

TASK-BASED USER INTERFACE DESIGN

Martijn van Welie



SIKS Dissertation Series No. 2001-6.

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. A. Eliëns (co-promotor)

dr. F. Paternò (CNUCE Italy, referent)

prof.dr. M. Rauterberg (IPO & Technische Universiteit Eindhoven)

prof.dr. J.M. Akkermans (Vrije Universiteit Amsterdam)

prof.dr. B. Wielinga (University of Amsterdam)

VRIJE UNIVERSITEIT

TASK-BASED USER INTERFACE DESIGN

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen \ Wiskunde en Informatica
op dinsdag 17 april 2001 om 13.45 uur
in het hoofdgebouw van de universiteit,
De Boelelaan 1105

door

Martijn van Welie

geboren te Lisse

Promotor: prof.dr. J.C. van Vliet
Copromotoren: dr. G.C. van der Veer
dr. A. Eliëns

Preface

This thesis could not have been written without the help and inspiration of many people around me. At the Vrije Universiteit, I was in an interesting environment with many different ideas. I want to thank Gerrit van der Veer, Anton Eliëns and Hans van Vliet for their enthusiasm and how they enriched my research by showing me their perspectives. Thanks to Gerrit for introducing me to the “world of Human Computer Interaction”. I very much enjoyed our collaboration where we had different approaches to a common interest. Thanks to Anton for “convincing” me to do a Ph.D. in the first place and for showing me the use of several interesting techniques during the development of my tool EUTERPE.

I also want to thank many other people who supported me throughout the last years. A big part of that was due to Bastiaan Schönhage, by being a friend and roommate for many years. We sure had a lot of fun in between the “serious” work. It was very good to have someone like him to discuss “raw” ideas and to keep me from “overdoing” it. During the last months, we were both writing our dissertation which definitely helped me to keep up the spirit and finish it in time.

In addition, thanks to my direct colleagues Nico Lassing, Jacco van Ossenbruggen, Frank Niessink and Jaap Gordijn. During a working day, it is also important to relax and have pointless discussions, for example during lunch when Arno Bakker and Gerco Ballintijn used to join us.

Thanks to Hallvard Trætteberg for his work on the patterns. During the development of the patterns he was my best reviewer and contributor. Without his help I would not have developed them so fast. Joerka Deen also proved to be a good discussion partner, always emphasizing the more “philosophical” side of patterns.

And of course, many thanks to my parents, my sister and Merche for all their support and love during these years.

Contents

1	Introduction	1
1.1	The Task-based Design Process	2
1.1.1	Analyzing the current task situation (Task model 1)	3
1.1.2	Envisioning the future task situation (Task model 2)	4
1.1.3	Detailed design	4
1.1.4	Evaluation and usability testing	4
1.2	Research Goals	5
1.2.1	Improving the design process	5
1.2.2	Improving the final product	5
1.3	Research Approach	6
1.4	Outline of this Thesis	7
1.5	Publications	8
2	Designing for Usability	9
2.1	Introduction	9
2.2	Potential Benefits of Usability	10
2.3	Understanding Usability	11
2.3.1	Understanding humans	11
2.3.2	Understanding the work	12
2.3.3	Understanding the interaction	13
2.4	Definitions of Usability	14
2.5	Heuristics, Guidelines and Principles	16
2.6	A Layered Model of Usability	18
2.7	Usability and Software Quality	20
2.8	Usability Evaluation and Improvement	21

2.8.1	Measuring usability	22
2.8.2	Improving usability	24
2.8.3	Usability process improvement	24
2.9	Usability and Design Methods	25
2.9.1	Modeling humans and work	25
2.9.2	Modeling the system	26
2.10	Summary	27
3	An Ontology for Task Models	29
3.1	Introduction	29
3.2	A Short History of Task Analysis	30
3.3	Methods and techniques	31
3.3.1	Hierarchical Task Analysis (HTA)	31
3.3.2	Goals Operators Methods Selectors (GOMS)	31
3.3.3	Méthode Analytique de Description des tâches (MAD)	34
3.3.4	Groupware Task Analysis	35
3.4	An Ontology for Task World Models	37
3.4.1	Modeling work structure	37
3.4.2	Modeling the work flow	39
3.4.3	Modeling work artifacts	40
3.4.4	Modeling the work environment	40
3.4.5	Defining an ontology	42
3.5	Related Work	45
3.6	Summary	46
4	Task Modeling and Analysis	47
4.1	Introduction	47
4.2	Representations for Task Modeling	48
4.2.1	Common representations	48
4.2.2	A collection of ontology-based representations	55
4.3	Static versus Dynamic Representations	62
4.4	Analyzing the Task World	63
4.4.1	Heuristic model-based evaluation	64
4.4.2	Model verification	65

4.4.3	Comparing two specifications	67
4.4.4	Model validation	68
4.5	Summary	68
5	Detailed Design	71
5.1	Introduction	71
5.2	The Gap between Analysis and Design	72
5.3	Guidelines for Bridging the Gap	72
5.4	Designing the User's Virtual Machine	74
5.4.1	Designing the functionality	75
5.4.2	Designing the dialog	76
5.4.3	Designing the presentation	77
5.5	Cognitive aspects in UVM Design	78
5.6	Specifying the User Interface	79
5.6.1	Informal methods for detailed design	79
5.6.2	Formal specification techniques	81
5.7	NUAN: New User Action Notation	83
5.7.1	Adding an interface pre-state column	84
5.7.2	A modified interface feedback column	85
5.7.3	Expanding time capabilities	85
5.7.4	Mental actions	85
5.7.5	Generic interaction diagrams	86
5.7.6	Parallellism	86
5.8	Evaluating Design Alternatives	86
5.8.1	Scenarios, guidelines, and patterns	87
5.8.2	Prototype evaluation tools	88
5.8.3	Formal usability evaluation of the user interface	89
5.9	Summary	92
6	Interaction Patterns in User Interface Design	93
6.1	Introduction	93
6.2	Guidelines or Patterns?	94
6.3	An Example	95
6.4	Patterns as Design Tools	96

6.4.1	Defining a pattern	97
6.4.2	Anti-patterns	97
6.4.3	Types of patterns	97
6.5	Interaction Design Patterns	98
6.5.1	Taking the user perspective	99
6.5.2	Categorizing user problems	100
6.5.3	A focus on usability	101
6.5.4	A template for interaction patterns	102
6.6	Towards a Pattern Language	105
6.6.1	Structuring a pattern collection	106
6.6.2	Developing a pattern language	107
6.7	Summary	108
7	Tools for Task-based Design	109
7.1	Introduction	109
7.2	User Interface Design Tools	110
7.3	Supporting Task-based Design	110
7.3.1	Support throughout the process	111
7.3.2	Integrated modeling and modeling Purposes	111
7.4	An Overview of Current Task Analysis Tools	112
7.4.1	Commercially available tools	113
7.4.2	Research tools	116
7.5	Requirements for Task Analysis Tools	123
7.5.1	Base the tool directly on a conceptual framework	124
7.5.2	Offer consistent and coherent representations	124
7.5.3	Support cooperative task analysis	124
7.5.4	Support documentation including multimedia	124
7.5.5	Support design tracking	125
7.5.6	Offer stability, robustness and product support	125
7.6	Discussion	125
7.7	Summary	126

8	EUTERPE, a design workbench	127
8.1	Introduction	127
8.2	The Project's Context	128
8.3	An Ontology-based Tool	129
8.4	Supporting Task-based Design	130
8.4.1	Supporting task modeling	131
8.4.2	Supporting model analysis	132
8.4.3	Supporting cooperative design	134
8.4.4	Supporting documentation	134
8.4.5	Supporting dialog modeling	136
8.5	Extending EUTERPE	137
8.5.1	Adding support for design space analysis	137
8.5.2	Adding support for simulation	138
8.6	EUTERPE in Use	139
8.7	Implementation of Euterpe - a logic approach	140
8.7.1	A model-view-controller architecture	141
8.7.2	Embedding a Prolog engine	142
8.7.3	Mapping the ontology to Object Prolog	144
8.7.4	Prolog and application functions	146
8.8	Lessons Learned	146
8.9	Summary	147
9	Task-based Design in Practice	149
9.1	Introduction	149
9.2	Applications in Industry	149
9.2.1	Dutch social security	150
9.2.2	Seibersdorf	151
9.3	Application Issues of Task-based Design	154
9.3.1	Performing task analysis	154
9.3.2	Integration with current design practice	156
9.3.3	Designing for usability	157
9.3.4	Developing design alternatives	158
9.4	Limitations and Critical Success factors	159
9.5	Summary	160

10 Conclusions and Future Research	163
10.1 Summary of the Thesis	163
10.2 Contributions	165
10.3 Future Research	166
A NUAN symbol definitions	169
B Interaction Design Patterns	173
Bibliography	187
Samenvatting	197
Index	203

Chapter 1

Introduction

Every day many people get frustrated with systems that do not adequately support them in their work. Often the software is of high internal quality, but when the system does not match the users' tasks and needs, internal software quality becomes almost irrelevant (Bevan 1999). Users will try to avoid using such systems if they can or may even reject using them at all. Real users have *real* problems trying to do *real* work. Many books such as (Cooper 1999, Norman 1999) are dedicated to describing and analyzing the problems users are nowadays facing when they use interactive systems. Although understanding these problems is very important, they often constitute a "post-mortem" analysis that does not contribute to improvements of the design processes.

The usage problems so often encountered in practice are not only caused by a lack of good design methods. It is simply true that marketing issues and business practices play an important role as inhibitors of delivering usable systems. Perhaps even the end-users are to blame; e.g. who ever asks to see the remote control before buying a TV set? This thesis takes the position that user interface designers should try to deliver usable products, despite the presence of all those inhibiting forces.

As a consequence, user interface design should mature as a discipline that can deliver usable systems under those restricting circumstances. To do so, we need efficient and effective design methods complete with the appropriate techniques and tools that allow us to systematically deliver usable systems. When designing for usability, the focus should be on usability throughout the process and should not be added in the final stages of product development. Therefore, user interface design needs to become more engineering rather than a pure creative or artistic process, focusing on quality in use instead of artistic qualities.

Many methods rely solely on usability testing to guard the usability of the system. Usability testing is very valuable but "trying to get it right the first time" should not be thought of as an illusion. Although incremental development can be very effective, iteration is often done because there is a lack of effective techniques. The more traditional engineering disciplines use proven techniques and knowledge to get it right more

or less the *first* time. There is only room for minor adjustments during the construction process. User interface design often relies too much on trial-and-error techniques. The methods and techniques for user interface design need to be improved in order to systematically develop better systems. Better methods and techniques can improve the quality of the initial design and less iterations will be needed to get the details right.

This thesis describes "*Task-based User Interface Design*" (TB-UID)¹. It is a design method that covers the important aspects from task analysis to prototype evaluation. The main idea behind the method is that a firm understanding of the users' tasks is the proper basis for interactive systems development. With a systematic process of building systems to support users in their tasks, a higher level of usable and useful systems can be achieved by both expert and novice designers.

Task-based design is not the only existing method for designing interactive systems. However, most methods cover only specific aspects or activities in the design process. The method presented in this thesis builds on existing techniques, but it also adds several original insights, tools and techniques. In the next sections, the task-based design process is outlined briefly. All activities in the design process will be discussed in the following chapters. Additionally the main research goals are discussed, followed by an outline of the thesis.

1.1 The Task-based Design Process

Over the past years we have taken useful bits of theories and combined them with our own techniques to form a coherent practical method for designing complex interactive systems, called DUTCH². The method has been used successfully in both industry and education proving the practical value of the method. From our experiences, we learned that for a practical method it is required to a) define a clear process, b) define the models and representations including their semantics and c) support the method and models with tools. In the next chapters these requirements will be discussed in detail. Our design process is task based, which means that it uses the tasks of users as a driving force in the design process. The goal is to design both usable and useful systems. We think it is important to base the design on the work that has to be done by the users. Therefore, the users play an important role in acquiring knowledge about their work as well as for usability testing.

Our process consists of four main activities: (a) analyzing a "current" task situation, (b) envisioning a future task situation for which information technology is to be designed, and (c) specifying the information technology to be designed. In parallel to these activities, (d) evaluation activities make the process cyclic. Figure 1.1 gives an overview of the whole design process with all activities and sources of information. In the next chapters, the four main activities will be described in detail.

The design process starts by an extensive task analysis using our method Groupware Task Analysis (GTA). We distinguish two task models. The first task model we make

¹Throughout this thesis the phrase "task-based design" is also used as a synonym.

²Designing for Users and Tasks from Concepts to Handles

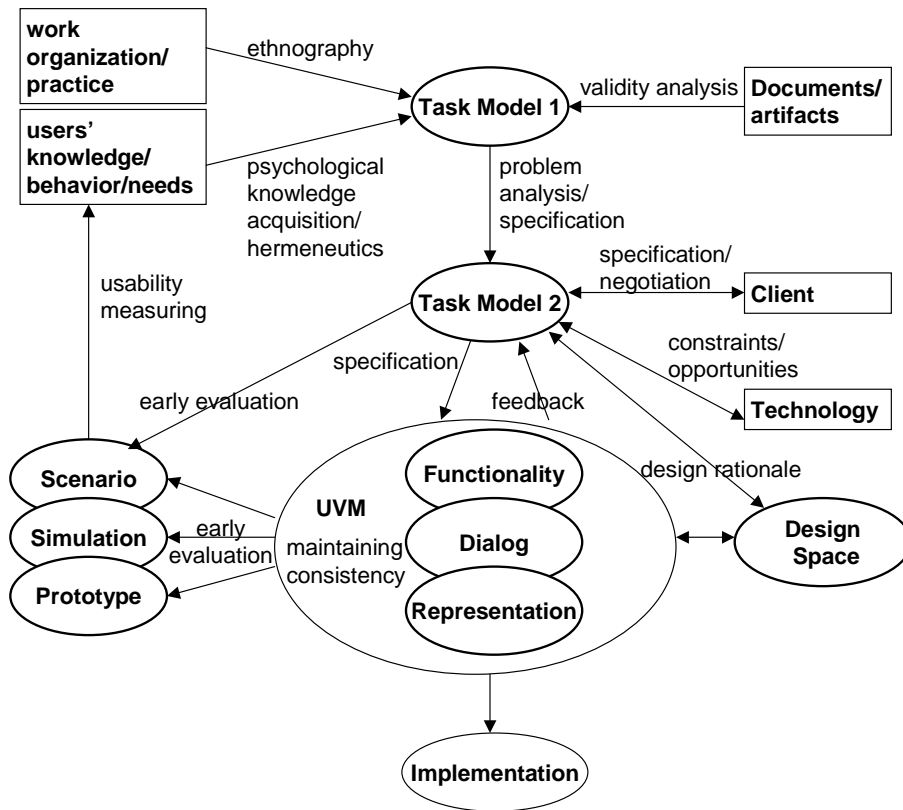


Figure 1.1: The DUTCH Design Process

is a *descriptive* task model which is used for analyzing the current task situation. The second task model is a *prescriptive* task model for the system to be designed.

