the representation for analysis and prediction of usability aspects. Formal specification models should also allow being used to represent, analyse and predict usability aspects of the user interface. From this, it follows that both the representation itself and the results of using it for analysis and prediction purposes should be valid.
- *Usability* of the representation itself in terms of the functionality, ease of use and ease of learning for its users. Formal modelling techniques are themselves artefacts, and as tools to help designers perform their tasks, the models themselves should be functional, easy to use, and easy to learn and remember.

9.1.2 Models for User Interface Design Representation

Users create a mental model of the computer systems they use to perform their tasks, and the success of performing the tasks critically depends on the quality of the mental model. In order to create usable computer systems, the representations themselves should address that users create mental models and allow, in one way or another, the specification of user interfaces in

terms of the user's mental model. Since it is not possible to have direct access to users' mental models, the second best solution is to describe the knowledge that should be in the user's mental model as a necessary albeit not a sufficient condition for successful task performance.
To designers it is advantageous to have representations that are as precise as possible and, preferably, to have representations that can be used for other purposes than specification only, such as the automatic generation of software and the mathematical analysis of characteristics of the design in early stages of the design process and well before it is built. In principle, these advantages may be attained by using formal models.
On the basis of the two considerations: concern with the user's mental model and the advantages of formality, only formal models that allow for the specification of the user's knowledge or behaviour were taken into consideration. Models which allow only for an external description of either the user's behaviour, the user-system interaction, or the behaviour of the system as a whole have not been taken into consideration.
From the cognitive ergonomic literature a number of formal modelling techniques for user interface design representation is described and reviewed in chapter 2:

Models for Task Environment Analysis:
- ETIT (External Internal Task Mapping Analysis; Moran, 1981)

Models to Analyse User Knowledge:
- Action Language (or Psychological BNF; Reisner, 1981, 1983)
- TAG (Task-Action Grammar; Payne, 1984; Payne and Green, 1986)

Models of User Performance:
- GOMS (Goals, Operators, Methods and Selection Rules; Card, Moran and Newell, 1983)
- CCT (Cognitive Complexity Theory; Kieras and Polson, 1985)

Models of the User Interface:
- CLG (Command Language Grammar; Moran, 1981)
- ETAG (Extended Task-Action Grammar; Tauber, 1988, 1990)

9.1.3 A Most Suitable Model for Design Representation

In chapter 2, in applying the selection criteria it is noted that none of the modelling techniques meets all the criteria of completeness, wide applicability, validity and ease of use. Most models are restricted to task-action modelling and fail to address the presentation of information and the semantic aspects of tasks and devices. Both Extended Task-Action Grammar (ETAG; Tauber, 1990) and Command Language Grammar (CLG; Moran, 1981) provide a complete description of user interfaces at the different levels of abstraction, and from these, ETAG is selected for further investigation because it uses a simpler and cleaner formalism.
As a specification model for design purposes, ETAG and CLG are most complete. Regarding the width of the applicability, the models for performance prediction suffer from the need to state restricting assumptions, such as modelling error-free expert behaviour only. Both CLG and ETAG are models of competence knowledge which have a cleaner architecture, and are specifically meant to model mental representations.
The validity of the more restricted models would be easier to show. However, for design purposes, the analytic validity of differences between user interfaces is advantageous but it is

not essential and the validity of predicting expert task times hardly matters at all. For design purposes it is essential that the representation is valid with respect to the psychology of the user and, as an approximation of that, the psychological theories of knowledge representation (hence: chapter 3). Regarding the validity for design representation, the advantage lies with CLG and ETAG and, considering that CLG has an ad-hoc formalism whereas ETAG has one that is specifically intended to be psychologically valid, ETAG should be regarded as the better model.
Finally, when comparing CLG and ETAG, only CLG has been subjected to usability investigation. The results of the study indicated problems with the size and the formalism of CLG. Provided that ETAG has a much neater formalism, it is expected that it will be more usable.
The main conclusion of chapter 2 is that ETAG should be selected for further investigations and that these investigations should also address psychological validity (chapter 3) and the validity as a tool for analysis and prediction (chapter 7), its use as a tool for design specification (chapter 6) and for task analysis (chapter 5), its use as a source of information about the user interface (chapter 8), and as a the central notation for a design method (chapter 4).

Chapter 3 describes and discusses the psychological foundations of ETAG and how these determine the structure and content of the formalism. It describes how users and user interfaces are viewed in ETAG and in several related models, it briefly explains the ETAG formalism, and it discusses the psychological considerations underlying the structure and the content of the ETAG formalism and its use of existential logic.

With respect to structure, ETAG follows Payne and Green's (1986) use of a feature grammar in TAG because of its ability to capture the perception of family resemblances between different but otherwise similar tasks. Family resemblances cannot be explained by less advanced grammars, such as BNF. ETAG further uses a slightly different adaptation of Moran's (1981) levels of abstraction (semantic, syntactic, etc.) to describe user interfaces. Evidence from psychology and psycholinguistics points out that knowledge is layered, even though most formal modelling techniques do not distinguish different levels of abstraction or do so in arbitrary ways. The structure of ETAG is also motivated by non-psychological considerations, by the need to make explicit decisions at certain points during the design.

With respect to content, ETAG follows psychological theories that describe semantic memory as a conceptual structure with basic scripts or schemes. From Klix (1989), the idea is adopted that events are represented by means of a verb as core concept, surrounded by standard relations such as actor, object and purpose. Several further adaptations concerning the form and the content of the ETAG notation to represent knowledge are derived from Sowa's (1984) and Jackendoff's (1983, 1985) interpretations of existential logic, such as the use of basic ontological categories. In comparison to e.g. predicate logic, existential logic is much better able to express knowledge in a psychologically valid way and it is much easier to understand.
The chapter further exemplifies how the structure and content choices have been interpreted in the ETAG formalism, and concludes with a warning that, even though ETAG is built from the start on psychological considerations it does not necessarily follow that ETAG representations themselves are automatically valid.

9.1.4 Using ETAG to Answer Design Questions

ETAG representations have been created of different user interfaces, task situations, and design problems to determine its usefulness for the purposes of task analysis, design specification, user interface analysis, and as a source of information about the user interface.

Chapter 5 discusses *task analysis* and how the ETAG notation may be used for this purpose. In applying ETAG as a task analysis tool to model the high-level aspects of a real-world task situation, it was hypothesised that the formality, grammaticality and object-orientation would influence the usability of the model to give a complete and accurate representation of the situation.
It was established that ETAG is well able to describe the goals, the tasks, the formal aspects of the organisation, and the task objects and tools in the task situation but that it is less suitable to describe the temporal relations between tasks, and to describe the aspects of the task situation that are not formally related to the tasks, such as the causes of problems outside the work context, informal motivations, and personal experiences. Because ETAG is less suitable to describe the temporal relations between tasks and because it does not provide a good overview of the relations between tasks ETAG should not be used to replace the task-decomposition tree.
ETAG representations of task situations rapidly become very big and thereby so complex that it is necessary to simplify the notation. Nevertheless, it is concluded that it is both possible as well as advisable to use ETAG for task analysis representation.

Chapter 6 discusses how to create an ETAG representation as a *design specification* in user interface design. The method that is proposed is a generalisation of the experience gained in creating ETAG models of user interfaces in different projects. The method identifies six steps to make the process manageable:

- A raw list of tasks, objects, attributes, etc.
- A list of concepts for the canonical basis
- A preliminary list of tasks and basic tasks
- A first specification of the elements of the UVM
- A complete specification of the UVM
- A specification of the perceptual interface

The result of each step is the specification of a required piece of information at an increasing level of detail until, eventually, the specification is complete. For each step, guidelines are provided concerning how to proceed. An extensive example from a research project is provided to illustrate the whole process and each of the steps as well as to demonstrate how it may be necessary to deviate from the sequential process and re-iterate over results.
An extensive example of applying ETAG for design specification in a research project illustrates the method and demonstrates when it may be necessary to deviate from the process and iterate. The example further indicates that ETAG specifications may become quite large and difficult to oversee, and that ETAG is not able to represent the presentation interface.

In chapter 7, ETAG was applied to the *user interface analysis* of several page tools, a spreadsheet application to exchange currencies, and two email packages that differ in terms of sophistication. The purpose of these studies was to establish the strong and the weak points of the ETAG notation and to determine how the notation should be developed further. The main conclusions that were derived from the studies are that ETAG is able to yield complete descriptions of a variety of user interfaces and that it is well able to describe the main differences between interfaces.
ETAG descriptions are not too difficult to understand but it is not at all easy to create and change them. Additional tools and directives are required to facilitate creating and handling ETAG models and to improve the prediction of usability characteristics of the user interface. Finally, ETAG is in need of facilities to specify and analyse the perceptual characteristics of a user interface (the presentation interface) and facilities to deal with advanced dialogue control structures and user interrupts.

Chapter 8 discusses the use of ETAG as a *source of information* about user interfaces in order to automatically generate user help information as an example of an additional way to utilise formal representations of user interfaces. In terms of increasing sophistication, the information contained within a formal representation of a user interface can be translated directly into the user's natural language, it can be interpreted and translated to allow for more flexibility, and the information can be used to drive a simulation, for example, in parallel to running the actual application as a black box, to generate context sensitive information about the dynamic aspects of using the computer system.
The results of these demonstration projects indicate that although there are some major obstacles left to provide, for example, genuinely natural language and strategic help information, in principle it is possible to use a formal model of a user interface for the automatic generation of help information, which is not possible when using another type of user interface specification. Extending these findings, it is hypothesised that it is feasible to generate software, create prototypes, and even to run computer systems, on the basis of a user-oriented user interface specification such as an ETAG model.

9.1.5 The Received View as a Method for User Interface Design

Chapter 4 introduces the concept of the received view on user interface design as the greatest common denominator or the ruling paradigm among cognitive ergonomic design methods. The received view is assumed to have two specific characteristics:

- It is supposed to express what is and what is not worthwhile to investigate
- It is supposed to express which aspects should and which need not be considered in user interface design.

Unfortunately, the received view in cognitive ergonomics only describes a general structure for the design process and does not specify anything about the contents of either the design process or the research field. The consequences are twofold. First, there is no guidance as to what research is necessary and this has led to a lack of general theories and methods and an abundance of popular ideas and individual facts. Second, there is no guidance about what is the nature of cognitive ergonomic design which has led to e.g. the general adoption of

software design methods with the undesirable additions of trial-and-error iteration and the idea that user interfaces are mere pieces of software.
To improve things it is proposed to (1) view user interfaces in terms of the user's knowledge about performing tasks, (2) develop a cognitive ergonomic science base with the view on user interfaces as a guiding principle, and (3) develop engineering design methods on the basis of the view and in such a way that iteration is avoided as much as possible and only allowed when it can be managed.

9.1.6 A Defining View of the User Interface as a Concept

In order to overcome the limitation that the received view in cognitive ergonomics only provides structural guidance to user interface design, chapter 4 proposes the ETAG view of user interfaces as the user-oriented view of the user interface that includes everything that a perfectly knowing user knows about the system. This view does not only go beyond what is generally considered to be a user interface but it also replaces the software-oriented view with a cognitive ergonomic one that is based on user knowledge.
People use computers to perform tasks, hence: engage in psychological behaviour. It is necessary to reject the software engineering view of a user interface as a piece of software and to replace it by the definition that the user interface includes everything that users must know about the computer system they use to perform their tasks.
This definition differs considerably from the class of software-oriented definitions of the user interface. In increasing sophistication this class of definitions includes the magazine view which defines user interfaces as the visible elements of computer systems, the toolkit view which defines the user as the set of widgets ("window elements") and the interaction dialogue with them, and the UIMS view ("User Interface Management System") whose definition encompasses both presentation, dialogue and functionality but fails to relate them to the semantics of the user's task world.

From the idea that to the user and, as such, for user interface design, the meaning of tasks, task commands and task objects (the semantics of the task world) are essential, user interface design is regarded as specifying the task world from the user's point of view. In ETAG, this is done by specifying what users must know for successful task performance. The view on the task world that the computer system offers is called the User Virtual Machine which is a *User's* virtual machine to express that the computer system is only of interest for the user to perform tasks, and it is a user's *Virtual* machine because it is a specification in terms of user knowledge, which may be different from the actual technical workings of the machine.
In ETAG, a user interface specification consists of the conceptual model, which consists of the user virtual machine (the conceptual specification of the task world) and the dictionary of basic tasks (the system's commands), and the perceptual machine, which in its turn consists of the language interface (the interaction language) and the presentation interface (the display information).