1.1.1 Analyzing the current task situation (Task model 1)

In many cases the design of a new system is triggered by an existing task situation. Either the current way of performing tasks is not considered optimal, or the availability of new technology is expected to allow improvement over current methods. A systematic analysis of the current situation may help formulate design requirements, and at the same time may later on allow evaluation of the design. Whenever it is possible to study the "current" task situation, a thorough analysis is very valuable for the design of a new system. We use a combination of classical HCI techniques such as structured interviews (Sebillotte 1988) and CSCW techniques such as ethnographic studies and interaction analysis (Jordan 1996). The current situation is described by task model 1, also referred to as the "current task model".

1.1.2 Envisioning the future task situation (Task model 2)

Many design methods in HCI that start with task modeling are structured in a number of phases. After describing a current situation (task model 1) the method requires a re-design of the task structure in order to include technological solutions for problems and technological answers to requirements. Johnson et al. (1988) provide an example of a systematic approach where a second task model is explicitly defined in the course of design decisions. Task model 2, the future task model, will in general be formulated and structured in the same way as the previous model, but in this case it is not considered a descriptive model of users' knowledge but a prescriptive model of the future task world. Task model 2 describes the task situation as it should be when the system has been developed and is used.

1.1.3 Detailed design

After the task modeling activity, the actual system needs to be designed and specified. Task model 2 describes the envisioned task world where the new system will be situated. From there, the details of the technology and the basic tasks that involve interaction with the system need to be worked out. This activity consists of three sub-activities that are strongly interrelated: specifying the functionality, structuring the dialog between the users and the system, and specifying the way the system is presented to the user. This activity is focused on a detailed description of the system as far as it is of direct relevance to the end-user. We use the term User Virtual Machine (UVM) (Tauber 1988) to indicate the total of user relevant knowledge of the technology, including both semantics (what the system offers the user for task delegation) and syntax (how task delegation to the system has to be expressed by the user). When making the transition from task model 2 to a detailed design of the UVM, the users' tasks and the objects determine the first sketch of the application. The task or object structure is used to create the main displays and navigational structure. From there on, the iterative refinement process takes off together with the use of explicit design knowledge such as guidelines and patterns.

1.1.4 Evaluation and usability testing

During the entire process, some kind of evaluation activity can take place. As soon as an initial task model is available it can already be evaluated using scenarios and use cases. Later on when some initial sketches for the new system are known, mockups and prototypes can be used for early evaluation of design concepts. Each evaluation activity can cause another iteration. For instance, a task model may turn out to be incomplete after a mockup is evaluated in a critical scenario. In that case, the designers need to go back to the task model and rethink their model. When some parts of the design are worked out in detail, we can begin usability testing with a prototype and users. Early evaluation can be done by inspecting design specifications or by performing walkthrough sessions with designers and/or users.

1.2 Research Goals

The research goals addressed in this thesis are all concerned with the improvement of both the design process as well as the final product being designed. Nowadays, many systems are not usable enough and the question is how to develop usable systems in practice.

1.2.1 Improving the design process

Many design methods exist but only few of them try to address the entire design process. In task-based user interface design, we try to cover the entire process from early analysis to prototyping. Effective and practical methods are needed in practice and task-based design is intended to meet that need. The industry's context can be characterized by limitations in the available resources such as time, money, people, and skills. These limitations should be taken into account when designing. The main process goals this dissertation addresses are:

- *Improving the available techniques for both task analysis and design.* Although task analysis is an old topic, the techniques have not been developed far enough to be of sufficient value.
- *Reducing the effort of performing a detailed task analysis.* Partly because of poor techniques, performing an effective task analysis takes too much effort. This often leads to rejection of the entire activity.
- *Improving effective use of task analysis results in the design of the actual product.* Most task analysis methods are weak on the topic of how the results lead to new or improved systems. Consequently, task models often do not contain the knowledge that is needed in the later stages of design.

In this thesis, we try to address these issues by first analyzing task modeling on the conceptual level followed by the development of suitable representation techniques. Those representation techniques are based on the conceptual model for tasks and are supported by tools. Tools can reduce the effort to build and analyze task models. Consequently, task models will be of higher quality and require less effort to build.

1.2.2 Improving the final product

Although a structured analysis method can improve the final product by dictating the necessary activities, it is still not a guarantee for high quality products. Most methods are focussing on analysis activities or specification techniques while the actual design activities remain largely *creative* and ad hoc. The main problem in ensuring quality of the final product is to use the analysis results effectively in combination with *explicit* design knowledge. Incorporating design knowledge into the design process makes detailed design less prone to ad hoc design and increases at least the minimum quality of the product. The main research goals in designing for usability are:

- *Providing a better understanding of what usability is.* Usability is a poorly understood quality concept. Although most designers have an idea what it is conceptually, the concept remains fuzzy and hard to measure.
- *Allowing re-use of successful solutions in design.* Besides the use of guidelines, almost no re-use is done in interaction design. Design knowledge and the use of it is highly designer dependent. This gives design an ad-hoc character combined with a trial-and-error mentality.

In this thesis we address these issues first by providing a new framework for the usability concept. A better understanding of usability can facilitate measurements and comparisons hopefully leading to a more objective notion of quality in use of the system. Additionally, interaction design patterns are presented as a new way of documenting explicit design knowledge. Patterns describe solutions in a particular context and allow for re-use of successful solutions.

1.3 Research Approach

At the start of the project the main structure of the DUTCH design method was already developed. However, the theory, the tools and the techniques were in need of improvements. During the project the idea to develop tool support for the method has been a driving force. Effective tool support requires clear conceptual ideas about the necessary models and representations. A large part of the research has focused on developing task analysis to a point where tool support could actually be built. To do that, it was necessary to define a detailed meta level model of what needs to be described in task models. The task world ontology we developed is such a meta model. The task world ontology has been formalized and can be used in tool development. Our tool EUTERPE provides an operationalization of the ontology by offering ontology based representations.

The results of task analysis are to be used in the detailed design phase. This gives a different perspective on what kind of information task models need to contain. Trying to bridge the gap from analysis to design led to questions regarding the concept of usability. One of the important insights is that quality of the design is related to both task knowledge and design knowledge. This led to a refinement of the existing ideas on usability and the development of interaction design patterns.

An important question regarding the research methodology is how to determine the quality of our method. Since it is almost impossible to conduct a controlled experiment to compare design methods, we have adopted a more pragmatic approach. By continuously applying the method whenever we were able to do so, we tried out new ideas and assessed the value of the used techniques. The methods and techniques as described in this thesis show the result of this approach.

In the period the work for this thesis was conducted, one new method was published that has similar goals and activities i.e. *Contextual Design* (Beyer & Holtzblatt 1998).

Although they both share the main goals and ideas there are many differences. Where appropriate the differences are discussed in the relevant chapters.

1.4 Outline of this Thesis

The structure of this thesis loosely follows the activities that comprise the DUTCH design method. Before we discuss the first activity, chapter 2 discusses the usability concept in detail. For user interface design, high quality products are products with high levels of usability. This chapter explains what it is and how it is related to the users and their tasks. It gives a frame of reference on what quality in use is and it will be used in other chapters whenever quality aspects are discussed.

The first step in most design methods is an analysis of the current context of use of the product. In task-based design this is called task analysis. Chapter 3 investigates what the important aspects of the task world are and puts them into a basic structure called the task world ontology. It forms the basis of how we look at the tasks and goals that need to be supported by the system that is to be designed.

The task world ontology gives designers structure and is intended to link the important concepts from a theoretical perspective. In practice, designers also need techniques to describe or document all these aspects. Chapter 4 discusses representation and analysis techniques that designers can use in practice. These techniques are based on the ontology and form an implicit use of the ontology in practice. For producing these representations quickly, a modeling tool called EUTERPE has been developed which is discussed in chapter 8.

When the analysis has been more or less completed, the detailed design activities need to start. Creating detailed design solutions is done on the basis of the analysis results *and* the available design expertise. It uses the analysis results as input and produces a concrete design as output. Chapter 5 discusses the transformation of analysis results and highlights some of the common techniques that can be used. One of those techniques, New User Action Notation (NUAN), is an extension we developed to improve one of the existing techniques.

All of the techniques for describing the detailed design lack the explicit input of proven design knowledge. In chapter 6, interaction design patterns are introduced as a possible way to include explicit design knowledge into the design process. Design patterns capture proven design knowledge and provide means to define a link between user tasks and suitable design solutions. Some example patterns are included in appendix B.

In practice, time is always limited. Software tools are necessary for reducing the time needed to develop specifications throughout the design process. Chapter 7 discusses the state of the art in tools that can be used in task based design. Based on the overview of tools, several requirements are set for future tools.

EUTERPE is a tool developed to meet those requirements. It is directly based on the task world ontology and supports designers in creating representations and performing

semi-automatic analysis of task models. In chapter 8, the functionality is discussed together with an overview of the technical aspects of building an ontology based tool.

Although we have described a method for designing user interfaces that should be useful in practice, several issues are still present when the method is applied in practice. Chapter 9 gives an overview of the issues encountered when the method is (partly) applied. They give an informal validation of the method as well as a direction for further improvements.

In chapter 10 the main conclusions are presented together with suggestions for future research.

1.5 Publications

The chapters of this dissertation are largely based on previous published work. This section lists the publications this dissertation is based on.

Chapter 2 is based on a paper published at the *Interact 99 conference* (van Welie et al. 1999a).

The task world ontology and most of Chapter 3 has been published at *Design Specification Verification of Information Systems 98 (DSV-IS)* (van Welie et al. 1998a).

Chapter 4 is based on two papers published at the *European Conference on Cognitive Ergonomics* (van Welie et al. 1998b, van Welie, van der Veer & Koster 2000).

Chapter 5 contains work published at *DSV-IS99* (van Welie et al. 1999b) and the *Code-signing Conference 2000* (van Welie & van der Veer 2000).

The theory about patterns and the patterns themselves can be found in chapter 6 and appendix B. They have been published in *Tools for Working with Guidelines 2000* (van Welie, van der Veer & Eliëns 2000) and at the *Pattern Languages of Programming* conference (van Welie & Trætteberg 2000).

Chapter 8 contains portions from publications at *Designing Interactive Systems 2000* (van der Veer & van Welie 2000), and *ECCE 9* (van Welie et al. 1998b).

Two papers published at *European Conference on Cognitive Science Approaches to Process Control* (van der Veer et al. 1997, van Loo et al. 1999) describe work that is used in Chapter 9 and 4.

Chapter 2

Designing for Usability

2.1 Introduction

From a designer's point of view, the main goal in design is to design a highly usable and useful system (Hartson 1998). We design for people and the systems we develop need to be usable by them. The last decade has shown a trend towards systematic development of systems with a high degree of usability. Users are now recognized as important participants in the design process, for their knowledge about their work and sometimes even for active participation in designing a new system. Human centered design methods all share the thought that systems should be developed around the people that are going to use them.

These methods are based on the assumption that systems that are developed from a *human* and *work* perspective, will have a higher degree of usability. When comparing different methods on their ability to "deliver" usable systems, the usability of the final system is the main measure. Hence, it needs to be clear what usability is, how it can be determined and preferably how usability is influenced by certain design choices. In this chapter, we take a detailed look at the concept of usability.

Literature on user interface design frequently uses the term usability. Several definitions exist, some show a high degree of similarity but others may seem very different. In addition, several guidelines, standards, heuristics and checklists have been produced that all claim to deal with usability. Although it is not really a problem if each of these "lists" is designed for slightly different purposes (assessments/ improvement/ standardization), they do need to share a common view on the concept of usability. For designers that have to deal with these "lists" the question arises whether one list is "better" than another and when to use which list. Over the years, frequent usage of the term usability has cluttered the concept. Therefore, in this chapter we look into the different views on usability and we give a framework for dealing with these various views on usability.

2.2 Potential Benefits of Usability

Although user interface designers may see a high degree of usability as a main design goal, others may not. For systems where the amount of work that should be done by users is crucial, designing for usability is a fairly natural goal. For other systems, commercial or financial goals may be regarded far more important resulting in less attention for the usability goals. For example, the Dutch Railways introduced ticket vending machines as a way to reduce the personnel costs for the ticket counters. The machines were a success. Less personnel was needed although at the cost of a lower service to their customers. The machines had several usability problems causing longer waiting times and they were also unusable for several kinds of user groups (elderly people and non-Dutch speakers).

Usability is strongly of interest to the end users but it is also of interest to other stakeholders. A high degree of usability could bring several potential benefits (Karat 1994). These include:

- *Increased Efficiency.* Systems are often introduced to get more work done in less time. When the system is not usable the user will waste time struggling with the interface, making them less efficient in their work.
- *Increased Productivity.* A usable interface allows users to concentrate on the work rather than the tools they use.
- *Reduced Errors/Increased Safety.* A significant portion of the errors humans make can be related to poor design. In some cases, these errors may cause dangerous situations for both humans and the environment.
- *Reduced Training.* When a system closely matches the tasks humans have to perform and care has been taken concerning memorability, the need for extensive training is reduced. Hence, the time and costs are reduced.
- *Reduced Support.* A usable product may cause less problems for users and hence the need for product support is reduced.
- *Improved Acceptance.* When users like the system because it is very usable, they are more likely to accept and use it.
- *Savings in Development Costs.* Making changes early in the design life cycle is much cheaper than making changes late in the life cycle.
- *Increased Sales.* A system can have a competitive edge over other systems because of its usability.