9.1.7 The Structure of ETAG-based User Interface Design

In chapter 4, ETAG-based design is described as a design method with three distinctive features. First, since user interfaces are regarded as a tool for task performance in ETAG-based design, the design process starts with an analysis of the user's task situation.
Secondly, designing user interfaces is regarded as specifying, by means of the ETAG formalism, a competent user's knowledge about the task world. Special attention is paid to the conceptual specification of the user interface as a distinct and central step in the design process. In all other respects, the structure of the ETAG-based design process is similar to that of different user interface design methods or the received view.
Thirdly, to stimulate improvements to the design process and to facilitate managing the process, in ETAG-based design iteration takes place as much as possible within design steps and as little as possible between design steps.
Given the three distinctive characteristics, ETAG-based design is described as a structured design process that consists of task analysis, task modelling, and the conceptual and perceptual specification of the user interface, followed by a software implementation and the evaluation of the new work situation.

Task and context analysis is concerned with collecting information about the old and about related task situations to enable reasoning about the user's tasks, distinct from any implementation details. The results of task and context analysis are collected in task model 1 which includes at least a task decomposition tree of the old situation.

Task design is concerned with the creation of a new task situation at a high level of abstraction without referring to any specific technology and adapted to the task context (e.g. the organisation, the way of working, the goals and responsibilities, the tools). Also during task design an allocation of tasks and responsibilities to the user, the computer system or their combination takes place. The result of task design is task model 2 which includes a task decomposition tree of the to-be task situation.

Conceptual design is concerned with the semantic specification of the user's task world by means of the ETAG formalism. It consists of the specification of concepts of the canonical basis, the objects, attributes and events of the user virtual machine, and task-commands of the dictionary of basic tasks. Conceptual design also includes a low-level allocation of tasks and responsibilities.

Perceptual design or user interface design in the traditional sense, is concerned with the specification of the perceptual aspects of the user interface in order to support understanding and using it. During perceptual design, the interaction language, the visible and otherwise perceivable aspects of the interface, and the metacommunication such as documentation and help facilities are designed and specified. During both conceptual and perceptual design usability studies and prototyping are used to evaluate design options.

During *software design*, once all aspects of the user interface design are completed, the specification, implementation and introduction of the computer system takes place. In order to create opportunities to improve the design method as well as to allow for local adaptations of the system, software design is completed with an evaluation of the task situation.

In comparison to other cognitive ergonomic task and model-based user interface design methods, such as ADAPT (Wilson et al., 1993), MUSE*/JSD (Lim and Long, 1994), and DIGIS (de Bruin et al., 1994), there are some differences with respect to the concern with software design, the role of the formalism in the process and the amount of tool support, but the conclusion of chapter 4 is that the similarities are greater and more important than the differences.

9.1.8 Using ETAG throughout User Interface Design

Since question four already dealt with opportunities to use and improve ETAG for purposes different from design specification, the discussion will only deal with using an ETAG representation as the central model or the red thread throughout the design process.

Chapter 5 discusses the use of ETAG during *task and context analysis* and, hence, for *task design*. The chapter analyses the purpose of task analysis and discusses the depth, the scope, the level of detail and other aspects of the process.
From an analysis of the missing issues in early approaches to task analysis, it is concluded that, in comparison to the more traditional approaches, at least in principle, ETAG provides additional means to analyse the non-physical aspects of task performance including the use of objects and tools, cognitive task aspects, the organisation among tasks, and, to a limited extent, the social and organisational aspects of work.
ETAG is useful for representation but needs to be complemented by task decomposition trees to provide structure overviews and to specify task order. Also, additional facilities are required to annotate the aspects of task situations that are difficult to formalise.

Chapter 6 and 7 discuss the use of ETAG for specifying the *conceptual design* of user interfaces. Chapter 6 provides a method of which the first five of the six steps are concerned with conceptual design, as well as a set of guidelines and an example. The result of each step is the specification of a required piece of information at an increasing level of detail until the specification is complete. An extensive example from a research project is provided to illustrate the whole process and each of the steps as well as to demonstrate how it may be necessary to deviate from the sequential process and re-iterate over results.
The chapter briefly discusses the use of ETAG in ETAG-based design compared to other task or model-based approaches to user interface design. The chapter concludes that it is feasible to use a psychologically motivated representation to specify user interfaces, even without solving the basic problems of cognitive psychology. In addition, it concludes that several theoretical problems about the ETAG notation, such as the size and complexity of the specification, can be solved in practice by using complementary tools.
In chapter 7, ETAG is applied to various user interfaces to evaluate ETAG as a tool to analyse user interfaces. With respect to its use for conceptual design representation, the main conclusions are that ETAG descriptions are complete and show important differences between interfaces. ETAG descriptions are not very difficult but due to size and complexity they are not easy to create or change. In order to model large and complex interfaces, drawing and diagramming tools may be useful to present overviews. ETAG representations may benefit from the introduction of "menu tasks". User level tasks and basic tasks are both important in design. In addition, the introduction of user level tasks with control structures

may help to reduce the size of ETAG specifications because they make it unnecessary to decompose such tasks into basic tasks (hence: without control structures).

The utility of using ETAG during *perceptual design* is discussed in chapters 6, 7 and 8. In chapter 6 it is noted that, in principle, it should be possible to automatically generate parts of the perceptual interface from an ETAG specification even though, in practice, no tools exist for this.
Provided that a higher-level ETAG specification is available, specifying the interaction language is a relatively straightforward process because there is a one-to-one relation between each of the user's tasks in the dictionary of basic tasks and each of the top-level specification rules of the interaction language. For the rest, specifying the interaction language is a matter of choosing and specifying the command syntax, the (interaction) reference style, the naming and the behavioural actions.
ETAG has no facilities to specify the presentation interface. Albeit mainly a theoretical problem, in practice it is possible to use different tools for this in combination with the information from the ETAG specification.
With respect to the interaction language it is noted in chapter 7 that ETAG is not able to describe dialogue control elements such as user interrupts. With respect to the presentation interface this chapter also concludes that ETAG lacks an adequate structure and the required concepts to address the presentation interface.
Regarding the use of ETAG representations to generate metacommunication facilities, such as user documentation and help information, chapter 8 concludes that ETAG representations provide sufficient information about the user interface to create these aspects of the perceptual interface either automatically or manually.

9.2 Discussion and Future Research

This section discusses two problems concerning ETAG as a cognitivistic model. Cognitivism states that users act upon a mental representation of reality rather than upon characteristics of reality itself. As a result, cognitivistic models, such as ETAG, emphasise the structure of the cognitive representation and say little about the perceptual sources of knowledge and about the development of it. Since there is evidence which indicates that perception and development are important, approaches like ETAG are faced with problems.
The first problem concerns the implications of ETAG's lack of facilities to specify the presentation interface. The second problem concerns the question for which types of user interface ETAG and ETAG-based design are and are not suitable in relation to the dynamic aspects of human-computer interaction. This section provides a brief introduction to cognitivism and its problems. Each of these problems will be discussed in more detail and in relation to ETAG and ETAG-based design in the two subsections that follow.

The basic assumption of cognitivism is that human beings act upon an internal representation of the outside world. Problems occur when the assumption is pushed too far and the mental representation is not taken for an abstraction of the state of mind but rather as a real and singular representation of the state of the outside world. Along with the necessary working hypothesis of psychology, that it must be possible to investigate and to describe mental

phenomena, the result is the idea that it is possible to describe the mental representation or its contents.
Applied to HCI design and considering that users create mental representations, the idea becomes that it is possible to prescribe beforehand (!) how users will or need to create a mental representation in order to use an interface. In practise, few if any designers will admit that they are designing their user's actual mental models for the simple reason that mental models are not directly observable entities which, as such, cannot be explicitly specified. Nevertheless, every designer or design method that uses a psychological description of an interface that is designed indirectly assumes, not only that users create mental representation, but also that users do so, in one way or another, according to the design specification. This idea can be named the necessary working hypothesis of cognitive ergonomics.

The first problem with respect to cognitivism concerns the fact that the conceptual model is the corner stone of ETAG-based design whereas relatively little importance is given to the presentation interface. Cognitivism assumes that people act on the basis of an internal mental representation of the outside world. In ETAG the assumption is used as the basis to describe what users should have in their mental representation for successful task performance. However, users do no always seem to act as if they first create a mental model and subsequently use the model to plan their actions. In addition, there is evidence that creating a mental model may not be a universal type of behaviour but rather the result of having to use particularly bad and complex types of user interfaces like command language interfaces. In both cases, it is necessary that more attention is paid to the design of the presentation interface, even though ETAG has no facilities to specify the presentation interface.
The second problem is a new one and concerns the question for which types of user interface ETAG should and for which it should not be used. Chapter 5, and to a limited extent, chapter 6, discuss the suitability of ETAG as a formal, grammatical and object-orientish model for different application domains to determine which additional means are required to provide a complete representation of relevant aspects of the task situation and the user interface. Throughout the preceding chapters, the validity criteria were internal: provided ETAG is able to represent user interfaces, for which aspects of user interfaces does ETAG provide a complete and valid representation and for which does it not? In this case, the criteria at stake are external and put the question forward: provided that the assumption of cognitivism may and may not hold with respect to representing particular types of user interfaces, for which types of user interfaces does ETAG provide a complete and valid representation and should it be used and for which does it not.

9.2.1 ETAG-based Design and the Presentation Interface

In chapter 4, it is argued that ETAG as a knowledge specification of the interface being designed differs from the mental model that users will develop when working with the system. That a particular user's mental model is less complete and less accurate than the ETAG representation is not a problem since ETAG intends to represent what a perfectly knowing user knows. Likewise, that users organise their mental representation of the user interface differently from the ETAG formalism is neither a problem since ETAG intends to provide a valid description of knowledge content rather than a psychologically real formalism.

There is some evidence that cognitivism may not fit certain HCI problems. First, studies by Briggs (1988, 1990) and Barnard, Ellis and MacLean (1989) question the relation between the design model of the system and the development of expertise. According to these studies, the development of the user's knowledge has little or no relation to the design model of the system: many features of a system remain undiscovered, user knowledge develops in an idiosyncratic way, and there is no clear relation between the user's knowledge and performance. The implication is that although models like ETAG may be useful for design representation, such models may not be useful to analyse user interface problems and, instead, more attention should be paid to models that focus on the development of users' knowledge.

Secondly, van der Veer (1990) and Mayes et al. (1988) compared differences in mental model development between users of character-based and graphical interfaces and found that graphical interface users have less well developed mental models, indicating that when there is no need to, subjects will not learn the conceptual structure of an application. The implication from these studies is that the coherence and consistency of the conceptual model may actually be less important than design characteristics of the presentation interface, which should not be regarded as a mere means to help users acquire knowledge about the conceptual interface but, rather, as the single most important source of knowledge about the interface.

Finally, Holst et al. (1997), Golightly and Gilmore (1997) and Cockayne et al. (1999) used different (graphical) presentation interfaces and varied things like interface style and response times while maintaining the conceptual structures of the underlying application and found that the presentation interface influences both the performance levels as well as the task strategies. Whereas the results from van der Veer and Mayes may be explained because of the conceptual differences between the user interfaces, these studies seem to imply that, indeed, the design of the presentation interface is very important with respect to user performance.

Most model-based approaches to HCI design, including ADEPT (Wilson et al., 1993), MUSE*/JSD (Lim and Long, 1994) and ETAG-based Design (de Haan, 1994, 1996, 1997, chapter 4) focus on the design of the conceptual model and treat the presentation interface as a mere means to support the users in discerning the conceptual model. This corresponds with Norman's thesis about mental models (1983).
The studies that were mentioned as possible evidence against the cognitivistic view may indicate that cognitivism might not be the proper method to deal with things that people must do as opposed to things they must learn or know because too much value is assigned to the conceptual interface in comparison to the perceptual interface of applications. Note that from these studies it does not necessarily follow that the conceptual model is useless but only that model-based design approaches should give more weight to the presentation interface and to the dynamic aspects of learning to use a user interface.