Estimating how much is gained when designing for usability is hard to quantify, but the benefits have been shown to have a considerable impact in practice (Bias & Mayhew 1994). Some attempts have been made to create models that predict the gains in terms of costs (Mayhew & Mantei 1994). These benefits are not necessarily benefits for

all stakeholders of the system. Stakeholders may have different roles such as end-user, designer, software developer, project manager, or client. For some stakeholders certain benefits may actually turn out to be disadvantages. For example, companies whose core business is product support and training may see a decrease in demand for their services. Other companies may want to release a new version every year for commercial reasons, adding new needed functionality which requires additional training. The role of a user interface designer is to make the functionality of the system available to the users in a usable way. Other stakeholders may have different roles and consequently other interests.

This conflict in goals is normal and needs to be recognized by designers as well as the other stakeholders. It is the designers' task to guard and promote usability aspects despite the less than ideal circumstances. Unfortunately, in practice usability is a poorly understood concept and it often has to give way to other goals. Usability does not come for free and the benefits versus the costs are often discussed (Bias & Mayhew 1994). This situation can be improved in at least two ways. Firstly by providing a better understanding of what usability actually is, how to measure it and how to improve it. Secondly by providing methods and techniques for designing usable systems. Methods can make usability a manageable concept. The costs can be kept low and the impact of the effort can be predicted much better. This chapter focuses on understanding what usability is and the next chapters discuss methods and techniques for developing usable systems.

2.3 Understanding Usability

The first issue with usability is to understand why a system would be usable in the first place. Usability is concerned with humans interacting with a system for some purpose and therefore we need to look at what is known about *the users*, the *work* they do and about the *interaction* that takes place when using a system.

2.3.1 Understanding humans

The systems we design are being used by humans so we need to understand the abilities and limitations of humans. Especially cognitive and perceptual abilities are relevant to design but sensor and motor abilities are important as well. Humans have serious limitations when it comes to information processing and other tasks such as decision making, searching and scanning (Shneiderman 1998). The fields of cognitive psychology and ergonomics give a theoretical and practical background on these matters. Research in those fields has given useful knowledge that can be used in practice, for instance knowledge about working memory can directly be used to improve learnability of systems. Methods such as GOMS (Card et al. 1983) and CCT (Kieras & Polson 1985) have tried to incorporate cognitive aspects to predict the influence of changes to dialog aspects of a design. Other knowledge such as Fitts' Law (Fitts 1954)

and color perception can also be applied directly in screen design and layout (Mullet & Sano 1995).

When users work with a system they develop a mental model (van der Veer 1990) of the system. The mental model is part of the long term memory and it is an internal representation of the system. The mental model is formed through interaction with a system and is 'activated' when humans use a system. This mental model guides them in using the system and is modified when the model is not entirely valid. Generally speaking, usability problems are often caused by a mismatch between the users' mental model and the system structure, procedures and organization that the mental model is representing.

Another important aspect of knowledge about humans is the social and organizational viewpoint. Users perform their tasks in a larger context where they have a social and organizational position that is important to them. In this context they may have to work together or are part of a team. Consequently there are issues concerning individual and group knowledge. Contextual aspects about users have a less direct impact on the design process but are strongly related to the position of a new system in the organization where it is going to be used (Jordan 1996). Therefore, knowledge about humans as a social organization may not be immediately used for detailed design decisions but it is very useful when (re)structuring the work and organization.

2.3.2 Understanding the work

Generally speaking, humans use systems to aid them in their work. Therefore, understanding the work they do is crucial for designing usable systems. In their work, users perform tasks in order to achieve certain goals and some of these tasks may be performed using a computer system. In this respect, we can speak of task allocation between humans and computer systems. Designing often means changing this task allocation. A conservative approach to design is to allocate tasks performed by humans to the computer. However, this may lead to design solutions that do not take full advantage of the technology. A more innovative approach may change the task structure because of the available technology. In this case, the user's technology independent goals are being preserved while the used technology leads to a new task structure.

One difficult aspect in design is finding out what tasks the users perform and to distinguish goal directed tasks and instrumental tasks. Instrumental tasks exist because of the technology currently used. Goal directed tasks are needed for achieving a user goal. If instrumental tasks are being automated, there is a big danger of automating "the wrong thing". For example, if the classical typewriter would have been automated by replacing all instrumental tasks by corresponding tasks in a computer system, no usability gains would occur. Sometimes it indeed makes sense to change a large part of the task structure, especially when the tasks are often instrumental. Innovative design focuses on the main user goals whereas conservative design may lead to automation of instrumental tasks.

Understanding the work humans do is important for designing usable systems but using that knowledge in design can be difficult. Hierarchical models of the task structure are

widely accepted but many important aspects of work are not covered by these models. The work is performed in a certain *context of use* which also needs to be described in addition to details of the task structure. According to ISO9241 (ISO 1991c) the context of use comprises user, tasks, equipment, the social and physical environment. Understanding of the work is important when design choices are made but a direct relationship between work models and design solution is often hard to specify. In chapter 3 we look at all aspects of the context of use in detail.

2.3.3 Understanding the interaction

Understanding interaction means understanding what happens when users interact with a system. Users are confronted with a user interface that allows them to do many possible things. Users have to "see" what they can do and how they can use the available possibilities to reach their goals. Naturally, some interfaces work better than others. Every designer acquires skills and experiences during projects and that knowledge helps the designer in later projects. Because of this knowledge, the designer knows what kind of solutions work for which design problems. This knowledge about interaction comes not only from both practical experience but also from literature. Currently the amount of knowledge available in literature is rather limited which makes the personal experience of the designer an important factor for the usability of the system. The more explicit knowledge is available the more UI design becomes an engineering discipline.

Basically the only concrete design knowledge that can be used during design is embedded in guidelines. Guidelines deal with both structural (the dialog) and presentational aspects of a user interface. For example, guidelines on color use and button sizes refer to the presentation and guidelines on feedback and menu structure (Mayhew 1992, Shneiderman 1998) deal with dialog. Usually no explicit distinction between dialog and presentation is made, although both have a distinguishable impact on usability. Since guidelines often go into depths on describing a platform's style, mainly presentational aspects are covered and there is little guidance for dialog and functional aspects. Some guidelines such as the Macintosh (Apple Computer Inc. 1992) or MS Windows (Microsoft 1995) guidelines mainly describe a platform *style* and hardly contain style independent guidelines. The underlying assumption that applications that have been designed according to the guidelines have good usability, remains unjustified. Other guidelines such as those developed by Smith & Mosier (1986) focus on a narrow scoped list of guidelines dealing with detailed design choices and consequently they are quickly outdated by new technological developments. Some of the older guidelines were designed for character based applications and it is not clear to what extent they apply to e.g. WIMP (Windows, Icons, Menus and Pointers) interfaces or Virtual Reality interfaces.

Despite the differences in guidelines they certainly embody design knowledge and every designer should know them. However, there may be several reasons why the guidelines are not followed during the design process. Even if a designer tries to use the guidelines there are still many problems applying them. In Dix et al. (1998) a number of problems with guidelines are discussed such as when to apply a guideline or

choose one out of contradicting guidelines. Also the effectiveness of guidelines is under discussion and research has shown that not all guidelines are as practical as desired (Scapin & Bastien 1997).

Another way of capturing design knowledge is in design patterns. Such patterns describe generalized problems and proven solutions that can be immediately used in practice. Because design patterns work from a problem to a solution it is more likely to find guidance on structural aspects emphasized in design patterns. Research on design patterns has just started and only a small number of patterns have been written. See chapter 6.

2.4 Definitions of Usability

In this section, definitions of usability are discussed. After a short discussion on heuristics, guidelines and principles, section 2.6 introduces a framework that provides an integrating view on usability and related concepts. There is not one agreed upon definition of usability and usability certainly cannot be expressed in one objective measure. Several authors have proposed definitions and categorizations of usability and there seems to be at least some consensus on the concept of usability since they mostly differ on more detailed levels.

Shackel defines usability of a system as *“the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfil the specified range of tasks within the specified range of environmental scenarios”* (Shackel 1991). The definition is then operationalized by using four criteria; effectiveness, learnability, flexibility and attitude. This definition already shows the importance of the context of use and that usability is depending on it.

Shackel also makes a distinction between usability, utility (also known as usefulness) and likability. Löwgren (Löwgren 1995) elaborates on the issue of usability and utility. Whether or not utility is “included” in usability is a matter of perspective but it is agreed that both are desirable. Especially the engineering approaches seem to make this distinction whereas others simply see utility as part of usability. The ISO 9241-11 (ISO 1991c) standard contains another abstract three-parts definition of usability; *The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.* “Effectiveness” is defined as *the accuracy and completeness with which users achieve specified tasks* and “Efficiency” as *the resources expended in relation to the accuracy and completeness with which users achieve goals.* “Satisfaction” is a subjective measure and concerns the *comfort and acceptability of use by end users.* Satisfaction is actually the least understood aspect of usability. In (Hassenzahl et al. 2000) the relationship between satisfaction aspects such as fun and challenge are compared to effectiveness and efficiency aspects. One of the interesting conclusions is that the most effective interface is not necessarily the most fun or pleasant to used, it might even be boring.

In another definition by Preece et al. (1990) usability is defined as *a measure of the ease with which a system can be learned or used, its safety, effectiveness and efficiency, and the attitude of its users towards it*. This definition does not make an explicit reference to the context of use or specified user goals and suggests that usability is a "property" of a system.

These definitions approach usability from a theoretical viewpoint and may not be very practical. Nielsen (1993) has a slightly different definition that is specified in elements that are more specific, see Table 2.1. A similar definition is given by Shneiderman (1998). Shneiderman does not call his definition a definition of usability but he calls it "*five measurable human factors central to evaluation of human factors goals*". As can be seen from Table 2.1, Shneiderman's definition is essentially identical to Nielsen's definition and only differs in terminology. Note that the potential benefits mentioned in section 2.2 are almost completely derivable from the definition given by Nielsen. It is surprising that both Nielsen and Shneiderman do not mention any aspect concerning to usefulness (e.g. task completion, can the users complete their tasks).

Table 2.1: Usability as in ISO9241-11, B. Shneiderman and J. Nielsen

ISO 9241-11	Shneiderman	Nielsen
Efficiency	Speed of performance	Efficiency
	Time to learn	Learnability
	Retention over time	Memorability
Effectiveness	Rate of errors by users	Errors/Safety
Satisfaction	Subjective satisfaction	Satisfaction

Table 2.2 shows the usability factors as described by Dix et al. (1998). This categorization looks rather different from the ISO, Shneiderman and Nielsen definitions. Dix defines three main groups; learnability, flexibility and robustness, suggesting that those concepts are on the same abstraction level. The groups are specified further by factors that *influence* the concept they belong to. For instance, consistency influences learnability positively when a design is consistent within the application and between applications on the same platform. Learnability is subdivided into aspects that are mostly of cognitive nature thereby giving more grip on the important cognitive skills of users in relation to learnability. Robustness corresponds more or less to effectiveness. In flexibility also some lower level concepts such as multi-threading are mentioned but most aspects are mainly related to efficiency. This categorization raises the issue of which forces *influence* usability rather than how it is defined.

Table 2.2: Usability categorization by Dix et al.

Learnability	Flexibility	Robustness
Predictability	Dialog initiative	Observability
Synthesizability	Multi-Threading	Recoverability
Familiarity	Task Migratability	Responsiveness
Generalizability	Substitutivity	Task conformance
Consistency	Customizability	

When comparing these categorizations and definitions it is remarkable that Nielsen and the ISO standard give a concise outline of the term usability while Dix focuses

more on the concrete elements that influence usability. From a practical viewpoint, Dix's categorization gives the designer concrete means for improving the usability of a design. On the other hand, it is odd that Nielsen's notions of efficiency or error rate can not be found in Dix's categorization, as they are clear indicators of usability. The most interesting aspect of Dix's categorization is that it raises the question what the causes for sub-optimal usability might be and how it might be improved.

2.5 Heuristics, Guidelines and Principles

In addition to definitions of usability, there are also several lists of design principles, heuristics or criteria. These should help the designer in designing usable systems. The assumption is that when the guidelines are appropriately applied, they have a positive effect on the overall usability. Both Nielsen and Shneiderman give a set of design heuristics to follow that should have a positive effect on his categories. Table 2.3 shows that Shneiderman's eight "golden rules" for design (Shneiderman 1998) and Nielsen's heuristics are very similar:

Table 2.3: Rules and heuristics

Shneiderman	Nielsen
Strive for consistency	Consistency
Enable frequent users to use shortcuts	Shortcuts
Offer informative feedback	Feedback
Design dialogs to yield closure	
Offer error prevention and simple error handling	Prevent Errors / Good Error Messages
Permit easy reversal of actions	Forgive the user
Support internal locus of control	
Reduce short-term memory load	Minimize User Memory Load
	Simple and Natural Dialogue
	Help and Documentation
	Speak the User's Language
	Clearly Marked Exits

Both Dix's categorization and Nielsen's heuristics show that the root factors that influence usability need to be found in the cognitive and perceptual abilities of users such as working memory, problem solving, decision making, searching and scanning (Shneiderman 1998). A similar list is given by the ISO 9241-10 (ISO 1991*b*) standard and is called a set of dialog principles, see Table 2.4. Where Nielsen and Shneiderman mainly focus on human specific guidelines, the ISO 9241-10 principles also explicitly mention *suitability for the task* referring to the importance of *work* in usability. On the other hand, none of these principles give concrete hints on how to actually *achieve* this "suitability".