Two things are necessary to better address the presentation interface in ETAG-based design. First, a specification of the presentation interface may be added by providing ETAG-based design with either additional design steps or a parallel design stream or a combination of both. Given that there are no indications that the presentation interface is so much more important than the conceptual interface, the option to completely do away with the conceptual interface

is not considered here.
Secondly, since the presentation interface is an important source of information while actively using the interface, more attention needs to be directed at the dynamic aspects of human-computer interaction. This will be discussed in the next section in which the use of dynamic models in general, and not just regarding the presentation interface, is presented as an option to extend the applicability of ETAG-based design to different types of user interface.

ETAG may be extended with better facilities to specify the presentation interface. In order to do this, it is necessary to develop a notation to describe the presentation interface (the display screen or perceptual space of the device) and to connect the description to both the elements in the user virtual machine, the tasks in the dictionary of basic tasks, and the rules of the interaction language. Richards et al. (1986) show that it is possible to extend a notation with perceptual elements, and May et al. (1995) show that it is also possible to completely describe visual displays. It will be very difficult, however, to create a specification notation that describe presentation interfaces in the same psychologically valid way as ETAG allows for with respect to the conceptual interface.

A first option is to *add a parallel stream* to ETAG-based design that is dedicated to presentation issues which would allow e.g. a graphical designer to create the presentation aspects of the user interface while the cognitive ergonomist concentrates on the conceptual design. This option must be rejected on grounds that using a second representation next to ETAG as the red thread throughout the design process would needlessly increase the amount of redundant information and it would increase rather than decrease the chance that communication problems occur in the design process. Co-operation with software engineering design being difficult enough does not motivate adding yet other parties.

A second option to provide better facilities for the design of the presentation interface is to *add sequential steps to* the ETAG-based design process. For example, after the specification of the user virtual machine, steps may be added to specify the type of interface and the perceptual representations of its elements., and after the specification of the dictionary of basic tasks and the production rules, steps may be added to complete the presentation interface specification. A major disadvantage and a reason to reject this option is that it would increase the number of iteration cycles between and over design steps and, thereby, increase the difficulty of managing the design process.
A third and preferred option is to *add activities to each design step* and add presentation aspects to specification elements which are already there. This might increase the size and complexity of the ETAG specification but avoids the problems that are associated with the other options: iteration may still be restricted to within design steps and only a single specification needs to be maintained.

In standard ETAG, the presentation interface is specified only after the completion of the conceptual interface and during the design of the perceptual interface. The example of the Comris system (chapter 6) indicates that specifying the presentation interface at this stage is too late because investigating prototypes created ideas for improvement which required a redesign of the user virtual machine. Making decisions about principle options regarding the interface style, the interface metaphor, and the perceptual representation of the objects,

attributes, etc. during the specification of the user virtual machine makes it possible to use this information when it is necessary. It may be expected that the availability of, at least, a starting point for the design of the presentation interface will smoothen the transition from the specification of the user virtual machine to the dictionary of basic tasks and to the specification of what's left of the perceptual interface.

9.2.2 ETAG-based Design and different Types of User Interfaces

Cognitivism works best when it is easy to create an internal representation of outside phenomena, as is the case when there is a clear distinction between form and content, and between data and control. The distinction between form and content and between data and control is most clear in command line interfaces, particularly when there is a single thread of interaction between one user who issues commands to one computer-system which presents the resulting data.
The problem is not whether or not ETAG is able to represent things like multitasking or not. The ability do model multitasking is only relevant in models of the user interface as a software entity. It may be important in models of user performance but it is not a principle concern in user knowledge models. However, when tasks are not performed one after the other, factors may become important for successful task performance that differ from those which are stressed in models like ETAG, such as consistency and coherence, with the result that ETAG becomes less applicable. This is similar to the argument in the previous section with respect to the difference between graphical versus command-line interfaces.

The problem is that certain developments in HCI may make cognitivistic models less suitable to represent all types of user interface. First, ping-pong human-computer interaction is getting increasingly rare. Interaction has become much more complex by the virtually simultaneous use of multiple applications, computer systems, and modes of interacting. For the user this makes it more difficult to keep track of the state of things, with the result that the interaction cycle becomes more important and, with it, the importance of information presentation and feedback. To use modern office software that is based on 'customer demand', it may not be a good idea to attempt to determine how it is organised. Instead, it is presumably better to use heuristics and local optimisation strategies to perform tasks, and this is precisely what direct manipulation interfaces afford.

Secondly, HCI has also become less predictable or less deterministic because of the introduction of human and computer agents into the interaction process. In principle (see: e.g. chapter 8), it is possible to use ETAG to model the changes in the task domain. However, even though ETAG is able to represent how objects and agents are created, change attributes, and may cease to exist, it is not realistic to use ETAG to model what a perfectly knowing user might actually know about a distributed agent system like the example of the Comris system in chapter 4. Even though knowledge about the rules of engagement remains necessary to use multi-agent systems like Comris and computer games, for using such systems successfully, performance factors may be more important.

Finally, the difference between data and command has become blurred by the introduction of free mixtures of data and commands, active data objects, virtual and immersive environments,

and the use of "natural" constraints on the user's actions. Systems in which, for example, icons or browser links may represent commands, data objects, or both at the same time are difficult to represent in models which assume a clear distinction between the two. In addition, at least for the most frequent or important tasks, systems like browsers are characterised by very simple and easy to learn interaction languages and device structures. The difficulty of interacting with systems which mix data and commands is in the structure of the task domain and in the mapping between the domain and how it is represented by means of the interface concepts. Because models like ETAG can be used to describe both user interfaces as well as task domains it may be possible to use these models to describe such mappings but it is probably better to use dynamic learning models instead.

The problem of ping-pong interaction getting rare is not a real problem for models like ETAG. If the user's task complexity does indeed increase, it does not mean that models for user interface representation suddenly stop being useful. It does mean that more attention should be spent on modelling the dynamic aspects of the interaction. However, when software companies, for whatever the reasons, create software that is purposively not usable and thereby require users to develop unusual skills there are still no scientific reasons to go along with such practices.
The problems of the decreasing predictability and the increasing use of shallow user interfaces point at genuine problems for cognitivistic models for which it is necessary to add facilities to the available models to deal with performance aspects, like a psychologically realistic ETAG interpreter, or to propose alternative models and tools to address performance issues.

The studies that were mentioned in the previous section about the importance of the presentation interface and the problems of task-performance complexity, dynamic agency and the structure of the task domain as factors that become increasingly important for successful task performance all exemplify the relevance of dynamic aspects of human-computer interaction. Chapter 2 discussed GOMS (Card, Moran and Newell, 1983) and CCT (Kieras and Polson, 1985) as models to predict user performance. In this case performance models are not what is required. GOMS, CCT and all other models discussed in chapter 2 model the user interface in a static model and subsequently use that for either descriptory, analysis or predictory purposes. Since these approaches use a static model of the user interface, they will be called static models. What is required in this case are tools to describe, analyse and predict the interaction between the user and the system, the problems that occur during the interaction, and how the interaction evolves dynamically as a result of learning. These models will be called dynamic models.

Static models of the user interface are excellent tools for specification and representation. Creating specifications is generally not easy, and in software-oriented design, user-centred specifications require additional effort. Specification models tend to be sizeable and complex, but no more than their software engineering counterparts. Static models can also be used for analysis purposes, to determine in advance things like system complexity, the time to perform tasks or to master an interface.
There are examples of how static models can be used to generate part of the user interface code (chapter 8), but it was concluded that, in practice, it is more efficient to use script

languages and graphical design environments.

Static models tend to be less flexible with respect to different types of users. Competence models like ETAG often assume perfectly knowing users and identify only one "proper" organisation of the user's task knowledge. Performance models like GOMS often assume expert task performance and lack context-sensitive facilities for planning.

Static models are not very good concerning relations to the "real world". Designers have to feed their models with things like performance time predictions and psychological meanings ("up" is the opposite of "down"). Performance models lack facilities to address affordances and real-world constraints on actions, and both competence and performance models are blind to so-called "knowledge-in-the-world" such as the names of labels, information on what is on the display, etc. Payne et al. (1990), Payne (1991) and May et al. (1995) investigated the structure of information on the visual display screen and the problem of mapping between problem space, device and display, but, for the time being, it seems safe to say that for analysis purposes static models are still blind. Static design models allow the linking of concepts to perceptual elements but not in ways that allow designers to easily mix conceptual and visual design.

Dynamic task and user models like Act-R (Anderson, 1983) and Soar (Laird et al., 1987) are excellent for studying the development of knowledge, and to a smaller extent for studying the influence of external, i.e. visual information on task performance. Both Soar and Act-R are used to simulate user behaviour. For example, Soar was used to create a Programmable User model (PUM; Young et al., 1989) as a tool to help designers dynamically analyse problems in the interaction between the user and the system.

Two problems are connected to the use of dynamic models. First, in order to simulate meaningful behaviour, the modeller is required to specify things at a very low level of detail. Secondly, most dynamic models are created for research purposes and not for use in design. The original aim of the Programmable User Model (PUM; Young et al., 1989) was the creation of a "dummy" user to track down dynamic interaction problems as early as possible during the design of a user interface. Unfortunately, the original purpose has been watered down to the possibility that creating the Soar specification may help designers to recognise and learn about the interaction problems of the user interface at stake (Blandford and Young, 1995). To apply Soar to study the effects of displayed information on learning performance, for example, it is the modeller who has to provide the initial settings and specify how important features like font, colour, word spacing, etc. will be to the simulated user. In this respect, dynamic models are information representation tools that are able to learn rather than toolkits to create user simulations. In other words, they are researcher's tools and not developer's tools.

Dynamic representations are important because they address two main weaknesses in static models. To analyse user interface problems, static models have to make use of general and abstract principles underlying good and bad ways to organise the knowledge and behaviour that is necessary to operate the computer system. A good interface is one that allows users to organise their knowledge and actions consistently or coherently. Unfortunately, and apart from the fact that so-called modern interfaces generally suffer from grotesque inconsistencies, such principles only apply to users who have fairly complete knowledge of the interface or who have mastered a fairly complete repertoire of actions, and such users are a rare species. Dynamic user and task models also allow conclusions about interface problems when users'

knowledge is not complete.
The second weakness that dynamic models address is that in addition to using general and abstract principles to predict and analyse interface problems, dynamic models also allow one to address specific and concrete factors, such as, the development of knowledge and performance from an incomplete and possibly fragmentary state by learning and doing, and the influence of external factors like the information on the display unit, instructions and the observation of the effects of actions.

Described in this way, the problem is that static models are not able to deal with perceptual and performance aspects whereas dynamic models are too detailed and complicated to be useful in user interface design.
First, on the basis of this observation and with respect to the importance of perceptual features it is proposed that research aims to extend both the static and the dynamic models with perceptual information. With respect to static models like ETAG, the proposal is similar to the conclusion of the previous section about the specification of the presentation interface. Given that such a specification of the presentation interface is available it should be possible, in principle, to analyse user interfaces in terms of perceptual-conceptual alignment and establish the degree to which the conceptual and presentation aspects mutually support each other. Note that it is not necessary to find a complete solution to the problem of developing a grammar for perceptual information; a coarse specification may be sufficient. With respect to dynamic models, the same or a similar type of specification of the presentation interface could then be used to analyse and predict the development of the user's knowledge and skills.
Secondly, it is necessary to investigate ways to bridge the gap between static and dynamic models. Static models like ETAG may be extended with interpreters which act in a much simplified form like genuine users. This idea is similar to how, for example, the Soar architecture interprets the information that is specified in Soar programs in a humanoid way, except that the proposal should aim, first and foremost, at creating tools which are usable for design purposes and not as vehicles for psychological research. Dynamic models may become more useful for design purposes by crudifying them down to a level at which they understand ETAG-alike specifications or the output of programs to analyse, for example, the structure and complexity of the information presented on a computer screen.

Provided it is possible to model the conceptual knowledge that is required to perform tasks with a given user interface, as exemplified by static models like ETAG (this thesis), and provided it is possible to model the perceptual information that a user interface presents to the user, either from the design specification or from the actual information from the presentation interface, as exemplified in models to describe the structure of visual information (May et al., 1995), and provided it is possible to build software systems that behave like human users, as exemplified in Act-R (Anderson, 1983) and Soar (Laird et al., 1987) applications, building a simple, usable and useful "simplePUM" tool to analyse the static and dynamic aspects of user interface design specifications should come down to a mere engineering problem. Provided, of course, that a good design specification is created first.

References

When summing up forms of excretion, an important one is generally forgotten: the excretion of offspring. On average the human being excretes each year one kilo of offspring. I must admit, it is not much, but nevertheless. The curious thing about excretion of offspring is that it goes on after death. That one kilo each year remains a kilo each year. In the course of centuries the amount of offspring piles up to enormous quantities.

Jansen, T. (1997).

Agyris, C. (1957). Personality and Organization. Haper and Brothers.

Anderson, J.R. (1983). The Architecture of Cognition. Harvard University Press.

Annett, J. and Duncan, K.D. (1967). Task Analysis and Training Design. Journal of Occupational Psychology 41, 211-212.

Aristotle (ca. 340 bc), Translation by Creed, J.L. and Wardman, A.E., in: Bambrough, R. (1963). The Philosophy of Artistotle. The New English Library.

Arnold, A.G. and Roe, R.A. (1987). User errors in Human-Computer Interaction. In: Frese, M., Ulich, E. and Dzida, W. (eds.) Psychological Issues of Human Computer Interaction in the Work Place. North-Holland.

Ashworth, C. and Goodland, M. (1990) SSADM: A Practical Approach. McGraw-Hill.

Attema, J. and van der Veer, G.C. (1992). Design of the Currency Exchange Interface: Task-Action Grammar used to check consistency. Zeitschrift für Psychology 200, 121-134.

Backus, J.W., Bauer, F.L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A.J., Rutishauer, H., Samelson, K., Vauquois, B., Wegstein, J.H., van Wijngaarden, A. and Woodger, M. (1964). Revised Report on the algorithmic language Algol 60. A/S Regnecentralen, Copenhagen.

Bakker, G.W.A and van Wetering, M.W. (1991). A Dynamic Help-system Based on the Execution of an ETAG-description. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

Balzert, H. (1995). From OOA to GUI: The Janus System. In: Nordby, K., Helmersen, P., Gilmore, D.J. and Arnesen, S. (eds.) Proceedings INTERACT'95, pp. 319-324. Chapman and Hall.

Bannon, L.J. and Bödker, S. (1991). Beyond the Interface: Encountering Artifacts in Use. In: Carroll, J.M. (ed.) Designing Interaction: Psychology at the Human-Computer Interface, pp. 227-253. Cambridge University Press.

Barnard, P. and Harrison, M. (1989). Integrating Cognitive and System Models in Human Computer Interaction. In: Sutcliffe, A. and Macaulay, L. (eds.) Proceedings People and Computers V, pp. 87-103. Cambridge University Press.

Barnard, P., Ellis, J. and MacLean, A. (1989). Relating Ideal and Non-Ideal Verbalised Knowledge to Performance. In: Sutcliffe, A. and Macaulay, L. (eds.) Proceedings People and Computers V, pp. 461-473. Cambridge University Press.

Bayman, P. and Mayer, R.E. (1984). Instructional Manipulation of Users' Mental Models for Electronic Calculators. Int. Journal of Man-Machine Studies 20(2), 189-199.

Bekker, M.M. and Vermeeren, A.P.O.S. (1996). An Analysis of User Interface Design Projects: Information Sources and Constraints in Design. Interacting with Computers 8(1), 112-116.

Benyon, D., Green, T.R.G. and Bental, D. (1999). Conceptual Modeling for User Interface Development. Springer Verlag.
Bewley, W.L., Roberts, T.L., Schroit, D. and Verplank, W.L. (1983). Human Factors Testing in the Design of the Xerox's 8010 'Star' Office Workstation. Proceedings CHI'83, pp. 72-77. ACM Press.
Bias, R.G. and Mayhew, D.J. (eds.)(1994). Cost-Justifying Usability. Academic Press.
Blandford, A.E. and Young, R.M. (1995). Separating User and Device Descriptions for Modelling Interactive Problem Solving. In: Nordby, K., Helmersen, P., Gilmore, D.J. and Arnesen, S. (eds.) Proceedings INTERACT'95, pp 91-96. Chapman and Hall.
Blomberg, J.L. (1995). Ethnography: Alinging Field Studies of Work and System Design. In: Monk, A.F. and Gilbert, G.N. (eds.) Perspectives on HCI - Diverse Approaches, pp. 175-198. Academic Press.
Blyth, A. (1996). Responsibility Modelling and its Application to the Specification of Domain Knowledge. In: Sutcliffe, A., Benyon, D. and van Assche, F. (eds.) Domain Knowledge for Interactive System Design, pp. 48-57. Chapman and Hall.
Bodart, F., Hennebert, A.M., Leheureux, J.M., Provot, I. and Vanderdonckt, J. (1994). A Model-Based Approach to Presentation: A continuum from task analysis to prototype. In Proceedings Design, Specification, Verification of Interactive Systems '94, pp. 25-39. Spinger Verlag, Berlin, 1995.
Boehm, B.W. (1988). A Spiral Model of Software Development and Enhancement. IEEE Computer 21(1), 61-72.
Bomsdorf, B. and Szwillus, G. (1999). Tool Support for Task-Based User Interface Design. ACM CHI'99 workshop report. SigCHI Bulletin 31(4), 27-29.
Boole, G. (1854). An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities. Dover Publications, 1990.
Brazier, F.M.T. (1991). Design and Evaluation of a User Interface for Information Retrieval. Doctoral Thesis, Vrije Universiteit, Amsterdam.
Briggs, P. (1988). What We Know and What We Need to Know: The User Model Versus the User's Model in Human-Computer Interaction, Behaviour and Information Technology 7(4), 431-442.
Briggs, P. (1990). Do They Know What They're Doing? An evaluation of word-processor users' implicit and explicit task-relevant knowledge, and its role in self-directed learning. Int. Journal of Man-Machine Studies 32(4), 385-398.
Brooks, R.E. (1977). Towards a Theory of the Cognitive Processes in Computer Programming. Int. Journal of Man-Machine Studies 9(6), 737-751.
Broos, D. (1989). ETAG-based help: Generating Human Readable Explanations from a Formal Description of the User Interface. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.
de Bruin, H., Bouwman, P. and van den Bos, J. (1994). A Task-oriented Methodology for the Development of Interactive Systems as Used in DIGIS. In: Tauber, M.J., Traunmüller, R. and Kaplan, S. (eds.) Proceedings Interdisciplinary Approaches to System Analysis and Design. Schärding, Austria, 24-26 May 1994.
Butler, K.A., Esposito, C., and Hebron, R. (1999). Connecting the Design of Software to the Design of Work. Communications of the ACM 42(1), 38-46.
den Buurman, R., Leebeek, H.J., Lenior, T.M.J., Scoltens, S., Verhagen, L.H.J.M. and Vrins (1985). Beeldschermergonomie. Nederlandse Verenigning voor Ergonomie, Amsterdam.
Cakir, A., Hart, D.J. and Stewart, T.F.M. (1980). Visual Display Terminals. John Wiley.

Card, S.K., Moran, T.P. and Newell, A. (1983). The Psychology of Human-Computer Interaction. Lawrence Erlbaum.

Carroll, J.M. (ed.)(1987). Interfacing Thought: Cognitive aspects of Human-Computer Interaction. A Bradford Book, MIT Press.

Carroll, J.M.(ed.)(1991). Designing Interaction: Psychology at the Human-Computer Interface. Cambridge University Press.

Carroll, J.M. (1995). Artifacts and Scenarios: An Engineering Approach. In: Monk, A.F. and Gilbert, G.N. (eds.), Perspectives on HCI - Diverse Approaches, pp. 121-144. Academic Press.

CCTA (1989). Central Computers and Telecommunications Agency. Information Technology Infrastructure Library (ITIL), 1989-1998.

Chomsky, N. (1965). Aspects of the Theory of Syntax. MIT Press, Cambridge, MA.

Coad, P. and Yourdon, E. (1990). Object-Oriented Analysis. Prentice-Hall.

Cockayne, A., Wright, P.C. and Fields, B. (1999). Supporting Interaction Strategies Through the Externalization of Strategy Concepts. In: Sasse, M.A. and Johnson, C. (eds.) Proceedings INTERACT'99, pp. 582-588. IOP Press.

Collins, A.M. and Quillian, M.R. (1972) How To Make A Language User. In: Tulving, E. and Donaldson, W. (eds.) Organization of Memory, pp. 141-398. Academic Press.

CORBA (1999). Object Management Group. Common Object Request Broker Architecture, CORBA specification, version 2.3, 1997-1999; www.omg.org/library/specindx.html

Coutaz, J. (1987). PAC, An Object-Oriented Model for Dialogue Design. In: In: Bullinger, H.J. and Shackel, B. (eds.), Proceedings INTERACT'87, pp. 431-436. North-Holland.

Curtis, B. (1988a). The Impact of individual differences in programmers. In: van der Veer, G.C., Green, T.R.G., Hoc, J.M. and Murray, D.M. (eds.) Working with Computers: Theory versus Outcomepp. 279-294. Academic Press.

Curtis, B. (1988). Five Paradigms in the Psychology of Programming. In: Helander, M. (ed.), Handbook of Human-Computer Interaction, pp. 87-105. North-Holland.

Davis, W.S. (1983). Systems Analysis and Design: a structured approach. Addison-Wesley.

DCOM (1998). Microsoft Corporation and Digital Equipment Corporation. Distributed Component Object Model specification, DCOM version1.0, 1998; Object Component Model specification, COM version 0.9, 1995.

Descartes, R. (1637). Discours de la Méthode. Translation: Sutcliffe, F.E. (1968). Discourse on Method and the Meditations. Penguin Books.

Diaper, D. (1989a). Bridging the Gulf between Requirements and Design. In: Proceedings Simulation in the Development of User-Interfaces, Brighton, UK., 18-19 May 1989, pp. 129-145.

Diaper, D. (ed.)(1989b). Task Analysis for Human-Computer Interaction. Ellis Horwood.

Diaper, D. (1989c) Task Analysis for Knowledge Descriptions (TAKD): the method and an example. In: Diaper, D. (ed.). Task Analysis for Human-Computer Interaction, pp. 108-159. Ellis Horwood.

Dijkstra, E.W. (1976). A Discipline of Programming. Prentice Hall.

Dourish, P. (1995). Accounting for System Behaviour: Representation, Reflection and Resourceful Action, In: Proceedings Computers in Context CIC'95. Aarhus, Denmark, August 1995. EuroPARC Technical Report EPC-95-101, EuroPARC, Cambridge, UK.

Eason, K.D. (1988). Information Technology amd Organizational Change. Taylor and Francis.

Eason, K.D. (1990). New Systems Implementation. In: Wilson, J.R. and Corlett, E.N (eds.)