Another interesting list is the list of ergonomic criteria developed by Scapin & Bastien (1997), see Figure 2.1. These criteria have been developed for finding usability problems in a walkthrough evaluation. Most of the criteria can also be found in Nielsen's and Shneiderman's lists but some are new. For example, Scapin's list of ergonomic

Table 2.4: ISO 9241-10 dialog principles

Dialogue Principles
Suitability for the task
Self-descriptiveness
Controllability
Conformity with user expectations
Error tolerance
Suitability for individualization
Suitability for learning

criteria mentions "grouping and distinguishing items". Grouping is concerned with the "visual organization of information items in relation to one another" and is therefore concerned with presentational aspects. Usually there is no explicit distinction between dialog and presentation level aspects and only the design as a whole is considered. It is useful to realize that measures have both dialog and presentation aspects. However, often a clear distinction cannot be made. Mullet & Sano (1995) show the importance of presentational aspects and their effect on usability. In addition, they also provide techniques for improving presentational aspects such as grouping, grids etc.

Scapin's list also shows that some aspects are hard to organize. For example, *flexibility* is a main category in Dix's categorization but it is merely an aspect of *adaptability* in Scapin's criteria.

1. Guidance
1.1 Prompting
1.2 Grouping and distinguishing items
1.2.1 Grouping by location
1.2.2. Grouping by format
1.3 Immediate feedback
1.4 Legibility
2. Workload
2.1. Brevity
2.1.1. Conciseness
2.1.2. Minimal actions
2.2 Information density
3. Explicit control
3.1. Explicit user actions
3.2. User control
4. Adaptability
4.1 Flexibility
4.2. Users' experience
5. Error management
5.1. Error protection
5.2. Quality of error messages
5.3. Error correction
6. Consistency
7. Significance of codes
8. Compatibility

Figure 2.1: Ergonomic criteria by Scapin et al.

Arnold (Arnold 1998) gives another set of heuristics for design. These heuristics are much more task related with a strong focus on cognitive aspects of interaction.

1. Adapt as much as possible to the user's knowledge of language, work procedures, computer operation, etc.
2. Supply means for action preparation, especially for orientation and action program design.
3. Contribute to an uninterrupted, swift execution of action programs by giving signals, and feedback on the course and results of activities.
4. Leave room for modification of action programs and their manner of execution.
5. Supply means for supervision of action execution including anticipation to future operations or actions.
6. Take into account the capabilities and limitations of sensory, cognitive and motor mechanisms.
7. Take into account and try to support user's tendency towards optimization of action efficiency; allow changes of the regulation level.
8. Contribute to the maintenance of a workload balance.
9. Take into account and try to support user's tendency to be engaged in more than one task at the same time.

2.6 A Layered Model of Usability

All the different definitions and principles make usability a confusing concept when actually designing a new system. Usually authors spend a lot of effort trying to find out what is the "best" set of principles or to define a "complete set of heuristics". Although these "aids" are useful it remains unclear how they are related and how to judge when an "aid" is useful to improve usability. Figure 2.2 shows a layered model of usability (van Welie et al. 1999a) that helps understanding the various aids. On the highest level, the ISO definition of usability is given split up in three aspects: efficiency, effectiveness and satisfaction. This level is a rather abstract way of looking at usability and is not directly applicable in practice. However it does give three solid pillars for looking at usability that are based on a well-formed theory (ISO 1991b). The next level contains a number of usage indicators which are indicators of the usability level that can actually be *observed* in practice when users are at work. Each of these indicators contributes to the abstract aspects of the higher level. For instance, a low error-rate contributes to a better effectiveness and good performance speed indicates good efficiency. The desired "level" for each of the usage indicators depends on the nature of the system. For a production system efficiency may be the main goal but for an entertainment web site satisfaction may be far more important than efficiency.

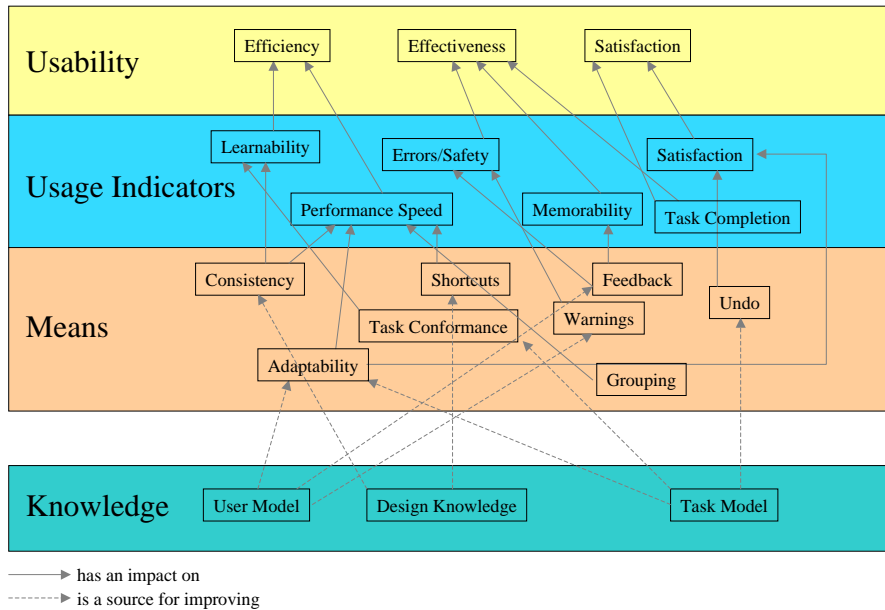


Figure 2.2: A Layered Model of Usability

One level lower is the level of means. Means cannot be observed in user tests and are not goals by themselves whereas indicators are observable goals. The means are used in "heuristics" for *improving* one or more of the usage indicators. For instance, consistency may have a positive effect on learnability and warnings may reduce errors. On the other hand, high adaptability may have a negative effect of memorability while having a positive effect of performance. Hence, each means can have a positive or negative effect on some of the indicators. The means need to be "used with care" and a designer should take care not to apply them automatically. The best usability results from an optimal use of the means where each means is at a certain "level", somewhere between "none" and "completely/everywhere/all the time". Determining the appropriate usage of means is looking for the right compromise. It is up to the designer to find those optimal levels for each means. In order to do that, the designer must use the three knowledge domains (humans, interaction, and work) to determine the appropriate levels. For example, when design knowledge is consulted by using guidelines, it is clear that the guidelines should embody the knowledge of how changes in use of the means affect the usage indicators.

The means of Figure 2.2 are examples of means and the given set is certainly not complete. The different lists and heuristics all give suggestions for useful means. More research is needed to determine which means are most effective for improving usability. One interesting issue is what kind of groupings of means can be made. Scapin's list shown in Figure 2.1 suggests a reasonable set of groupings. The usage indicators of Figure 2.2 are complete, i.e. based on the operational definitions of usability and usability metrics. However, satisfaction is difficult to measure and may in the future be

split up in more detailed aspects.

When comparing the definitions of section 2.4 and 2.5 using our layered model it is clear that some definitions are on a single level and that others have aspects from more than one level. For instance the dialog principles of the ISO9241-10 standard mention "suitability for learning" (which is learnability) and "error tolerance" which are both usage indicators, but it also mentions "suitability for individualization" (which is adaptability) which is a means. Looking at the usability categorization of Dix it shows that mainly means are summed up and the categories are a mixture of means and indicators; learnability is an indicator and flexibility and robustness are means. Scapin's list of ergonomic criteria is essentially a list of means grouped together. Shneiderman's golden rules say to strive for "a certain level" for each of the eight specific means he considers the most important.

When means and indicators are mixed in one list the semantics of the list are ambiguous which causes confusion and makes it more difficult to apply them for actual design decisions. Additionally, it is good to realize that none of the lists can be regarded as being complete. Each of the lists has at least one element not mentioned by any of the others. Therefore, all of the lists can be useful but the most important thing is to realize what the semantics of a list is. That way, it is clear how a list can be used and what the limitations are.

2.7 Usability and Software Quality

From a different perspective, that of a software engineer, usability is closely related to the quality of software. In the ISO 9126 (ISO 1991*a*) standard, usability is put in relationship with software quality. Usability is seen as one of the "six quality aspects of software quality from a users' perspective", see figure 2.3. Each aspect can be further described in factors.

The software quality model suggests that the six aspects are independent. However, if we look at usability from a user's perspective we see that it surely depends on the functionality (e.g. the suitability) as well as efficiency of the system. Usability is split up in three factors (understandability, learnability and operability) which is a rather narrow definition in comparison to the definitions of section 2.4. Furthermore, these quality aspects are not explicitly related to any stakeholders. Not every stakeholder has an interest in all aspects. For end-users maintainability and portability of a system are probably not of interest. Other factors such as reliability are of interest for end-users but only indirectly, users mostly expect a system to work and be reliable.

Another important issue is the notion of context. The model suggests that software quality is a "property" of the system itself. However, for all aspects and factors it holds that they do not have *any* absolute value, they all depend on a context. For example, fault tolerance depends on the hardware used and environmental conditions. Similarly, changeability depends on the changes that occur rather than those that are foreseen. Therefore, it is important to notice that software quality, just like usability, is not an inherent attribute but depends on a specified context of use.

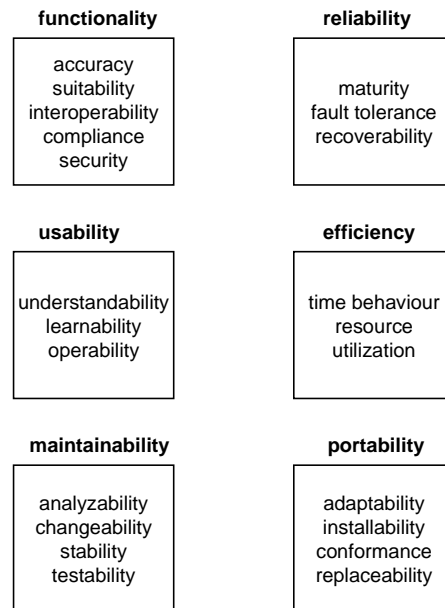


Figure 2.3: The ISO9216 Software Quality Model

In the software quality model, internal and external quality attributes are distinguished. Internal quality attributes are related to the product itself. External quality attributes are related to the product in its environment. However, it remains unclear if users are part of this environment. The external quality attributes suggested in the ISO 9126 standard seem directed to the performance of the system when used, not the perceived quality by a certain user. Therefore, in (Bevan 1999) a revision of the standard is described that explicitly sets users apart.

The term *quality in use* is used to indicate the quality when used by a specified user in a specified context and can hence be regarded as a synonym for usability. The usage indicators and means are related to the different kinds of quality as is shown in Figure 2.4. It shows the relationships between software quality and quality in use/usability as we described it. The usage indicators are quality metrics for quality in use which are influenced by means, possible by changing external attributes of the system. Quality in use depends on the context of use in the same way external quality depends on the system's context.

2.8 Usability Evaluation and Improvement

Evaluation of usability can be done afterwards and during design. The usefulness of all the guidelines, heuristics and other aids is related to the kind of evaluation that is being conducted. In terms of our layered usability model, evaluating with users should be done by looking at the usage indicators. When evaluating during design without users,

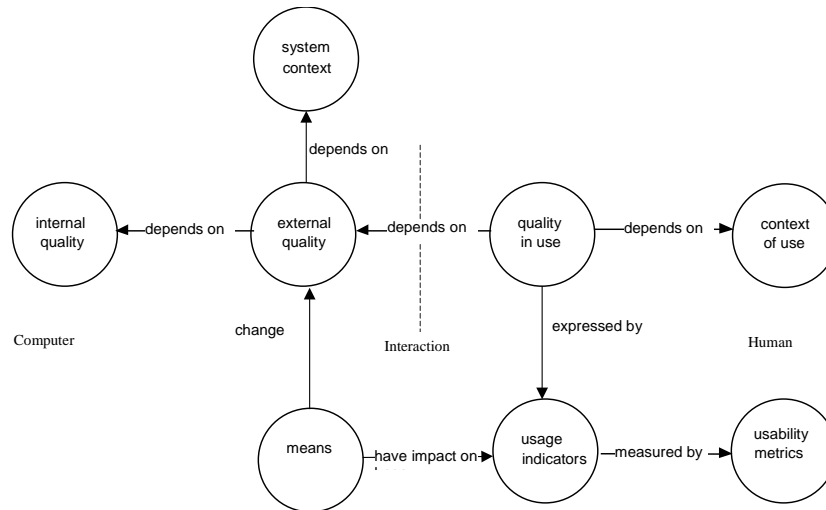


Figure 2.4: Usage indicators, means and software quality

the usage indicators cannot provide any data and designers can only look at how the means are used and make an estimate on their impact.

2.8.1 Measuring usability

Evaluating with users is a good method for obtaining data about the actual usage. Using scenarios and other techniques, data about the number of errors or speed of performance can be obtained which should provide a good indication of the usability of the product. The scenarios should be explicitly related to the usability goals for the system. The actual measuring activities should be done in the actual context of use for which the system is designed. When this is not possible usability labs may be an inferior surrogate. Measuring of the usage indicators is not always easy. Task performance times are easy to measure but satisfaction and memorability are harder to quantify. Using questionnaires such as QUIS and SUMI, a more standardized measurement can be obtained, although these are only valid for certain classes of systems. Measuring of usage indicators can be done using *usability metrics*. In Whiteside et al. (1988) and Nielsen (1993) lists of metrics are given. Table 2.5 summarizes all given metrics.

Evaluation during the design process is more problematic than evaluating with users. Although mockups and paper prototypes can be tested with users, the usage indicators cannot be evaluated directly. What can be done is looking at the means that influence the usage indicators. Using walkthroughs and scenarios each of the means can be evaluated by looking at the way they are present in the design and by estimating the positive or negative impact on the usage indicators. For instance, it can be checked if a design is consistent with a platform's style guideline or if sufficient warnings are given. This is where the heuristics and ergonomic criteria of Scapin & Bastien (1997)

Table 2.5: Usability Metrics

Metric	Usage Indicator
Time to complete a specific task	Performance Time
Number of commands used	Performance Time
Percent of task completed per unit time	Performance Time
Relative time spent in physical actions	Performance Time
Relative time spent in mental actions	Performance Time
Time spent waiting for the system responds	Performance Time
Number of tasks that can be completed within a given time limit	Performance Time
Number of regressive behaviors	Memorability
Number of system features users can remember afterwards	Memorability
Time spent in errors	Errors
Percent of number of errors	Errors
Number of repetitions of failed commands	Errors
Number of immediately subsequent erroneous actions	Errors
Time spent using help or documentation	Learnability
Frequency of help and documentation use	Learnability
Ration of users using effective vs. ineffective strategy	Learnability
Number of good and bad features recalled by users	Satisfaction
Percent of favorable/unfavorable user comments	Satisfaction
Number of users preferring your system	Satisfaction
Number of times user expresses frustration or satisfaction	Satisfaction
Number of times interface misleads the user	Task Completion
Percent of task completed	Task Completion
Number of available commands not invoked	Task Completion
Ratio of successes to failures	Task Completion
Number of runs of successes and of failures	Task Completion
Number of times users need to work around a problem	Task Completion
Number of times the user is disrupted from a work task	Task Completion
Number of times user loses control of the system	Task Completion

and Nielsen (1993) are very useful. This kind of early evaluation does not replace the need for late evaluation with users but can contribute when good choices of means can be made.

Another way of ensuring usability during the design process is by using formal design models. Many models and techniques exist for describing designs using formal notations. State charts, GOMS (Card et al. 1983), ConcurTaskTree's (Palanque & Paternò 1997) and similar notations are used to describe designs. These kinds of notations are usually strong in describing structural aspects of a design (the dialog structure) and very weak at describing presentational aspects. In (Payne & Green 1989) Payne says, "*as far as TAG is concerned, the screen could be turned off*". Although ETAG (Tauber 1990) also does not consider presentational aspects it deals with functionality. Reisner's Action Language (Reisner 1983) also allows knowledge sources to be modeled, e.g. it is possible to model if a user can get needed information from the screen or from working memory.

In relation to the means of our model, this is already a big limitation since a lot of means such as consistency, warnings or feedback are strongly related to presentational aspects. A heuristic that says "speak the user's language" is difficult to deal with using formal models. Another factor is that most formal models are usually built with the viewpoint of describing "correct" use of the application and therefore do not describe error handling or issuing warnings. For formal models to be really valuable, they

should include the context of use as well and relate properties of the system model to the context models.

2.8.2 Improving usability

When an evaluation shows that the usability needs to be improved the problem is to find out which means need to be changed and how they need to be changed. As was mentioned earlier means may have a positive effect on one usage indicator while having a negative effect on another. In some cases, it may be obvious how to improve usability but in cases where problems are of a more structural kind it may not be so simple to solve. In that case, the designer has to take a step back and look at the knowledge domains again. The knowledge domains are the *only* sources for judging why and how a means is to be changed. For instance, when the task conformance is seen as a problem the task model can give the designer information about what is wrong with the task conformance. Similarly, the user model may give information about the memory limitations which may require the design to have more or better feedback of user actions. Unfortunately the knowledge domains are not always available or written down in a way that makes it easy to use them in practice. Task models may not contain the right information or the available guidelines do not say anything about a particular problem.

2.8.3 Usability process improvement

Designing for usability should be a focus in the system development process. However, most organizations have not sufficiently recognized the importance of usability and have not incorporated it in their current design methods. Within the field of software engineering the concept of maturity is often used to indicate how good the process is in a certain area. To indicate how well an organization is dealing with usability, a usability maturity scale was developed in the INUSE project (Earthy 1999). The levels are:

- X: Ignorance - “We don’t have problems with usability”, Usability is not discussed as an issue.
- A: Uncertainty - “We don’t know why we have problems with usability”, User-Centred processes are not implemented, or fail to achieve their purpose
- B: Awakening - “Is it absolutely necessary to always have problems with usability?”, User-Centred processes are implemented but are performed by inappropriate staff using sub-optimal methods
- C: Enlightenment - “Through management commitment and improvement of human-centered processes we are identifying and resolving our problems”, User-Centred processes are implemented and produce results, but these results do not always give the expected benefits to the software development process

- D: Wisdom - “Usability defect prevention is a routine part of our operation”, User-Centred processes are integrated into the software life cycle and used to improve all work products
- E: Certainty - “We know why we do not have problems with usability”, The culture of the organization is user-centered

The experience of the INUSE project was that much of European industry is at level X, A or sometimes B on this scale. It shows that there is still a strong need for methods that incorporate usability aspects into the current design practice. Obviously, most companies still need to reach the awakening process before the necessary changes can occur.

2.9 Usability and Design Methods

In (Nielsen 1993) Nielsen says “*A Holy Grail for many usability scientists is the invention of analytic methods that would allow designers to predict the usability of a user interface before it has even been tested.*”. This implies two areas of interest for analytical methods; modeling humans and work, and modeling design solutions. The first is important for the understanding of the human and the work they do. The search is for models that capture all aspects of humans and work that are relevant for designing for usability. The second is important for predicting the usability of the design solution (among other things like communication). Crucial for predicting usability is the success of human and work models since they provide the context of use.

2.9.1 Modeling humans and work

Task modeling research has a strong background in cognitive psychology and the focus is on how users perform their work and think about their work from the viewpoint of looking at individual users. The strongest link is the fact that if you know more about the user and his work you can build a more usable system. In practice, most modeling methods such as HTA (Annett & Duncan 1967) do not model much more than a task hierarchy. The task hierarchy helps to establish task conformance and does not help to improve other means such as adaptability or error prevention. However, when the task model is taken as a model describing the users, their work, the objects they use and the organization they are part of, it is possible to capture information that can actually help to improve usability. The task model should be able to answer questions about the task world related to effective use of means. Table 2.6 shows some examples of questions for a task model in relation to a means. As can be observed from Table 2.6, a task model needs to contain more than a simple task hierarchy.

Besides these concepts, the right information about the concepts needs to be captured. For instance, when a designer wants to know what the critical tasks are, the task model must be able to make a distinction between critical and non-critical tasks, for instance

Table 2.6: Questions for Task Models

Means	Question for task model
Warnings	What are the critical tasks? How frequent are those tasks performed? Always performed by the same user?
Adaptability	Which types of users are there? Which roles do they have? Which tasks belong to which role?
Undo	Which tasks should be undoable? Which tasks have not undoable side effects?
Error prevention	What errors are expected? What are the consequences for users? How can prevention be effective?

by means of task typing. When the questions from Table 2.6 need to be answered, all of these aspects and probably even more need to be added. We use the means to check whether our task analysis method GTA (van der Veer, Lenting & Bergevoet 1996) contains the necessary information and will indeed add missing aspects.

2.9.2 Modeling the system

Dialog modeling and especially formal dialog modeling (Palanque & Paternò 1997) enjoys great interest in HCI research. One problem of most formal methods such as described in (Palanque & Paternò 1997, Payne & Green 1989) is that they are designed to describe the behavior of the interface and not to enable usability evaluation. Some methods can be used to do verification of systems but this is limited to properties such as state-reachability, deadlocks and interaction path lengths. Although interaction paths can say something about the speed of performance, it is impossible to make predictions about other usability aspects. In the same way as for task models, the means can be used to determine some requirements for dialog models that enable usability evaluation. A dialog model also needs to be built using the right concepts and they should be verifiable in some respect. Looking at Table 2.7 it is clear that a dialog model needs to

Table 2.7: Questions for dialog evaluation

Means	Questions
Warnings	When are warnings given?
Speed of performance	How many steps needed for accomplishing a task?
Undo	Which functions are undoable?
Feedback	When and how is feedback given?
Consistency	What are similar task-action decompositions?

be more than a state-based description. A dialog model must be able to identify system feedback as either a warning or state feedback and must also contain more detailed information about the functionality as in how far it can be undone or not. In fact there are techniques that partially address these aspects; UAN (Hix & Hartson 1998) deals explicitly with feedback and TAG (Payne & Green 1989) allowed analysis of consistency by identifying similar task-action decompositions. When such additions

are done, a dialog can be evaluated by looking at how well constraints are satisfied, e.g. "Does the user get a warning before executing a function that is undoable?" or "Given a starting point what is the average number of steps needed to perform this task?"

2.10 Summary

Designing for usability is important for end-users as well as for other stakeholders in systems design. Usability is the main design quality in user interface design. A good understanding of the concept of usability allows effective design processes and reduces the costs of these activities. This chapter has put usability into a reference framework using a layered model consisting of usage indicators, means, and knowledge. This framework shows what usability is by distinguishing what can be measured and how usability is affected by changes in the design. The usage indicators show which aspects can be measured to evaluate actual performance with end users. The means are changeable aspects in a user interface that have a certain impact on the usage indicators. In order to configure the means, the knowledge domains (task, work and interaction) provide the basis for making the appropriate design decisions.

We have shown that for most of such changes, knowledge about the context of use is needed. The users and their work are the primary source that guides the design process. Together with explicit design knowledge, designers can achieve the necessary quality in use. The fundamental problems of designing for usability are to learn the characteristics of the actual users and their tasks, to create systems that fit those users and their tasks, and to test the systems for any mismatches. Understanding humans, their work and the interaction with computers is crucial. In short, usability is all about the context of use of a system. In the next chapter, we take a detailed look at the different aspects of the context of use, also called the task world.

Chapter 3

An Ontology for Task Models

3.1 Introduction

Task analysis is the process of gathering data about the tasks people perform and acquiring a deep understanding of it. Several methods can be used to gather the data, including interviewing, observation, talk aloud protocols and ethnographic work place studies. While these techniques each have their own characteristics and problems, *documenting* the data and being able to write down the understanding gained is another problem. This chapter is concerned with the modeling and structuring techniques that are required once data is available.

The process of structuring data and gaining insight into the data is called task modeling. Task models are the "product" of the entire task analysis activity. Neither task analysis nor task modeling is a new topic. However, task analysis and task modeling have always been relatively ill-defined activities, where no commonly accepted models or representations were used, except perhaps for the task decomposition. Recently, task analysis has regained interest since the advent of user centered design. The users and their work are again recognized as important for developing usable interfaces. Although recognized, the problem of adequately describing the task world still remains unsolved.

In this chapter, we first look at *what* aspects of the task world need to be described for the purpose of user interface design. Then we look at how these aspects are *related* to each other. We propose a set of fundamental concepts and relationships which we integrate in an ontology. By ontology we mean a way of how we view the world. Using the ontology we can explain and derive the commonly used representations for task models and see how different representations are related to each other and to the ontology. The ontology acts as a conceptual foundation for describing task world knowledge while representations are used by the analyst to document real data.

For task-based user interface design, the task model serves as the primary source for determining the functionality the system needs to offer and how the interaction could

be structured. Additionally, the task models are used to restructure work in general and to provide detailed information about the users and stakeholders of the system. When designing for usability this information is required when creating or modifying user interface designs. As discussed in the previous chapter, usability depends on the context of use and the purpose of task modeling is to map the task world in sufficient detail.

3.2 A Short History of Task Analysis

The first publication about task analysis was probably “Hierarchical Task Analysis” (Annett & Duncan 1967). Task analysis has developed slowly over the next thirty or so years, see Figure 3.1. In the eighties there was a “revival” when methods such as GOMS (Card et al. 1983) and TAKD (Johnson et al. 1984) emerged. Task models were getting more formal and research was split in two areas; that of the task analysis *process* and that of task and knowledge *modeling*. On the process side, research was directed towards knowledge acquisition techniques such as structured interviews and ethnographic work place studies (Nardi 1996, Jordan 1996). On the modeling side, there was a distinction between techniques for modeling the knowledge needed to perform tasks and techniques for modeling the task structure itself. TAKD is an example of a method for knowledge description while GOMS focusses on a low-level description of the tasks of a user working with an interface. Soon models were becoming more formal in order to describe the low-level tasks or knowledge in sufficient detail. In this period, confusion of terms started when the word “task” was by some used as synonym for “goal” or “action” while others made explicit distinctions between these terms.

Later on the notion “task model” also acquired two different interpretations; that of a *descriptive* model and that of a *prescriptive* model. In the latter, task models were soon used as a way to describe the dialog (e.g. GOMS) of a user interface instead of describing the user’s task independent of the technology. In that case, the concept task is closely related to application functions and less to the more cognitive nature of tasks.

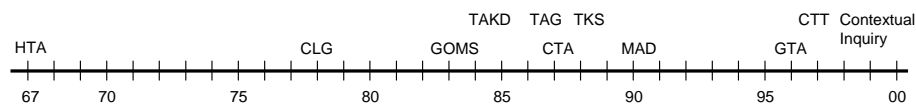


Figure 3.1: Task Analysis Methods in Time

Of all the methods listed in Figure 3.1, HTA (Annett & Duncan 1967) and GOMS (Card et al. 1983) have had the most impact. HTA because it was the foundation for many other methods and GOMS because it is probably the one most used in practice. When taking a closer look at these methods, it becomes clear that representational aspects have been weak spots of most methods. Most methods are rather formal which made them more powerful and at the same time less usable. It soon became clear that tools were needed to make these modeling activities feasible for practitioners. Strangely enough, such tools hardly appeared. Usually large textual representations need to be constructed and it is often difficult to understand what exactly is being represented.

Some methods such as CTA (Barnard 1987) and TKS (Johnson et al. 1988) focus on the psychological aspects but fail to convert their ideas in practical representation techniques. They aid in understanding the cognitive aspects of humans in general but hardly help to understand the specifications themselves. Both the cognitive techniques as well as the other techniques focus on describing the tasks of individuals and not of groups.

3.3 Methods and techniques

Few methods and techniques that have been developed over the years are still in use. In this section, we discuss the most influential ones. The reason for looking at the older methods and techniques is that each of them contains aspects that are important for task modeling. By looking at them, we can get an overview of interesting aspects in task modeling techniques. In this section we discuss some methods and techniques that are exemplary in that each contains one or more important aspects.

3.3.1 Hierarchical Task Analysis (HTA)

Hierarchical Task Analysis (Annett & Duncan 1967) is one of the oldest general purpose task description techniques and contains many ideas found in later techniques. HTA is a process of developing a description of tasks in terms of *operations* and *plans*. Operations are things people do to reach goals and plans are statements of conditions that tell when each operation is to be carried out. The operations can be hierarchically decomposed and with each new sub-task, a new plan exists. Figure 3.2 shows an example of a HTA diagram taken from (Kirwan & Ainsworth 1992).