Evaluation of Human Work, pp. 835-849. Taylor and Francis.
Edmondson, D. and Johnson, P. (1989). Detail: An Approach to Task Analysis. In: Proceedings Simulation in the Development of User Interfaces. Norfolk Resort Hotel, Brighton, UK., 18-19 May 1989.
Evans, J.St.B.T. (ed.)(1983). Thinking and Reasoning - Psychological Approaches. Routledge and Kegan Paul.
Fokke, M.J. (1990). An Interactive Question-Answering System Based on a Formal Description of a User Interface. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.
Foley, J.D. and Sukaviriya, P. (1994). History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Implementation. In: Paternó, F. (ed.) Interactive Systems: Design, Specification, and Verification '94, pp. 3-13. Springer Verlag, 1995.
Foltz, P.W., Davies, S.E., and Polson, P.G. (1988). Transfer Between Menu Systems. Proceedings CHI'88, pp. 107-112. ACM Press.
Forbrig, P. and Schlungbaum, E. (1999). Model-based Approaches to the Development of Interactive Systems. Cognitive Systems, Special issue on Cognitive Modeling and User-Interface Development 5(3), 211-224.
Foss D.J. and Hakes, D.T. (1978). Psycholinguistics. Prentice-Hall.
Fröhlich, D.M. and Luff, P. (1989). Some Lessons From an Exercise in Specification. Human Computer Interaction 4(2), 101-123.
Gaines, B.R. and Shaw, M.L.G. (1986). From Timesharing to the Sixth Generation: The Development of Human-Computer Interaction. Int. Journal of Man-Machine Studies 24(1), 1-27.
Geldof, S., Van de Velde, W. (1997). An Architecture for Template Based (Hyper)text Generation. In: Proceedings the 6th European Workshop on Natural Language Generation - EWNLG'97, Duisburg, Germany, pp. 28-37.
Gilmore, D.J., and Green, T.R.G., (1984). Comprehension and Recall of Miniature Programs, Int. Journal of Man-Machine Studies 21(1), 31-48.
Go, N.T. (1992) De Natural Language Front-end van een op ETAG Gebaseerde Helpmachine. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.
Goei, S.L. (1994). Mental Models and Problem Solving in the Domain of Computer Numerically Controlled Programming. Doctoral Thesis, Twente University of Technology, Enschede, The Netherlands.
Golightly, D. and Gilmore, D.J. (1997). Breaking the Rules of Direct Manipulation. In: Howard, S., Hammond, J., and Lindgaard, G.K. (eds.) Proceedings INTERACT'97, pp. 156-163. Chapman and Hall.
Good, M., Spine, T.M., Whiteside, J. and George, P. (1986). User-Derived Impact Analysis as a Tool for Usability Engineering. Proceedings CHI'86, pp. 241-246. ACM Press.
Goudsmit, A. (1993). Task Analysis: Analysing, Comparing and Applying. Master Thesis, Vrije Universiteit, Amsterdam.
Gould, J.D. and Lewis, C. (1985). Designing for Usability: key principles and what designers think. Communications of the ACM 28(3), 300-311.
Greatbatch, D., Heath, C., Luff, P. and Campion, P. (1995). Conversation Analysis: human-computer interaction and the general practice consultation. In: Monk, A.F. and Gilbert, G.N. (eds.) Perspectives on HCI - Diverse Approaches, pp. 199-222. Academic Press.
Green, T.R.G. (1990). Limited Theories as a Framework for Human-Computer Interaction.

In: Ackerman, D. and Tauber, M.J. (eds.) Mental Models and Human-Computer Interaction 1, pp. 3-39. North-Holland.

Green, T.R.G., Davies, S.P. and Gilmore, D.J. (1996). Delivering Cognitive Psychology to HCI: the problems of common language and of knowledge transfer. Interacting with Computers 8(1), 89-111.

Green, T.R.G., Schiele, F., and Payne, S.J. (1988). Formalisable Models of User Knowledge in Human-Computer Interaction. In: van der Veer, G.C., Green, T.R.G., Hoc, J.M. and Murray, D.M. (eds.) Working with Computers: theory versus outcome, pp. 3-46. Academic Press.

Green, T.R.G., Sime, M.E. and Fitter, M.J. (1980). The Problems The Programmer Faces. Ergonomics 23, 893-907.

Green, T.R.G. and Benyon, D. (1995). Displays as Data Structures: entity-relationship models of information artefacts. In: Nordby, K., Helmersen, P., Gilmore, D.J. and Arnesen, S. (eds.) Proceedings INTERACT'95, pp 55-60. Chapman and Hall.

de Groot, A.D. (1966). Perception and Memory versus Thought. In Kleinmuntz, B. (ed.) Problem Solving: research, method and theory. John Wiley.

Guest, S.P. (1982). The Use of Software Tools for Dialogue Design. Int. Journal of Man-Machine Studies 16(3), 263-285.

Guindon, R. and Curtis, B. (1988). Control of Cognitive Processes during Software Design: what tools are needed? Proceedings CHI'88, pp. 263-268. ACM Press.

Gulliksen, J., Lantz, A. and Boivie, I. (1999). User Centered Design - Problems and Possibilities. A Summary of the 1998 PDC and CSCW workshop. SigCHI Bulletin 31(2), 25-35.

Guthrie, W. (1995). An Overview of Portable GUI Software. SigCHI Bulletin 27(1), 55-69. Also available at rtfm.mit.edu as /pub/usenet/comp.windows.misc/pigui.faq.

de Haan, G. (1993). Formal Representation of Human-Computer Interaction. In: van der Veer, G.C., White, T.N. and Arnold, A.G. (eds.) Proceedings Human-Computer Interaction: preparing for the nineties, pp. 95-112. SIC, Amsterdam.

de Haan, G. (1994). An ETAG-based Approach to the Design of User-interfaces. In: Tauber, M.J., Traunmüller, R. and Kaplan, S. (eds.) Proceedings Interdisciplinary Approaches to System Analysis and Design. Schärding, Austria, 24-26 May 1994.

de Haan, G. (1995). The Psychological Basis of a Formal Model for User Interface Design. Unpublished manuscript.

de Haan, G. (1996). ETAG-Based Design: User Interface Design as User Mental Model Design. In: Palanque, P. and Benyon, D. (eds.) Critical issues in User Interface Systems Engineering, pp. 81-92. Springer Verlag.

de Haan, G. (1997). How to Cook ETAG and Related Dishes: Uses of a Notational Language for User Knowledge Representation for User Interface Design. Cognitive Systems 4(3-4), 353-379. From: Stary, C. (ed.) Proceedings of the First Interdisciplinary Workshop on Cognitive Modeling and User Interface Development. Vienna, Austria, 15-17 December 1994.

de Haan, G. (1998). The Politics of Information and Knowledge Sharing for Systems Management. In: Green, T.R.G., Bannon, L., Warren, C.P. and Buckley, J. (eds.) Proceedings the Ninth European Conference on Cognitive Ergonomics, ECCE-9. Limerick, Ireland, 24-26 August 1998.

de Haan, G. (1999a). The Design of a Information Infrastructure to Support System Managers and Business Procedures: how to catch a guru with quality. Cognitive Systems 5(3), 291-

302. From: Proceedings the Second International Workshop on Cognitive Modeling and User Interface Development. Freiburg, Germany, 15-17 December 1997.

de Haan, G. (1999b). The Usability of Interacting with the Virtual and the Real in COMRIS. In: Nijholt, A., Donk, O., and Van Dijk, D. (eds.) Proceedings Interactions in Virtual Worlds, TWLT 15, Enschede, The Netherlands, 19-21 May 1999.

de Haan, G. (2000). Presenting Spoken Advice: Information Pull or Push? Adjunct Proceedings CHI'2000, pp. 139-140. ACM Press.

de Haan, G. (submitted). Interacting with a Personal Wearable Device: getting disturbed and how to control it. Submitted to ECCE-10, Linköping, Sweden, 21-23 August 2000.

de Haan, G. and Muradin, N. (1992). A Case Study on Applying Extended Task-Action Grammar (ETAG) to The Design of a Human-Computer Interface. Zeitschrift für Psychology 200(2), 135-156.

de Haan, G. and van der Veer, G.C. (1992a). Analyzing User Interfaces: ETAG validation studies. Proceedings Task-Analysis in Human-Computer Interaction. Schärding, Austria, 9-11 June 1992.

de Haan, G. and van der Veer, G.C. (1992b). Etag as the Basis for Intelligent Help Systems. In: van der Veer, G.C., Tauber, M.J. and Bagnara, S. (eds.) Proceedings Human-Computer Interaction: tasks and organisation, ECCE-6, pp. 271-283. CUD, Rome.

de Haan, G., van der Veer, G.C. and van Vliet, J.C. (1991) Formal Modelling Techniques in Human-Computer Interaction. Acta Psychologica, 78, 26-76. Also in: Van der Veer, G.C., Bagnara, S. and Kempen, G.A.M. (1992)(eds.) Cognitive Ergonomics: contributions from experimental psychology, pp. 27-68. North-Holland.

Hartson, H.R. and Hix, D. (1989). Human-Computer Interface Development: concepts and systems for its management. Computing Surveys 21(1), 5-92.

Hewett, T.T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G. and Verplank, W. (1992). ACM SigCHI Curricula for Human-Computer Interaction. ACM Press.

Hice, G.F., Turner, W.S. and Cashwell, L.F. (1974). System Development Methodology. North-Holland.

Hix, D. and Hartson, R. (1993). Developing User Interfaces: ensuring usability through product and process, John Wiley.

Holst, S.J., Churchill, E.F. and Gilmore, D.J. (1997). Balloons, Boats and Ponies: interface manipulation style and learning in a constraint-based planning task. In: Howard, S., Hammond, J., and Lindgaard, G.K. (eds.) Proceedings INTERACT'97, pp. 180-187. Chapman and Hall.

van Hoof, M.P.J. (1992). A Question-Answering System Based on ETAG: the question input module. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

Howes, A. and Payne, S.J. (1990) Semantic Analysis During Exploratory Learning. Proceedings CHI'90, pp. 399-405. ACM Press.

Howes, A. (1995) An Introduction to Cognitive Modelling in Human-Computer Interaction. In: Monk, A.F and Gilbert, N. (eds) Perspectives on HCI: diverse approaches. Academic Press, pp. 97-119.

IEEE (1996). IEEE Guide for Developing System Requirements Specifications. ANSI Approved: 1996-04-17, ANSI/IEEE document number IEEE 1233-1996.

Innocent, P.R., Tauber, M.J., van der Veer, G.C., Guest, S., Haselager, E.G., McDaid, E.G., Oestreicher, L., Waern, Y. (1988). Representation Of The User Interface: comparison of

descriptions of interfaces from a designers point of view. In: Speth (ed.) Research into Networks and Distributed Applications, pp. 345-359. North-Holland.
ISO (1987). ISO Standards 9000-9004: Quality Management and quality assurance standards - guidelines (9000), Quality Systems - models (9001-9003), Quality management and quality system elements - guidelines (9004).
Jacobson, I. (1995). Object-Oriented Software Engineering: a use case driven approach. Revised 4th print, ACM Press.
Jacobson, I. Booch, G. and Rumbaugh, J. (1999). The Unified Software Development Process. Addison-Wesley.
Jackendoff, R. (1983). Semantics and Cognition. MIT Press.
Jackendoff, R. (1985). Consciousness and the Computational Mind. MIT Press.
Jensen, K. and Wirth, N. (1974). Pascal User Manual and Report. Springer Verlag.
John, B. E. and Kieras, D. E. (1994) The GOMS Family of Analysis Techniques: tools for design and evaluation. Carnegie Mellon University School of Computer Science Technical Report No. CMU-CS-94-181.
Johnson, H. and Johnson, P. (1991). Task Knowledge Structures: psychological basis and integration into system design. Acta Psychologica 78(1-3), 3-26.
Johnson, P. (1992). Human Computer Interaction: psychology, task analysis and software engineering. McGraw-Hill, New Jersey.
Johnson, P. (1989). Supporting system design by analysing current task knowledge. In: Diaper, D. (ed.) Task Analysis for Human-Computer Interaction, pp. 160-185. Ellis Horwood.
Johnson. J., Roberts, T.L., Verplank, W., Smith, D.C., Irby. C., Beard, M. and Mackey, K. (1989). The Xerox "Star": a retrospective. IEEE Computer 22(9), 11-29. Also in: Baecker, R.M., Grudin, J., Buxton, W.A.S. and Greenberg, S. (eds.)(1995) Readings in Human-Computer Interaction: toward the year 2000, pp. 53-70. Morgan Kaufmann.
Johnson, P., Johnson, H., Waddington, R. and Shouls, A. (1988). Task-Related Knowledge Structures: analysis, modelling and application. In: Jones, D.M. and Winder, R. (eds.) Proceedings People and Computers IV, pp. 35-62. Cambridge University Press.
Johnson, S.C. (1975). YACC: Yet Another Compiler-Compiler. Computer Science Technical Report No. 32. Bell Telephone Laboratories, Murray Hill, NJ.
Kaindl, H. (1998). Combining Goals and Functional Requirements in a Scenario-based Design Process. In: Johnson, H., Nigay, L. and Roast, C. (eds.) Proceedings People and Computers XIII, pp. 101-121. Springer Verlag.
Kant, I. (1781). Kritik der Reinen Vernunft. Translation: Müller, F.M. (ed.)(1881) Critique of Pure Reason. Anchor Books, 1966.
Kemp, J.A.M. and van Gelderen, T. (1996). Co-discovery Exploration: an informal method for the iterative design of consumer products. In: Jordan, P.W., Thomas, B., Weerdmeester, B.A. and McClelland, I.L. (eds.) Usability Evaluation in Industry, pp. 139-146. Taylor and Francis.
Kernighan, B.W., and Plauger, P.J. (1974). The Elements of Programming Style. McGraw-Hill.
Kieras, D. and Polson, P.G. (1985). An Approach to the Formal Analysis of User Complexity. Int. Journal of Man- Machine Studies 22(4), 365-394.
Kieras, D. (1988). Towards a Practical GOMS Model Methodology for User Interface Design. In: M. Helander (ed.) Handbook of Human-Computer Interaction. North-Holland.
Klix, F. (1984a). Über Wissensrepräsentation im Menschlichem Gedächtnis. In Klix, F.