In HTA, tasks are defined as activities that people do to reach a goal. A goal is then defined as a desired state of the system under control or supervision. However, in the notation used only a task hierarchy is modeled and the goals are not explicitly represented. In (Kirwan & Ainsworth 1992) HTA is also explained and there they speak of a goal hierarchy *and* a task hierarchy but they actually mean the same thing. This confusion between a task and goal hierarchy is common in task analysis literature.

It is remarkable that HTA uses a clear representation that is fairly usable, something which is lacking in most later representations. Understanding what you specify was already considered important. Although HTA was defined as "a process of developing ...", the process part of HTA was hardly explicitly defined. The main idea was to iteratively develop the HTA models by collecting data from interviews or documents. Therefore, the merit of HTA is mainly the clear representation technique.

3.3.2 Goals Operators Methods Selectors (GOMS)

GOMS (Card et al. 1983) is undoubtedly the most influential task analysis technique to this date. Ironically enough, it is not a real task analysis technique but rather a task-based dialog description technique. It describes the dialog steps that are needed to

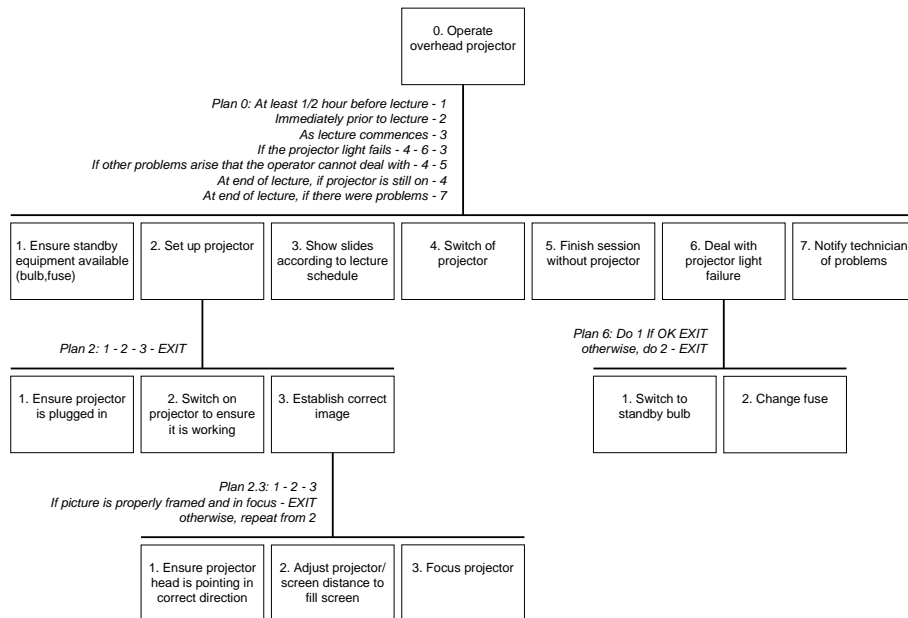


Figure 3.2: HTA example

perform a task with a specific user interface. GOMS is intended to estimate user performance based on a description of the system before the system is actually built. Over the years many variations of GOMS have been developed (John & Kieras 1996). GOMS does not explicitly define a process other than iteratively constructing GOMS models. The focus in GOMS is completely on the representation technique and performance estimation.

Goals are states that the user wants to achieve. A *Method* is a possible way to achieve the goal. A method usually contains a number of steps. For describing when to use which methods, *Selection* rules set the criteria for using a method. *Operators* are used to describe the nature of the steps in a method. There are three kinds of operators: external (perceptual and motor actions), mental and primitive. External operators need to be specified by the analyst and primitive operators are predefined. Primitive operators are often directly derived from the hardware that is being used, e.g. a mouse. Mental operators can be predefined or analyst-defined. Predefined mental operators include RECALL, RETAIN, FORGET, RETRIEVE and DECIDE while analyst-defined operators can be anything, e.g. FIND-MENU-ITEM "Cut".

The *Operators* in GOMS are the basis for making predictions about the expected user performance. For each operator an average duration is defined and by counting all the steps, the total expected time to complete the task is calculated. The prediction part of GOMS is interesting and controversial at the same time. GOMS tries to estimate *Learning Time*, *Execution Time* and *Mental Workload* solely by counting steps. It is assumed that the steps modeled by GOMS statements normally take 0.1 sec and

primitive operators such as mouse movements take between 0.2 and 1.1 sec. Mental operators are estimated to take 1.2 sec in case of lack of any other information. For experienced users this time should sometimes be zero seconds. The assumed values for operators have been heavily criticized (Nielsen 1993). Their values have proven to be rather variable and unpredictable which makes the foundation for estimating user performance weak, if not invalid. Other criticisms concern the fact that GOMS assumes error-free behavior and does not distinguish novice and expert behavior. In summary, the estimation part of GOMS remains controversial even after almost twenty years of research. However, it can still be used to compare design alternatives against each other to see which requires the minimal number of steps.

In GOMS, there is not an explicit task concept. In (Kieras 1994) it is stated that a task "describes the parameter list for methods". This suggests that a method can be seen as a task description. GOMS models can be hierarchical and the resulting hierarchy is necessarily a goal hierarchy. However, as Figure 3.3 shows, others might call the stated goals tasks. The example is an adaptation from the example found in (Kieras 1994).

```

Selection rule set for goal: select text
  If text-is word, then
    accomplish goal: select word.
  If text-is arbitrary, then
    accomplish goal: select arbitrary text.
Return with goal accomplished.

Method for goal: select word
  Step 1. Locate middle of word.           (M)
  Step 2. Move cursor to middle of word.   (P)
  Step 3. Verify the the correct word is located. (M)
  Step 4. Double-click mouse button.       (BB)
  Step 5. Verify that correct text is selected. (M)
  Step 6. Return with goal accomplished.

Method for goal: select arbitrary text
  Step 1. Locate beginning of text.        (M)
  Step 2. Move cursor to beginning of text. (P)
  Step 3. Verify that the correct beginning is located. (M)
  Step 4. Press mouse button down.         (B)
  Step 5. Locate end of text.              (M)
  Step 6. Move cursor to end of text.      (P)
  Step 7. Verify that correct text is marked. (M)
  Step 8. Release mouse button.            (B)
  Step 9. Verify that the correct text is selected. (M)
  Step 10. Return with goal accomplished.

```

Figure 3.3: A GOMS example

Concerning representations, GOMS is usually described in "structured" text like Figure 3.3 shows. This makes it tedious to use in practice and several attempts have been

made to developed tools. QGOMS (Beard et al. 1996) is a tool that uses graphical representations for GOMS models. GOMSED (Wandmacher 1997) is another tool that automates the calculation process. GOMS is one of the methods for which tools have been developed, albeit never commercially.

3.3.3 Méthode Analytique de Description des tâches (MAD)

MAD (Scapin & Pierret-Golbreich 1989) is the task modeling part of a larger method for designing interactive systems (Sebillotte 1995). MAD is the modeling part and the other half consists of structured interviewing techniques (Graesser et al. 1981). In MAD, task models are similar to HTA models except that the plan construct has been replaced by so called constructors. A constructor specifies the time dependencies of a task's subtasks i.e. a constructor scopes over all subtasks. The constructors such as PROV, SEQ, PAR, ALT, SIM and ELEM are used to specify the time order in which tasks are executed. Additionally, pre-conditions can be specified for each task in order to "tune" the time ordering. The task model is graphically represented in a similar way as in HTA but without plans and with constructors.

Table 3.1: MAD Task Body Attributes

Task Body Attributes	
Identification Number	Alphanumeric
Name	Alphanumeric
Goal	Alphanumeric
Comments	Alphanumeric
Degree of Freedom	{Optional, Obligatory}
Interruptability	{True, False}
Upper Task	Identification Number
Priority	Integer
Modality	{Manual, Automatic, Interactive}
Type	{Cognitive, sensori-motor}
Frequency	{high, medium, low}
Centrality	{important, not important}
Important entities	{among the task's objects}
Experience	{user = novice, occasional, expert}
State	{waiting for execution, inaccessible, in execution, interrupted, finished, ignored}
Mandatory sub-tasks finished	integer

The most interesting aspect of MAD is the fact that *templates* are used to describe the task details. Details include pre- and post-conditions, initial and final states, task types, priorities and interruptabilities. In MAD, a task consists of two parts: a *conditions* part and a *body* part, see Tables 3.1 and 3.2 taken from (Gamboa Rodriguez & Scapin 1997). In most other methods a task is regarded as a "black box" but MAD has shown that there are many relevant aspects of a task to be described.

The interesting part is that in MAD tasks are modeled in great detail but no other concepts are modeled i.e. no roles or objects. Using templates to describe an arbitrary set of attributes for task (or other concepts), is a technique that can be used with most other approaches as well. A disadvantage of using templates is that the amount of

Table 3.2: MAD Task Conditions

Task Conditions	
Initial State	State
Input Condition	{triggering, execution, stop}
Final State	State
Output Condition	{end}

documentation is dramatically increased. In MAD, most of the fields are merely for description purposes and are not used in any automated evaluation or transformation.

3.3.4 Groupware Task Analysis

Groupware Task Analysis (van der Veer, Lenting & Bergevoet 1996) is a recent method that combines aspects from several methods. In GTA, it is recognized that the systems that are being built nowadays are rarely used by single users in isolation. Therefore, GTA puts an emphasis on studying a group or organization and their activities rather than studying single users at work. Essentially, GTA consists of a conceptual framework that specifies relevant aspects of the task world that need attention when designing Groupware.

The broad conceptual framework is based on experiences with a variety of different approaches and on an analysis of existing methods for HCI and CSCW (van der Veer et al. 1995). When designing Groupware systems it is necessary to widen the notion of a task model to include descriptions of many more aspects of the task world than just the tasks. GTA makes a distinction between a *descriptive* task model and a *prescriptive* task model. The first is called task model 1 and the latter task model 2. The framework as such is intended to structure task models 1 and 2, and, hence, as a guidance for choosing techniques for information collection in the case of task model 1. Obviously, for task model 2 design decisions have to be made, based on problems and conflicts that are represented in model 1, in combination with requirement specifications as formulated in interaction with the client of the design.

In GTA, task models for complex situations are composed of three different aspects: *agents*, *work*, and *situation*. Each describes the task world from a different viewpoint, and each relates to the others. This allows designers to view and to design from different angles, and allows design tools to guard consistency and completeness. The three viewpoints are a superset of the main focal points in the domain of HCI as well as CSCW. Both design fields consider agents ('users' vs. 'cooperating users' or user groups) and work (activities or tasks, the objectives or the goals of 'interaction' and the cooperative work). Moreover, especially CSCW stresses the situation in which technological support has to be incorporated. In HCI this is only sometimes, and then mostly implicitly, considered. This section discusses the three views of the conceptual framework.

Agents

The first aspect focuses on agents. "Agents" often indicate people, either individuals or groups, but may also refer to systems. Agents are considered in relation to the task world, hence we need to make a distinction between humans, as acting individuals or systems, and the roles they play. Moreover, we need the concept of organization of agents. Humans have to be described with relevant characteristics (e.g. the language they speak, the amount of typing skill or experience with MS-windows). Roles indicate classes of agents to whom certain subsets of tasks are allocated. By definition, roles are generic for the task world. More than one agent may perform the same role, and a single agent may have several roles at the same time. Organization refers to the relation between agents and roles in respect to task allocation. Delegation and mandating responsibilities from one role to another is part of the organization.

Work

In GTA, both the structural and the dynamic aspects of work are considered, so the task is taken as the basic concept and several tasks can share the same goals. Additionally a distinction is made between tasks and actions. Tasks can be identified at various levels of complexity. The unit level of tasks needs special attention. A distinction is made between (1) the lowest task level that people want to consider in referring to their work, the 'unit task' (Card et al. 1983); and (2) the atomic level of task delegation that is defined by the tool that is used in performing work, like a single command in command driven computer applications. The latter type of task is called 'Basic task' (Tauber 1990). Unit tasks are often role-related. Complex tasks may be split up between agents or roles. Unit tasks and basic tasks may be decomposed further into user actions and system actions, but these cannot really be understood without a frame of reference created by the corresponding task, i.e., actions derive their meaning from the task. For instance, hitting a return key has a different meaning depending on whether it concludes a command, or confirms the specification of a numerical input value in a spreadsheet.

The task structure is often at least partially hierarchical. On the other hand, resulting effects of certain tasks may influence the procedures for other tasks (possibly with other roles involved). Therefore, the task flow and data flow over time as well as the relation between several concurrent flows need to be understood. A special concept is event, indicating a triggering condition for a task, even if the triggering could be caused by something outside the task domain we are considering.

Situation

Analyzing a task world from the viewpoint of the situation means detecting and describing the environment (physical, conceptual, and social) and the objects in the environment. Object description includes an analysis of the object structure. Each thing that is relevant to the work in a certain situation is an object in the sense of task analysis,

even the environment is an object. In this framework, "objects" are not defined in the sense of "object oriented" methods including methods, inheritance and polymorphism. Objects may be physical things, or conceptual (non-material) things like messages, gestures, passwords, stories, or signatures. Objects are described including their structure and attributes. The task environment is the current situation for the performance of a certain task. It includes agents with roles as well as conditions for task performance. The history of past relevant events in the task situation is part of the actual environment if this features in conditions for task execution.

The process and representations

GTA also defines a general process of task analysis and again a combination of techniques is used. For extracting data, GTA uses techniques such as structured interviews, interaction analysis, hermeneutics¹, document analysis and observation. Task modeling is then performed as a cyclic activity where models are created, evaluated, modified, and new information is gathered. Concerning representations, GTA mainly uses hierarchical decompositions and templates. Not only tasks and their structure are modeled but also related roles and objects.

3.4 An Ontology for Task World Models

In the previous sections different approaches to task modeling were highlighted. The approaches are not directly contradictory and they all seem to describe certain important aspects of the task world. In this section, a general ontology (van Welie et al. 1998a) is proposed that captures the most important aspects of the task world. It is an ontology in the sense that it explains the structure of the world and defines how we look at it. In other words, it describes task models on a meta level. First we discuss four views that structure the way we can look at task modeling. From those views the fundamental concepts and relationships that play a role are extracted and formed into an ontology.

3.4.1 Modeling work structure

Work structure modeling is the oldest and most common activity in task analysis. Humans do not think about their work as a collection of tasks but they think in a structured way about their activities (Sebillotte 1988). This structure can be captured in a task decomposition tree. The tree forms a hierarchy where the high level tasks are found at the top of the tree and the most basic tasks are at the leaf nodes. Such a "classical" task tree is usually enhanced with *constructors* that indicate time relationships between tasks. Many methods including HTA (Annett & Duncan 1967), MAD (Scapin & Pierret-Golbreich 1989) and MUSE (Lim & Long 1994) use this kind of task trees.

¹The process of gaining a methodological objective understanding of human behavior by iteratively interpreting subjective observation data. (van der Veer 1989)

Work structure is one of the most important aspects in task analysis. A design of an interactive system usually means restructuring the work and removing or adding tasks. A task tree can already give an indication of aspects that are considered sub-optimal in the current situation. For example, certain subtasks could be part of many complex tasks and could be automated. In other cases, tasks may turn out to be too complex and simplification is needed. When designing for usability, the work structure is important for developing the most appropriate interaction structure and functionality.

Distinguishing Tasks and Goals

A common definition of a task is "*an activity performed to reach a certain goal.*" A goal is then defined as "*a desired state in the system or task world.*". Distinguishing between tasks and goals can be very useful when analyzing a task model. For example, a complex task which has goal X may have a subtask with goal Y. In that case, that subtask "belongs" to that task but since it does not have the same goal, one could wonder whether it is really a needed task or if it causes problems. An example is a copying task where the user checks if there is paper in the copier. Checking the paper has "maintenance" as a goal and would ideally be unnecessary.

Some methods implicitly presume a one-to-one mapping between tasks and goals; for instance a Task Knowledge Structure (TKS (Johnson et al. 1988)) contains only a goal substructure which would be called a task substructure by others' methods. Other methods, such as GTA (van der Veer, Lenting & Bergevoet 1996) and MAD (Scapin & Pierret-Golbreich 1989), allow a goal to be reached in several ways. In this way, each task has a goal and goals can be reached by several tasks. In fact, this is similar to GOMS where different methods are selected to reach a goal. One step further is to define a task hierarchy *and* a goal hierarchy, which occurs only in complex situations.

In practice, distinguishing between tasks and goals is not always so easy or clear. Usually this is an iterative process where the distinction gradually becomes clear. When describing *detailed* actions of a user at work goals often indicate states of the *system* but when higher level goals are modeled they are often related to states or particular configurations in the *task world*. Additionally, in complex task trees based on real life situations, tasks near the leaves in a tree are usually connected with individual goals, and tasks represented by high level nodes are often closely tied with organizational goals (van der Veer et al. 1997). When modeling complex situations where the organization is of great relevance, it is important to be aware of the difference between individual and organizational goals and the ways they are related.

Describing Task Details

While in most cases a task hierarchy can already show a lot of interesting information, other task details may also be important. For example, conditions describe when exactly this task can be executed or when it stops. Other details may describe the exact transformations that occur in the tasks, the priority of the tasks or whether they can be

interrupted. In highly event driven tasks it may be vital to know what the priorities of the tasks are and whether or not tasks can/may be interrupted.

Besides such properties tasks can also be assigned a *type*. For example, tasks can be characterized as *Monitoring Tasks*, *Decision Making Tasks*, *Action Tasks* or *Searching Tasks*. The interesting aspect of distinguishing task types is that they have characteristic use of cognitive resources such as perception, attention, memory and motor capacity. This may be of importance when designing user interfaces because each task type poses constraints on the possibilities. At present it is unclear which task typing is needed. At least a distinction can be made in mental and non-mental tasks but even for mental tasks there is not one fixed list of possible types.

3.4.2 Modeling the work flow

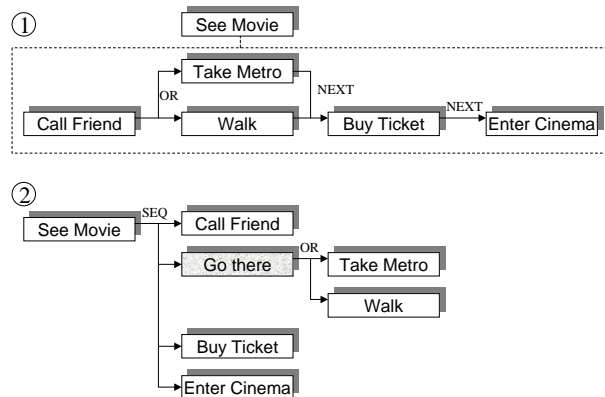


Figure 3.4: Problems with representation of time in trees

Another common feature of most task models is Task Flow, which indicates the order in which tasks are executed. Two forms of flow models can be distinguished: (1) workflow representations, with time on one axis, and (2) task trees enhanced with the "classic" constructors that give a kind of time structure (mixing time and task decomposition). Theoretically they are equally powerful to express any kind of task flow. However, the second type of flow model suffers from the fact that usually many constructors are needed and extra "dummy" tasks need to be added. For example, in figure 3.4, two representations are used. The first one is a task flow representation with time on the x-axis. The second is a task decomposition with constructors that represents the same task flow as the first representation. Because constructors scope over all subtasks, the second representation needs an extra task "Go there" which may not be desired. In Paternò's "ConcurTaskTrees" (Paternò et al. 1997) these types of tasks are called "abstract tasks". The important point here is that for both representations the specified task flow is basically the same but the visual representation does not always allow specification of the task structure as desired. Since both representations can be useful there is a need for an underlying model that allows both.

Events

Events are used to model external dynamic aspects of the task: things that happen in the task world, over which the agent does not always have direct control (e.g. an alarm goes off, a film breaks, a power supply fails or new mail arrives). Sometimes there may be no need to explicitly incorporate the event in the new design but in other cases incorporation is important. For example, it may prove very useful to model the agent's reaction to an event and how it influences the sequence in which tasks are performed. In complex situations, work can be highly event driven, e.g. in Air Traffic Control where people start and stop doing tasks all the time.

3.4.3 Modeling work artifacts

Artifact/object modeling is an addition to task analysis that resembles data structure modeling of the final design and implementation. The purpose is to say something about the objects, as they are physically present in the task world or mentally present in the user's mind. Not every object may be directly included in the new design but in the case of models for automated user interface (UI) generation there is usually a very strong link between objects and UI widgets, such as buttons or menus. The question remains how much object modeling should be in task models. Extensive data modeling does not appear to directly help in improving the usability of the product. It also depends on the purpose of the task model; models used as a basis for automatic UI generation have different requirements than models used for evaluation. For example, in GTA only the structure of the objects and the tasks they are used in are recorded. Other models such as ConcurTaskTrees (Paternò et al. 1997) and TKS (Johnson et al. 1988) also include actions that are performed on the object.

The most important purpose of a task is that it "changes" something, otherwise the task has no reason for existence. By change we mean any sort of change, including adding information (changing an unknown to a known). Some task analysis methods such as ConcurTaskTrees (Paternò et al. 1997) describe this with task input and task output objects.

Another way to describe changes is to specify the initial and final states in terms of object attribute values as done in MAD. In this way the information passing is indirectly achieved through changes in object attributes. There is no fundamental difference since the list of input and output objects can be generated from the task attributes. However, it is possible that the changes are not explicitly recorded, such as in the mental processes involved in a human's decision. In models that use object actions, changes are usually defined in the actions instead of the task states.

3.4.4 Modeling the work environment

Not only the work itself but also the work environment is important to study. In the past most methods focused on modeling one user and that user's tasks. However, in

current applications group aspects are becoming more important. Classic task modeling methods lack the power to deal with these situations, modeling only the static part of the task world by identifying roles. This neglects other parts of the organization and dynamic aspects of the task world.

People rarely perform their work in solitude. They work together with their colleagues and share offices, they help each other and form a social group. Certain aspects such as work place layout are traditionally the field of Ergonomics but are certainly also important for user interface design.

Physical Workplace Layout

One aspect of the work environment is the actual physical layout. How big is the room? Where are objects positioned and what are their dimensions? The physical layout can be modeled by assigning a dimension and location attribute to artifacts but in practice it is usually done by sketching the layout. Usually this is sufficient to gain the required understanding.

People and Organizations

Modeling the task world means modeling the people that are part of it and modeling its structure, which is often a part of the organizational structure. While it may be useful to see a model of the "official" organizational structure, for task analysis the structure of how tasks are *actually* being done is more relevant than how they officially should be done. Specifying roles in the organization and agents' characteristics gives relevant information that can be used in design. The roles then need to be attributed to agents. In TKS (Johnson et al. 1988), a role is defined to be responsible for performing the tasks it encompasses; for example, a movie projectionist is responsible for starting a movie. However, in real organizations task responsibilities frequently need to be handled more flexibly resulting in responsibilities being shifted by delegation or by mandate. The agent playing a role may therefore not perform the task he or she is responsible for; a movie projectionist could have someone from the snack bar push the button to start the movie.

Roles and Actors

In classic task analysis literature, as well as in ethnography, concepts such as actors and roles are commonly referred to for describing tasks and the task world. Although these terms are intuitively appealing, they can cause confusion when they need to be named during task analysis. A role is defined by the tasks the role is responsible for, e.g. a projectionist is responsible for starting and stopping the movie projector as well as setting up the movie projector. Mayhew (Mayhew 1992) defines an actor as a class of humans whereas others consider a particular person an actor. Usually there is no need to consider a particular person and provide a name for an actor (e.g. Chris, Pat) since we are only interested in describing relevant characteristics of the actor. Confusion

arises when an actor is to be named and the only sensible name seems to be the role name. For instance the actor who has the projectionist role is most intuitively called the "projectionist" which is already his/her role name. Therefore it is usually better to name these actors arbitrarily (A,123, People having role X) and simply record characteristics such as language, typing skill, computer experience, knows how to use Word etc. The important part is their characteristics and their relationships with roles. In other cases, where it does not matter who actually performed the task, it is sometimes more useful to specify that a task was performed by a role rather than by a particular actor. Sometimes even a computer system is the actor of a task (e.g an automated movie projector).

Work Culture

Every work environment has its own culture which defines the values, policies, expectations, and the general approach to work. The culture determines how people work together, how they view each other socially and what they expect from each other. Taking the culture into account for UID may influence decisions on restructuring of work when rearranging roles or their responsibilities.

Roles are usually used to describe the formal work structure extended with some "socially defined" roles. In practice, roles such as "management" or "marketing" influence each other and other roles. These kind of influence relationships are part of the work *culture*. Describing work culture is not straightforward but at least some influence relationships and their relative *strengths* can be modeled. Other aspects of culture include policies, values and identity (Beyer & Holtzblatt 1998).

3.4.5 Defining an ontology

From the four viewpoints that have been expressed, we can now propose a task world ontology. The ontology defines the basic concepts and relationships that we regard most relevant for the purpose of a task analysis. Basic, in this case, indicates that we are able to describe other relevant concepts and relations by using this minimal set of concepts and relations. The ontology is of importance because it is the conceptual basis of the information that is recorded and the way it is structured and may be represented. The ontology is constructed on the basis of literature and on our own experiences in task modeling. It has evolved over the years and elements have been changed or added when needed. The ontology should not be regarded complete or final and at most we claim that this ontology will help to cover many important aspects in a *systematic* way. From our experience doing design studies the ontology proved adequate but no formal evaluation has been done. Hence, the ontology as formulated should not be taken too strict. If during a study other relationships or attributes seem important, they should be added "on the spot". The ontology as specified in this chapter is intended to serve as a starting point which assures broad coverage of important aspects.

The concepts defined here are based on GTA and can be found in most other task models as well (with the exception of the event concept). This section will define the concepts and will define their relationships in detail.

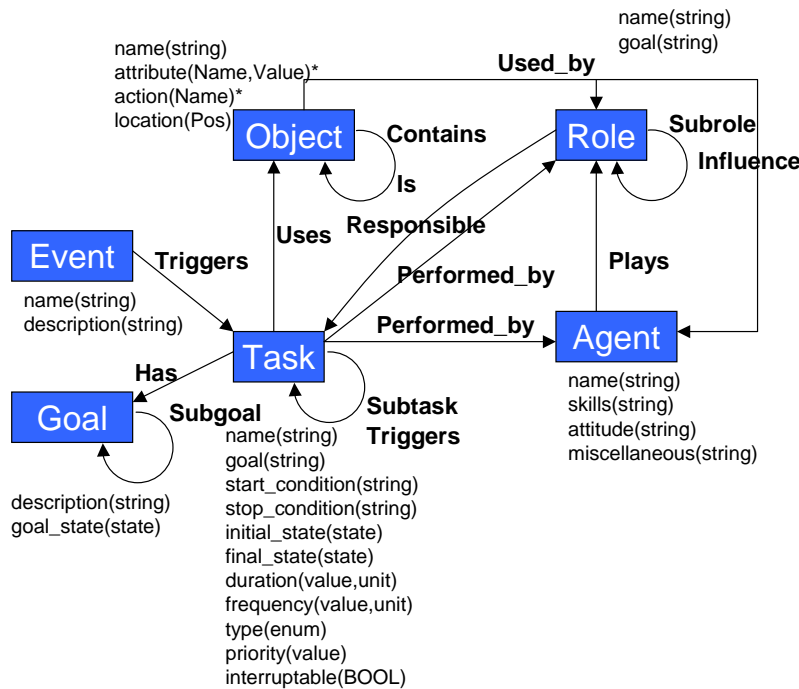


Figure 3.5: The task world ontology

- Task.** A task is an activity performed by agents to reach a certain goal. A task typically changes something in the task world and requires some period of time to complete. Complex tasks can be decomposed into smaller subtasks. Tasks are executed in a certain order and the completion of one task can trigger the execution of one or more other tasks. A task could also be started because of an event that has occurred in the task world. Important for the task concept is the distinction between unit tasks and basic tasks, where (ideally) a unit task should only be executed by performing one or more basic tasks. The relationship between the unit task and basic task is interesting because it can indicate the problems that an agent may have in reaching his goals.
- Goal.** A desired state in the task world or of the system. A goal can be reached by one or more tasks. Goals may also have sub-goals.
- Role.** A role is a meaningful collection of tasks performed by one or more agents. The role is meaningful when it has a clear goal or when it distinguishes between groups of agents. A role is consequently responsible for the tasks that it encompasses and roles can be hierarchically composed.
- Object.** An object refers to a physical or non-physical entity. A non-physical entity could be anything ranging from messages, passwords or addresses to gestures and stories. Objects have attributes consisting of attribute-name and value

pairs. What can be done with an object is specified by actions, for instance move, change, turn off etc. Furthermore, objects may be in a type hierarchy and can also be contained in other objects.