(ed.)(1984b) Gedächtnis, Wissen, Wissensnutzung. VEB, Berlin, pp. 9-73.
Klix, F. (ed.)(1984b). Gedächtnis, Wissen, Wissensnutzung. VEB, Berlin.
Klix, F. (1986). Memory Research and Knowledge Engineering. In Klix, F. and Wandke, H. (eds.) Proceedings MACINTER I, pp. 97-116. North-Holland.
Klix, F. (1989). Concepts, Inference and Cognitive Learning: towards a model of human active memory. In: Klix, F., Streitz, N.A., Waern, Y. and Wandke, H. (eds.) Proceedings MACINTER II, pp. 321-336. North-Holland.
Knowles, C. (1988). Can Cognitive Complexity Theory (CCT) Produce an Adequate Measure of System Usability? In: Jones, D.M. and Winder, R. (eds.) Proceedings People and Computers IV, pp. 291-307. Cambridge University Press.
Knuth, D.E. (1968) The Art of Computer Programming, Volume 1-3. Addison-Wesley.
Korson, T.D. and Vaishnavi, V.K. (eds.)(1992). Analysis and Modelling in Software Development. Special issue of Communications of the ACM 35(9).
Kyng, M. (1994). Scandinavian Design: users in product development. Proceedings CHI'94, pp. 3-9. ACM Press.
Laird, J.E., Newell, A. and Rosenbloom, P.S. (1987). Soar: an architecture for general intelligence. Artificial Intelligence 33(1), 1-64.
Laurel, B. (ed.)(1990). The Art of Human-Computer Interface Design. Addison-Wesley.
Laurel, (ed.)(1991). Computers as Theatre. Addison-Wesley.
Lauridsen, O. (1995). Generation of User Interfaces Using Formal Specification. In: Nordby, K., Helmersen, P., Gilmore, D.J. and Arnesen, S. (eds.) Proceedings INTERACT'95, pp. 325-331. Chapman and Hall.
Leibniz (1686). Traité sur les Perfections de Dieu. Translation: Karskens, M. (ed.)(1981) Metafysische Verhandeling. Wereldvenster, Bussum, The Netherlands.
Lerch F.J., Mantei M.M., Olson J.R. (1989) Skilled Financial Planning: the cost of translating ideas into action. Proceedings CHI'89, pp. 121-126. ACM Press.
Lesk, M.E. (1975). Lex - A Lexical Analyzer Generator. Computer Science Technical Report No. 39. Bell Telephone Laboratories Inc., Murray Hill, NJ.
Lim, K.Y. and Long, J.B. (1994). The MUSE Method for Usability Engineering. Cambridge University Press.
Loucopoulos, P. and Karakostas, V. (1995). System Requirements Engineering. McGraw-Hill.
Löwgren, J. (1995). Applying Design Methodology to Software Development. Proceedings DIS'95, pp. 87-95. ACM Press.
Maguire, H. (1997). Respect User Requirements Framework Handbook. Draft version 2.21. EC Telematics Applications Programme, Project TE 2010 Respect, Wp5 Deliverable D5.1, Husat Research Institute, Loughborough, UK.
Markopoulos, P. and Gikas, S. (1997). Formal Specification of a Task Model and Implications for Interface Design. Cognitive Systems, Special issue on Cognitive Modeling and User-interface Development 4(3-4), 289-310.
Marslen-Wilson, W. (1980). Speech Understanding as a Psychological Process. In: Simon, J.C. (ed.) Spoken Language Generation and Understanding, pp. 39-67. Reidel, Dordrecht, The Netherlands.
Martin, J. (1991). Rapid Application Development. MacMillan Publishing, New York.
May, J., Scott, S. and Barnard, P. (1995). Structuring Displays: a psychological guide. The Amodeus Project, ESPRIT 7040, UM/WP31.
Mayes, J.T, Draper, S.W, McGregor, A.M. and Oatley, K. (1988). Information Flow in a User

Interface: the effect of experience and context on the recall of MacWrite screens. In: Jones, D.M. and Winder, R. (eds.) Proceedings People and Computers IV, pp. 275-289. Cambridge University Press.

McCormick, E.J. (1976). Human Factors in Engineering and Design. McGraw-Hill.

van der Meer, R. (1992). Analysing Two Electronic Mail Systems by ETAG. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

Meyer, B. (1985). On Formalism in Specification. IEEE Software Engineering 2(1), 6-26.

de Michelis, G., Dubois, E., Jarke, M., Matthes, F., Mylopoulos, J., Schmidt, J.W., Woo, C. and Yu, E. (1998). A Three-Faceted View of Information Systems. Communications of the ACM 41(12), 64-70.

Miller, G.A. (1956). The Magical Number Seven, Plus or Minus Two: some limits on our capacity for processing information. The Psychological Review 63, 81-97.

Minsky, M. (1975). Frame-System Theory. In: Schank, R.C. and Nash-Webber, B.L. (eds.) Preprints Theoretical Issues in Natural Language Processing. MIT Press.

Monarchi, D.E. and Puhr, G.I. (1992). A Research Typology for Object-Oriented Analysis and Design. Communications of the ACM 35(9), 35-47.

Monk, A. (1998). Lightweight Techniques to Encourage Innovative User Interface Design. In: Wood, L.E. (ed.) User Interface Design: bridging the gap from user requirements to design, pp. 109-130. CRC Press, Boca Raton, FL.

Monk, A.F. and Gilbert, G.N. (eds.)(1995). Perspectives on HCI - diverse approaches. Academic Press.

Moran, T.P. (1981). The Command Language Grammar: a representation for the user-interface of interactive systems. Int. Journal of Man-Machine Studies 15(1), 3-50.

Moran, T.P. (1983) Getting into the System: external-internal task mapping analysis. Proceedings CHI'83, pp. 45-49. ACM Press.

Muradin, N. (1991). An Assessment Study of ETAG. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

Murray, D.M. (1988). A Survey of User Cognitive Modelling. National Physical Laboratory, NPL Report DITC 92/87.

Myers, B.A. (1994). User Interface Software Tools. Carnegie Mellon University School of Computer Science, Technical report CMU-CS-94-182 (and CMU-HCII-94-107).

Myers, B.A., Giuse, D.A., Dannenberg, R.B., Vander Zanden, B., Kosbie, D.S., Pervin, E., Mickish, A. and Marchal, P. (1990). Garnet: comprehensive support for graphical highly-interactive user interfaces. IEEE Computer 23(11), 71-85.

Nadin, M. (1993). Forum Column: Graphical User Interfaces and Construction Kits, Communications of the ACM 36(12), 15-16.

Nardi, B.A. (ed.)(1996). Context and Consciousness - activity theory and human-computer interaction. MIT Press.

Neerincx, M. and de Greef, P. (1993). How to Aid Non-Experts. In: Ashlund, S., Mullet, K., Henderson, A, Hollnagel, E. and White, T. (eds.) Proceedings INTERCHI'93, pp. 165-171. ACM Press.

Newman, W. A (1993). Preliminary Analysis of the Products of HCI Research, using Pro Forma Abstracts. In: Ashlund, S., Mullet, K., Henderson, A, Hollnagel, E. and White, T. (eds.) Proceedings INTERCHI'93, pp. 278-284. ACM Press.

Nielsen, J. (1986). A Virtual Protocol for Computer-Human Interaction. Int. Journal of Man-Machine Studies 24(3), 301-312.

Nielsen, J. (1990) A Meta-model for Interacting with Computers. Interacting with Computers

2(2), 147-160.
Nigay, L. and Coutaz, J. (1998). Software Architecture Modelling: bridging two worlds using ergonomics and software properties. In: Palanque, P. and Paterno, F. (eds.) Formal Models in Human-Computer Interaction, pp. 49-73. Springer Verlag.
Norman, D.A. (1983). Some Observations on Mental Models. In: Gentner, D. and Stevens, A.L. (eds.) Mental Models, pp. 7-14. Lawrence Erlbaum.
Norman, D.A. (1984). Four Stages of User Activities. In: Shackel, B. (ed.) Proceedings INTERACT'84, pp. 507-511. North-Holland.
Norman, D,.A (1986). Cognitive Engineering. In: Norman, D.A and Draper, S.W. (eds.) User Centered Systems Design, 31-61. Lawrence Erlbaum.
Norman, D.A. (1988). The Psychology of Everyday Things. Basic Books, New York.
Norman, D.A. (1998). The Invisible Computer. MIT Press.
Norman, D.A. and Rumelhart, D.E. and the LNR Research Group (1975). Explorations in Cognition. Freeman.
Oberquelle, H. (1984). On Models and Modelling in Human-Computer Co-operation. In: Van der Veer, G.C., Tauber, M.J., Green, T.R.G. and Gorny, P. (eds.) Readings on Cognitive Ergonomics: mind and computers, pp. 26-43. Springer Verlag.
Olson, A.M. (1993). Object-Oriented Analysis of Visual Computer-Human Interfaces. Dept. of Computer- and Information Science, Indiana University, Indianapolis, manuscript.
Paternò, F., Mancini, C. and Meniconi, S. (1997). ConcurTaskTrees: A Diagrammic Notation for Specifying Task Models. In: Howard, S., Hammond, J., and Lindgaard, G.K. (eds.) Proceedings INTERACT'97, pp. 362-369. Chapman and Hall.
Paulk, M.C., Weber, C.V., Garcia, S.M., Chrissis, M.B. and Bush, M. (1993). Key Practices of the Capability Maturity Model, Version 1.1. Carnegie Mellon University, Software Engineering Institute, Technical Report CMU/SEI-93-TR-25 (and ESC-TR-93-178).
Paulk, M.C., Weber, C.V., Curtis, B.and Chrissis, M.B. (1995). The Capability Maturity Model: guidelines for improving the software process. Addison-Wesley, Reading, MA.
Payne, S.J. (1984) Task action grammar. In: Shackel, B. (ed.) Proceedings INTERACT'84, pp. 139-144. North-Holland.
Payne, S.J. (1987). Complex Problem Spaces: modelling the knowledge needed to use interactive devices. In: Bullinger, H.J. and Shackel, B. (eds.) Proceedings INTERACT'87, pp. 203-208. North-Holland.
Payne, S.J. (1991). Display-based Action At the User Interface. Int. Journal of Man-Machine Studies 35(3), 275-289.
Payne S.J. and Green T.R.G. (1983). The User's Perception Of The Interaction Language: a two-level model. Proceedings CHI'83, pp. 202-206. ACM Press.
Payne, S.J. and Green, T.R.G. (1986). Task-Action Grammars: a model of the mental representation of task languages. Human-Computer Interaction 2(2), 93-133.
Payne, S.J. and Green, T.R.G. (1989). The Structure of Command Languages: an experiment on task-action grammar. Int. Journal of Man-Machine Studies 30(2), 213-234.
Payne, S.J., Squibb, H.R. and Howes, A. (1990). The Nature of Device Models: the yoked state space hypothesis and some experiments with text editors. Human-Computer Interaction 5(4), 415-444.
Perry, D.K. and Cannon, W.M. (1966). A Vocational Interest Scale for Programmers. Proceedings of the 4th Annual Computer Personnel Conference. ACM Press, New York.
Petroski, H. (1996). Design Paradigms: case histories of error and judgment in engineering. Cambridge University Press.

Pfaff, G.E. (ed.)(1985). User Interface Management Systems, Proceedings of the Seeheim Workshop. Springer Verlag.
Phillips, M.D., Bashinski, H.S., Ammerman, H.L. and Fligg, C.M. (1988). A Task Analytic Approach to Dialogue Design. In: Helander, M. (ed.) Handbook of Human-Computer Interaction, pp. 835-857. North-Holland.
Polson, P.G. (1987). A Quantitative Theory of Human-Computer Interaction. In: Carroll, J.M. (ed.) Interfacing Thought: cognitive aspects of human-computer interaction, pp.184-235. MIT Press.
Polson, P.G. (1988). The Consequences of Consistent and Inconsistent User Interfaces. In: Guindon, R. (ed.) Cognitive Science and its Applications for Human-Computer Interaction, pp. 59-108. Lawrence Erlbaum.
Polson P.G. and Lewis, C.H (1990) Theory-based Design for Easily Learned Interfaces. Human-Computer Interaction 5(2-3), 191-220.
Popper, K.R. (1959). The Logic of Scientific Discovery. Basic Books, New York.
Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. and Carey, T. (eds.)(1994). Human-Computer Interaction. Addison-Wesley.
Rassmussen, J. (1987) Cognitive Engineering. In: Bullinger H.J. and Shackel, B. (eds.) Proceedings INTERACT'87, pp. xxv-xxx. North-Holland.
Reisner, P. (1981). Formal Grammar and Human Factors Design of an Interactive Graphics System. IEEE transactions on Software Engineering 7, 229-240.
Reisner, P. (1983). Analytic Tools for Human Factors of Software. In: Blaser, A. and Zoeppritz, M. (eds.) Proceedings Enduser Systems and their Human Factors, pp. 94-121. LNCS 150, Springer Verlag.
Reisner, P. (1984). Formal Grammar as a Tool for Analyzing Ease of Use: some fundamental concepts. In: Thomas, J.C. and Schneider, M.L. (eds.) Human Factors in Computer Systems, pp. 53-78. Ablex Publishing.
Reitman Olson, J. and Olson, G.M. (1990). The Growth of Cognitive Modeling in Human-computer Interaction Since GOMS. Human-Computer Interaction 5(2-3), 221-265.
Richards, J.N.J., Bez, H.E., Gittins, D.T., and Cooke, D.J. (1986). On Methods for Interface Specification and Design. Int. Journal of Man-Machine Studies 24, 545-568.
Roa, A., Carr, L.P., Dambolena, I., Kopp, R.J., Martin, J., Rafii, F. and Schlesinger, P.F. (1996). Total Quality Management: a cross functional persperctive. John Wiley.
Roberts, T.L., and Moran, T.P. (1983). The Evaluation of Text Editors: methodology and empirical results. Communications of the ACM 26(4), 265-283.
Rosenberg, J. and Moran, T.P. (1985). Generic Commands. In: Shackel, B. (ed.) Proceedings INTERACT'84, 245-249. North-Holland.
Roe, R. (1988). Acting Systems Design: an action theoretical approach to the design of man-computer systems. In: de Keyser, V., Qvale, T., Wilpert, B. and Ruiz Quintanilla, S.A. (eds.) The Meaning of Work and Technological Options. John Wiley.
Rohr, G. and Tauber, M.J. (1984). Representational Frameworks and Models for Human-Computer Interfaces. In: Van der Veer, G.C., Tauber, M.J., Green, T.R.G. and Gorny, P. (eds.) Readings on Cognitive Ergonomics: mind and computers, pp. 8-25. Springer Verlag.
Rosch, E. H. (1975) Cognitive representations of semantic categories. Journal of Experimental Psychology: general, 104, 193-233.
Rubinstein, R. and Hersch, H. (1984). The Human Factor: designing computer systems for people. Digital Press, Burlington, MA.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). Object-Oriented Modeling and Design. Prentice-Hall.

Sastry, L. (1994). Report on Graphical User Interface Development Tools. Visualization Group, Informatics Department, DRAL. UK Engineering and Physical Sciences Research Council. Also: http://web.inf.ac.uk/vis/publications/guireport.html.

Scapin, D. and Pierret-Golbreich, C. (1989). Towards a Method for Task Description: MAD. In: Berlinguet, L. and Berthelette, D. (eds.) Work with Display Units 89, pp. 371-380. North-Holland.

Schank, R.C. and Abelson, R. (1977). Scripts, Plans, Goals and Understanding. Lawrence Erlbaum.

Schank, R.C., and Colby, K.M. (eds.)(1973). Computer Models of Thought and Language. Freeman and Co.

Schep, M.S. (1992). Question Answering System Based on the ETAG History Database. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

Schneider, W. (1987). Connectionism: is it a paradigm shift for psychology? Behaviour Research Methods, Instruments, and Computers 19(2), 73-83.

Scholtz, J. and Salvador, T. (1998). Systematic Creativity: a bridge for the gaps in the software development process. In: Wood, L.E. (ed.) User Interface Design: bridging the gap from user requirements to design, pp. 215-244. CRC Press, Boca Raton, FL.

Scholtz, J., Muller, M. Novick, D. Olsen, D.R. Jr., Shneiderman, B., and Wharton, C. (1999). A Research Agenda for Highly Effective Human-Computer Interaction: useful, usable, and universal, Interim Report. SigCHI bulletin 31(4), 13-16.

Sebillotte, S. (1988). Hierarchical Planning as a Method for Task-Analysis: the example of offica task analysis. Behaviour and Information Technology 7(3), 275-293.

Shackel, B. (1986). Ergonomics in Design for Usability. In: Harrison, M.D. and Monk, A.F. (eds.) Proceedings People and Computers II, pp. 44-64. Cambridge University Press.

Shepherd, A. (1989). Analysis and Training in Information Technology Tasks. In: Diaper, D. (ed.) Task Analysis for Human-Computer Interaction, pp. 15-55. Ellis Horwood.

Sheridan, T.B. (1988). Task Allocation and Supervisory Control. In: Helander, M. (ed.) Handbook of Human-Computer Interaction, 159-173. North-Holland.

Shneiderman, B. (1980). Software Psychology: human factors in computer and information systems. Winthrop Publishers.

Shneiderman, B. (1982a). The Future of Interactive Systems and the Emergence of Direct Manipulation. Behaviour and Information Technology 1, 237-256.

Shneiderman, B. (1982b). Multiparty Grammars and Related Features for Defining Interactive Systems. IEEE Transactions on Systems, Man and Cybernetics, 12(2), 93-133.

Shneiderman, B. (1986). Designing the User Interface. Addison-Wesley.

Sime, M.E., Green, T.R.G. and Guest, D.J. (1977). Scope Marking in Computer Conditionals: a psychological evaluation. Int. Journal of Man-Machine Studies 9, 107-118.

Simon, T. (1988). Analysing the Scope of Cognitive Models in Human-Computer Interaction: a trade-off approach. In: Jones, D.M. and Winder, R. (eds.) Proceedings People and Computers IV, pp. 79-93. Cambridge University Press.

Smit, R.G. (1991). On the Translating of ETAG into English. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

Smith, C.D. (1998). Transforming User-Centered Analysis into User Interface: the design of new generation products. In: Wood, L. (ed.) User Interface Design: bridging the gap from user requirements to design, pp. 275-304. CRC Press, Boca Raton, FL.

Smith, D.C., Irby, C., Kimball, R., Verplank, W. and Harslem, E. (1982). Designing the Star User Interface. Byte 7(4), 242-282.
Smith, S.L. and Mosier, J.N. (1984). Design Guidelines for User-System Interface Software. The Mitre Corporation, Bedford, MA.
Smith, E.E., Shoben, E.J. and Rips, L.J. (1974). Structure and Process in Semantic Memory: a feature model for semantic decisions. Psychological Review 81, 214-241.
Sommerville, I. (1985). Software Engineering, 2nd edition. Addison-Wesley.
Sowa, J.F. (1984). Conceptual Structures: information processing in mind and machine. Addison-Wesley.
Sproull, L and Kiesler, S. (1991). Computers, Networks and Work. Scientific American 265(3), 116-123. Reprinted in a special issue 6(1), 1995.
Suchman, L.A. (1987). Plans and Situated Actions: the problem of human-computer communication. Cambridge University Press.
Suchman, L.A. (1995). Making Work Visible. Communications of the ACM, 38(9), 56-64.
Summersgill, A.R. and Browne, A.D.P. (1989). Human Factors: its place in system development methods. Sigsoft Bulletin 14(3), 227-234.
Suppe, F. (1974). The Search for Philosophic Understanding of Scientific Theories. In: Suppe, F. (ed.) The Structure of Scientific Theories, pp. 3-232. University of Illinois Press.
Systä, K., (1994). Specifying User Interfaces in DisCo. SigCHI Bulletin 26(2), pp. 53-58.
Tamboer, E. (1991). An ETAG-based Help-module for PMAIL. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.
Tauber, M.J. (1988). On Mental Models and the User Interface. In: van der Veer, G.C., Green, T.R.G., Hoc, J.M. and Murray, D.M. (eds.) Working with Computers: theory versus outcome, pp. 89-119. Academic Press.
Tauber, M.J. (1990). ETAG: Extended Task Action Grammar: a language for the description of the user's task language. In: Diaper, D., Gilmore, D., Cockton, G., and Shackel, B. (eds.) Proceedings INTERACT'90, pp. 163-168. North-Holland.
Taylor, F.W. (1911). The Principles of Scientific Management. Harper.
Thimbleby, H. (1990). User Interface Design. ACM Press.
UML (1999). Object Management Group, Unified Modeling Language, UML, version 1.3 specifications; www.rational.com/uml/
Vanderdonckt, J.M. and Bodart, F. (1993). Encapsulating Knowledge for Automatic Interaction Objects Selection. In: Ashlund, S., Mullet, K., Henderson, A, Hollnagel, E. and White, T. (eds.) Proceedings INTERCHI'93, pp. 424-429. ACM Press.
van der Veer, G.C. (1990). Human-Computer Interaction: learning, individual differences, and design recommendations. Doctoral Thesis, Vrije Universiteit, Amsterdam.
van der Veer, G.C. and Guest, S. (1992). Fundamental Concepts for Designing Multi-media and Cooperative Technology. In: Tauber, M.J. (ed.). Proceedings Task Analysis for System Design, Austria, 9-11 June 1992.
van der Veer, G.C., Guest, S., Haselager, P., Innocent, P., McDaid, E., Oestreicher, L., Tauber, M., Vos, U. and Waern, Y. (1988). An Interdisciplinary Approach to Human Factors in Telematic Systems. Computer Networks and ISDN Systems 15, 73-80.
van der Veer, G.C., Lenting, B.F. and Bergevoet, B.A.J. (1996). GTA: Groupware Task Analysis - modeling complexity. Acta Psychologica 91, 297-322.
van der Veer, G.C., van Vliet, J.C. and Lenting, B.F. (1995). Designing Complex Systems - a structured activity. In: Olson, G.M. and Schuon, S. (eds.) Proceedings DIS '95, pp. 207-217. ACM Press.

van der Veer, G.C., Yap, F., Broos, D., Donau, K. and Fokke, M.J. (1990). ETAG - some applications of a formal representation of the user interface. In: Diaper, D., Gilmore, D., Cockton, G., and Shackel, B. (eds.) Proceedings INTERACT'90, pp. 169-174. North-Holland.

van der Velde (1997). Co-Habited Mixed Reality. Proceedings the Fifteenth International Joint Conference on Artificial Intelligence, Aichi, Japan, 23-29 August 1997.

Versendaal, L.M. (1991). Separation of the User Interface and Application. Doctoral Thesis, Delft University of Technology, Delft, The Netherlands.

van Vliet, J.C. (1994). Software Engineering - principles and practice. John Wiley.

Vlissides, J.M. and Tang, S. (1991). A Unidraw-Based User Interface Builder. In: Proceedings UIST'91, pp. 201-210. ACM Press.

Vossen, P.H., Sitter, S. and Ziegler, J.E. (1987). An Empirical Validation of Cognitive Complexity Theory. In: Bullinger, H.J. and Shackel, B. (eds.) Proceedings INTERACT'87, pp. 203-208. North-Holland.

Walsh, P (1989). Analysis for Task Object Modelling (ATOM): towards a method of integrating task analysis with Jackson System development for user interface software design. In: Diaper, D. (ed.) Task Analysis for Human-Computer Interaction, pp. 186-209. Ellis Horwood.

Wandke, H. (1992). Models in Human-Computer Interaction: application and verification problems. Zeitschrift für Psychologie 200(2), 105-119.

Wason, P.C. and Johnson-Laird, P.N. (1972). The Psychology of Reasoning: structure and content. Batsford, London, UK.

Weinberg, G.M. (1971). The Psychology of Computer Programming. Van Nostrand Reinhold.

van Welie, M., van der Veer, G.C. and Eliëns, A. (1998a). An Ontology for Task World Models. In: Markopoulos, P. and Johnson, P. (eds.) Proceedings Design, Specification and Verification of Interactive Systems '98, Abingdon, UK., 3-5 June 1998, pp. 57-70. Springer Verlag.

van Welie, M., van der Veer, G.C. and Eliëns, A. (1998b). Euterpe - tool support for analyzing cooperative environments: In: Green, T.R.G., Bannon, L., Warren, C.P. and Buckly, J. (eds.) Proceedings ECCE9 - Cognition and Co-operation, 24-26 August 1998, Limerick, Ireland, pp. 23-30.

Whitefield, A. (1987). Models in Human Computer Interaction: a classification with special reference to their uses in design. In: Bullinger, H.J. and Shackel, B. (eds.) Proceedings INTERACT'87, pp. 57-63. North-Holland.

Wilson, M.D., Barnard, P.J., Green, T.R.G. and MacLean, A. (1988). Knowledge-Based Task Analysis for Human-Computer Systems. In: van der Veer, G.C., Green, T.R.G., Hoc, J.M. and Murray, D.M. (eds.) Working with Computers: theory versus outcome, pp. 47-87. Academic Press.

Wilson, S., Johnson, P., Kelly, C., Cunningham, J. and Markopoulos, P. (1993). Beyond Hacking: a model based approach to user interface design. In: Alty, J.L., Diaper, D. and Guest, S. (eds.) Proceedings People and Computers VIII, pp. 217-232. Cambridge University Press.

Wixon, D., Holtzblatt, K. and Knox, S. (1990). Contextual Design: an emergent view of system design. Proceedings CHI'90, pp. 329-336. ACM Press.

Woudstra, W.M. and Kohli, V (1992). A Frond-end for the Simulator of a User Interface, Using the Formal Description Language ETAG. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

van Wijngaarden, A., Mailloux, B.J., Peck, J.E.L., and Koster, C.H.A. (1969). Report on the Algorithmic Language ALGOL 68. Numer. Math. 14, 79-218.

Yap, F. (1990). Etag Tested on Two Page Tools. Master Thesis, Department of Computer Science, Vrije Universiteit, Amsterdam.

Young, R.M., Green, T.R.G. and Simon, T. (1989). Programmable User Models for Predictive Evaluation of Interface Designs. Proceedings CHI'89, pp. 15-19. ACM Press.

Citations

Acknowledgements: Heller; J. (1989). Something Happened. Laurel Book, Dell Publishers, New York, pg. 319.

Contents: Kelley, J. (1997). In: The Correctness Survey Results: Reality -- Yes or No?. The mini-Annals of Improbable Research, Issue 1997-08, August 1997.

Chapter 1: Boden, M. (1989). Artificial Intelligence in Psychology. Chapter 2: Fashions of Mind. MIT Press, Cambridge, MA.

Chapter 2: Pirsig, R. (1974). Zen and the Art of Motorcycle Maintenance, an inquiry into values. Bantam Books, New York, 1979, pg. 93.

Chapter 3: Wittgenstein, L. (1953). Philosophische Untersuchungen, Part I, no. 496. Translation: Anscombe, G.E.M. (1978), Philosophical Investigations. Basil Blackwell, Oxford, UK.

Chapter 4: Descartes, R. (1637). Discours de la Méthode. Translation: Sutcliffe, F.E. (1968). Discourse on Method and the Meditations, pg. 45. Penguin Books, Harmondsworth, UK.

Chapter 5: Babbage, C. (1835). On the Economy of Machinery and Manufactures, fourth edition. Charles Knight, London, 1835. Reprinted by Frank Cass and co, London, UK., 1963, pg. 191.

Chapter 6: Aristotle (ca. 340 bc), Metaphysics, Book I, section I. Translation: Creed, J.L. and Wardman, A.E., in: Bambrough, R. (1963). The Philosophy of Artistotle, The New English Library, London, UK., pg. 41.

Chapter 7: Ayer, A.J. (1936). Language, Truth and Logic, Penguin Books, pg. 67.

Chapter 8: Böll, H. (1965). Ansichten Eines Clowns - The Clown, The New American Library, New York, 1966, pg. 199.

Chapter 9: de Bie, W. (1988). Schoftentuig, de Harmonie, Amsterdam, pg. 140 (translated GdH).

References: Jansen, T. (1997). Reflection "Gender", De Volkskrant, 8 November 1997.

Summary: Fry, S. (1994), The Hippopotamus. Arrow Books, London, UK., 1995, pg. IX.

Biography: Carmen Maura as Pepu in: Women on the Verge of a Nervous Breakdown. A film by Almodóvar, P., 1988, Spain.

Dutch Summary

I've counted up the words processed, a thing I do every hour, and, if technology can be trusted, it looks as if you're in for 94,536 of them. Good luck to you. You asked for it, you paid me for it, you've got to sit through it. As the man said, I've suffered for my art, now it's your turn.
Fry, S. (1994).

Dit proefschrift gaat over het gebruik van formele specificatiemethoden voor het ontwerpen van user interfaces, en in het bijzonder over het gebruik van Extended Task-Action Grammar (ETAG). Het proefschrift begint met het schetsen van een globale achtergrond en het formuleren van een aantal vragen die het vanuit deze achtergrond tracht te beantwoorden (hoofdstuk 1). De achtergrond van dit proefschrift is dat mensen voor het verrichten van taken met behulp van de gebruikersinterface van een computersysteem kennis moeten hebben van de wijze waarop hun taken (de taakgegevens, de taakstructuur, etc.) volgens het systeem georganiseerd zijn.

Het uitgangspunt van dit proefschrift is het idee dat het ontwerpen van gebruikersinterfaces opgevat kan worden als het specificeren van de kennis die mensen nodig hebben om hun taken succesvol te kunnen verrichten. Door de benodigde kennis te specificeren in de formele taal "ETAG" is het mogelijk om de verschillende aspecten van de gebruiksvriendelijkheid zoals gebruiksgemak, leerbaarheid, en efficiency in een vroeg stadium te berekenen en onderdelen van het ontwerp zoals hulp informatie automatisch te genereren. In dit proefschrift wordt de geldigheid van deze stelling onderzocht.

Het proefschrift geeft een overzicht van de beschikbare formele modellen in de HCI en daaruit wordt ETAG gekozen als de meest veelbelovende kandidaat voor verdere ontwikkeling (hoofdstuk 2). Van ETAG worden het ontstaan en de psychologische achtergrond beschreven, en aan de hand van feiten en theorieën vanuit de psychologie, de linguïstiek en de logica wordt de validiteit van ETAG besproken (hoofdstuk 3). Het beschrijft hoe ETAG representaties gecreëerd kunnen worden (hoofdstuk 6) en het beschrijft een methode "ETAG-Based Design" voor het ontwerpen van user interfaces (hoofdstuk 4). In een aantal hoofdstukken wordt de toepasbaarheid van ETAG geanalyseerd in verschillende ontwerpvragen en ontwerpstadia zoals de taakanalyse (hoofdstuk 5), het analyseren van de bruikbaarheid van user interfaces (hoofdstuk 7), en het automatisch genereren van hulp informatie en user interface prototypes (hoofdstuk 8). Tenslotte wordt (in hoofdstuk 9) het onderzoek samengevat en worden algemene conclusies getrokken.
Daar modellen als ETAG (te) weinig zeggen over de presentatie en het feitelijk gebruik van user interfaces wordt aanbevolen methoden te ontwikkelen voor het beschrijven en analyseren van de perceptuele kenmerken van user interfaces en de dynamische interactie tussen de gebruiker en het user interface.

SIKS Dissertations

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects
Promotor: Prof.dr. M.L. Kersten (CWI/UvA)
Co-promotor: Dr. A.P.J.M. Siebes (CWI)
Promotie: 30 maart 1998

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
Promotores: Prof.dr.ir. A. Hasman (UM)
Prof.dr. H.J. van den Herik (UM/RUL)
Prof.dr.ir. J.L.G. Dietz (TUD)
Promotie: 7 mei 1998

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
Promotores: Prof.dr.ir. J.L.G. Dietz (TUD)
Prof.dr. P.C. Hengeveld (UvA)
Promotie: 22 juni 1998

1998-4 Dennis Breuker (UM)
Memory versus Search in Games
Promotor: Prof.dr. H.J. van den Herik (UM/RUL)
Promotie: 16 oktober 1998

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting
Promotores: Prof.mr. H. Franken
Prof.dr. H.J. van den Herik
Promotie: 13 mei 1998

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
Promotor: Prof.dr. J. Treur
Co-promotor: Dr.ir. M. Willems
Promotie: 11 mei 1999

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
Promotor: Prof. Dr. A. de Bruin
Co-promotor: Dr. J.C. Bioch
Promotie: 4 juni 1999

1999-3 Don Beal (Queen Mary and Westfield College)
The Nature of Minimax Search
Promotor: Prof.dr. H.J.van den Herik
Promotie: 11 juni 1999

1999-4 Jacques Penders (KPN Research)
The practical Art of Moving Physical Objects
Promotor: Prof.dr. H.J. van den Herik

Co-promotor: Dr. P.J. Braspenning
Promotie: 11 juni 1999

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
Promotor: Prof.Dr. R.A. Meersman
Co-promotor: Dr. H. Weigand
Promotie: 1 oktober 1999

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
Promotor: Prof.dr. J. Treur
Co-promotor: Dr. F.M.T. Brazier
Promotie: 30 september 1999

1999-7 David Spelt (UT)
Verification support for object database design
Promotor: Prof. Dr. P.M.G. Apers
Assistent Promotor: Dr. H. Balsters
Promotie: 10 september 1999

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.
Promotor: Prof. dr. H.J. van den Herik
Co-promotor: Dr. P.J. Braspenning
Promotie: 3 december 1999

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance
Promotor: Prof.dr. J.C. van Vliet (VU)
Promotie: 28 maart 2000

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management
Promotores: Prof. dr. P.M.E. De Bra
Prof. dr. R.H. McClatchey
Co-promotor: Dr. P.D.V. van der Stok

XXXXXXX

2000-X Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
Promotor: Prof. dr. J.C. van Vliet
Co-promotor: Dr. G.C. van der Veer
Dr. M.J. Tauber

Biography

Motorcycle maintenance is easier than male psychology.

Maura, C (1988).

I exist as Geert de Haan since 14 May 1959.

I studied Psychology and Philosophy at the University of Leiden, and during the masters in Psychology, I specialised in Human-Computer Interaction. I spent 6 months as an intern at MRC's Applied Psychology Unit in Cambridge where I worked with dr. Thomas Green on Task-Action Grammar (TAG). My master thesis: "Programming in Parallel: A study in the psychology of software" was supervised by prof.dr. Patrick Hudson. I graduated cum laude in Psychology in 1988.

At the unit of Ergonomics of the University of Twente, Enschede, I did PhD research on "Formal Representations in HCI" with prof.dr. Ted White. I continued my PhD research, now on "Extended Task-Action Grammar", at the department of Computer Science of the Vrije Universiteit, Amsterdam, with dr. Gerrit van der Veer and prof.dr. Hans van Vliet. For a year I worked at Origin's TIS - Distributed Systems as a project manager on providing information for systems' management. Since 1998, I work as a researcher at IPO, Centre for User-System Interaction at the Eindhoven University of Technology. The first year, I worked on the usability evaluation of COMRIS, an intelligent wearable personal assistant. The last 6 months, I worked on user modelling and on completing this PhD thesis.

Within HCI, I am interested in methods for user-interface design, cognitive task-analysis, ubiquitous computing and information appliances, and formal models of user-knowledge and performance. I am a more-and-less active member of ACM SigCHI, SigCHI.nl, EACE, and IFIP TC 13.2.

Outside HCI, I like long-distance running, bicycle-camping, Linux, English literature, films, and to think about politics, ethics, and epistemology. I am a bicyclist, a smoker, a vegetarian, a coffee-addict, and a bad sleeper.