- **Agent.** An agent is an entity that is considered active. Usually agents are humans but groups of humans or software components may also be considered agents. Agents are not specific individuals (like "Chris") but always indicate classes of individuals with certain characteristics.
- **Event.** An event is a change in the state of the task world at a point in time. The change may reflect changes of attribute values of internal concepts such as Object, Task, Agent or Role or could reflect changes of external concepts such as the weather or electricity supply. Events influence the task execution sequence by triggering tasks. This model does not specify how the event is created or by whom.

These concepts are related in specific ways. The following list sketches the relationships that we are using. For each relationship the definition is given and explained. Figure 3.5 shows all the concepts and relationships.

- **Uses.** The uses relationship specifies which object is used in executing the task and how it is used. The uses relationship typically changes the state of the object.
- **Triggers.** The triggers relationship is the basis for specifying task flow. It specifies that a task is triggered (started) by an event or a task and how it is triggered. Several trigger types are possible such as OR, AND, NEXT to express choice, parallelism or sequences of tasks.
- **Plays.** Every agent should play one or more roles. The plays relationship also indicates how this role was obtained. For instance by delegation, mandate or a socially determined reason.
- **Performed_by.** The relationship performed by specifies that a task is performed by an agent. This does not mean that agent is also the one who is responsible for the task because this depends on his role and the way it was obtained. When it is not relevant to specify the agent that performs the task, a role can also be specified as the performing entity.
- **Has.** The has relationship connects tasks to goals. Each task has a goal that defines the reason for performing the task. A goal could be either a personal or business goal.
- **Subtask/Subgoal.** The subtask/subgoal relationship describes the task/goal decomposition.
- **Subrole.** The subrole relationship brings roles into a hierarchical structure. The subrole relationship states that a role includes other roles including the responsibility for the task that encompasses the role. When a role has subroles the task responsibilities are added up for the role.

- **Influence.** A role can influence another role. This is part of work culture and has a certain strength.
- **Responsible.** The responsible relationship specifies a task for which the role is responsible.
- **Used_by.** The used by relationship indicates who used which object and what the agent or role can do with it. The agents' rights regarding objects can be of existential nature, indicate ownership, or indicate daily handling of objects.

These relationships form a basis and other relationships can be derived from it. For example, one could be interested in who is *involved* in a certain task. That information can be derived using the *responsible*, *performs*, *plays* and *sub-role* relationships.

3.5 Related Work

Herczeg has also developed an ontology (Herczeg 1999) for task models. He calls his model a "task analysis framework" but actually defines an ontology similar to our ontology. As can be seen in Figure 3.6, the ontology is almost a sub-ontology of ours. The only notable difference is the *tool* concept. In our ontology we do not make a distinction between a *Managed Object* and a *Tool*, in fact both are regarded *Objects*. Herczeg uses Tools to indicate the "support systems for execution of tasks by operators" and they are characterized by a set of functions they support.

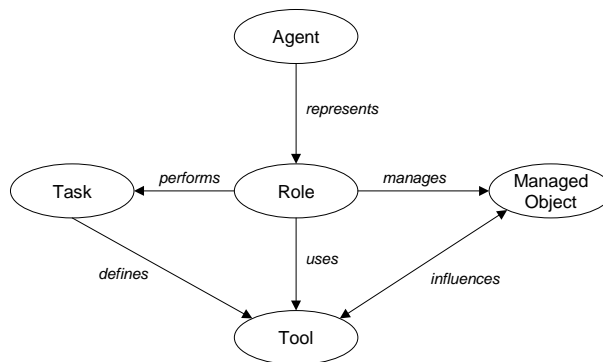


Figure 3.6: The Herczeg's ontology

Herczeg also defines attributes for each concept. The attributes are similar to the ones found in our ontology and include priority, interruptability and frequency. Herczeg defines three layers of use; first the model itself, then *classes* of the concepts that depend on the specific design case and *instances* for the final instantiations.

As Figure 3.6 shows, Herczeg's ontology can be seen as a simple version of the ontology as we defined it. This gives an indication that there is starting to become an agreement on the main concepts and relationships that are important for task modeling.

3.6 Summary

In this chapter, we first discussed the history of task analysis and the most influential methods. The methods all focus on slightly different aspects and there has been a development from a hierarchical structure describing the tasks of one user, to models that include work flow modeling, organizational and social aspects of groups of users. The central question of this chapter is *what* aspects need to be described for the purpose of user interface design.

In order to capture what needs to be described, an ontology for task models was proposed after a discussion of important aspects of the task world. The ontology defines the basic concepts and relationships that are relevant for the purpose of task analysis. The main concepts are *Goal*, *Task*, *Agent*, *Role*, and *Event*. Such an ontology helps understanding and modeling of work. For the designer, the ontology functions as a pair of "polarized" glasses with which the designer looks at the task world and structures it.

Designers do not use the ontology as a way to write down the knowledge that is gained. In that respect, the ontology is a theoretical model. Usually representations are used to communicate and document the knowledge. In the next chapter, we discuss how the ontology provides a theoretical basis for representations and analysis techniques that can be used in practice.

Chapter 4

Task Modeling and Analysis

4.1 Introduction

Task modeling is about modeling of use-relevant task knowledge, either of an *existing* task world or an *envisioned* task world. In this chapter, we discuss how such knowledge can be modeled, analyzed and documented. The representation techniques discussed apply to both current task models and envisioned task models. Although the same representation techniques can be used, the interpretations may vary slightly.

Task modeling¹ is the activity of transforming raw task and user related data or envisioning ideas into structured pieces of task knowledge. This knowledge is usually documented in a specification that uses several different representations. Each representation is intended to emphasize a certain aspect of this knowledge. Considering the complexity of the task world and the various possible views, it is clear that several different representations are needed. Ideally, the analysts have a collection of representations at hand that covers all aspects and views. At the same time, it is preferable that such a collection is kept small so that the designer does not drown in a plethora of overlapping representations. Representations must be useful and usable for designers. This chapter discusses the common representations for task modeling and defines a collection of representations that together cover most aspects of the knowledge that need to be documented, while minimizing the overlap.

Additionally, we take a look at task model *analysis* using the ontology described in chapter 3. Task modeling is typically an iterative process where structures are constantly changed as they are created and (re)interpreted. The interpretation process is done by analysis of what is modeled and comparing it with the data or problems to be solved. In a *current* task model, the specification needs to be interpreted for identification of the weaknesses and opportunities for improvements in the task world, once the pieces of knowledge become "stable". This is after all the main purpose of task analysis. In an *envisioned* task model, the specification needs to be analyzed to make

¹In this chapter, task analysis refers to the activity of analyzing the task model itself.

sure the new situation actually optimizes the task world. We propose a set of formal properties for automatic evaluation and heuristic evaluation that can be used to analyse task model specifications.

4.2 Representations for Task Modeling

The task world ontology that was discussed in chapter 3 is a model on the conceptual level and does not define any representations. The ontology is not intended to do so and merely gives structure to how we can look at the task world. In that sense it defines a “pair of polarized glasses” for looking at the task world. For documenting all that can be seen with these “glasses”, a variety of representations is needed. The task world is usually complex and simply cannot be captured in one single representation. It is therefore necessary to have a collection of representations that each show a partial view on the task world. Each view can then focus on one or two concepts with some relationships and together these views cover all important aspects. In this way, each representation can be kept simple and easy to understand while all representations together model a complex task world².

In the following sections, we discuss common graphical representations for task modeling. We discuss the strengths and weaknesses of the existing representations and we will then propose a collection of improved representations.

4.2.1 Common representations

Many representations already exist for task modeling as well as other related modeling activities. Not all of them are useful in practice and the question is what makes a representation useful and usable. One aspect of a representation is that it should be *effective*. In Macinlay Macinlay (1986) defines the effectiveness of a visual language as “*whether a language exploits the capabilities of the output medium and the human visual system*”. This notion can be expanded to include *purpose* and *audience* i.e. what is the representation intended for and who is going to use it, because “*visualizations are not useful without insight about their use, about their significance and limitations*” (Petre et al. 1997). Developing usable diagram techniques is difficult and requires insight in all of these aspects. In fact, one could say that usability is just as important for graphical representations as it is for user interfaces, both depending strongly on the context of use. Most of the research in this area is the field of *visualization* (Card et al. 1999) or *visual design* (Tufte 1990, Tufte 1983).

If we want to compare representations, we must first distinguish several purposes for which they can be used and by whom (Britton & Jones 1999). Within task analysis the purposes of representations typically include:

1. To document and to communicate knowledge between designers.

²In this chapter, we focus on task modeling techniques. The relationships with other modeling techniques such as used in OODA, Catalysis, RUP etc are beyond the scope of this thesis

2. To analyze work and to find bottlenecks and opportunities.
3. To structure thinking for individual designers.
4. To discuss aspects of the task world within the design team.
5. To propose changes or additions within the design team.
6. To compare alternatives in the design team or with a client.

Additionally we need some aspects that help discussing and comparing representations. For this discussion we will take the position that a representation essentially is *a mapping of concepts and relationships (and possibly attributes) to the visual domain*. Some aspects may concern the concepts and relationships while others concern the appropriateness of the mapping in relation to the purpose and audience. For discussing common representations we will use the following aspects:

- **Intended Purpose.** For what purpose is the representation intended? Certain representations work well for communicating with clients while others only help structure a single designer's thought. Similarly, certain representations focus on time while others focus on structure. Effectiveness is reduced when the representation does not support the purpose.
- **Coverage.** What concepts and relationships are involved? What information is shown and what is not? Is the information suitable for task analysis purposes? The information covered by a representation determines the view it supports.
- **Complexity.** What is the complexity of the representations in terms of the number of concepts and relationships that are shown? If the complexity is high the understandability is usually low, which is probably not desirable.
- **Understandability.** How well can the representation be understood? Understandability concerns how successful the concepts and relationships have been mapped to a graphical representation. Representations should be easy to understand for the intended audience/users. If not, they will not be used. Stakeholders may come from different disciplines which makes a common understanding more difficult to reach. Other aspects such as visibility also play a large role i.e. how easy parts of the representations can be distinguished or what kind of first impression a representation gives.
- **Intended Audience.** Who is going to use the representation? Certain representations are more familiar to designers from different disciplines than others. Also clients and other stakeholders may be familiar with certain representations. For example, UML is familiar to most Software Engineers while unfamiliar to ethnographers.

In the following sections we will use these aspects in the discussion of the different common representations that are used for task modeling: graphical trees, Universal Modeling Language, and Contextual Modeling.

Traditional Task Trees

Traditional task models mainly use task tree structures in some flavor. The task trees are intended to show the structure of the work in terms of tasks, goals and actions. Usually, some time information is added as well. HTA (Annett & Duncan 1967) uses trees with plans while most others use trees with constructors for time constraints. Some of the older methods do not even use graphical representations and use long textual descriptions e.g. GOMS (Card et al. 1983). Such descriptions become highly unreadable when the size of the task model is larger than just a few tasks. The indentation of task is not sufficient for large scale models and therefore graphical trees are an effective improvement. In the graphical representation, visual labels (shapes or icons etc.) add meaning to certain parts which make it much easier to distinguish the different parts (goal, task(type) or action) of the diagram. For example, in ConcurTaskTrees (Paternò et al. 1997) the task trees are combined with icons for task types and LOTOS-based (ISO 1988) operators that allow some more elaborate time semantics. In ConcurTaskTrees, tasks are of a certain type (abstract, user, machine) which is reflected in the icon that represents the task, see Figure 4.1. The figure shows that even when the labels are not readable, the different task types can still be distinguished.

Most graphical representations depict a task tree from top to bottom. However, drawing the tree from left to right makes better use of the drawing area since trees are usually much wider than they are deep, see figure 4.2. This issue becomes relevant when depicting large task models of more than ~ 25 tasks.

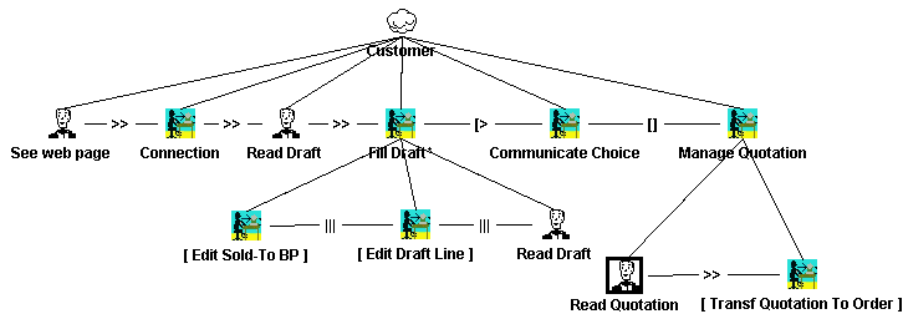


Figure 4.1: ConcurTaskTree example

Task trees are generally easy to understand and build, although they are usually built without software tool support. Tool support is slowly beginning to appear although still not commercially. Using sticky notes on a blackboard, a tree can be (re)structured easily but for documentation purposes the trees are (re)drawn manually (i.e. using a drawing package). Task trees are well-suited for communication purposes within the design team and to a certain extent also for communicating with domain experts. For the latter case the trees should not be too big.

Although task trees can be powerful, they are mainly based on the *subtask* relationship between tasks. Additionally, some information could be given about the ordering of

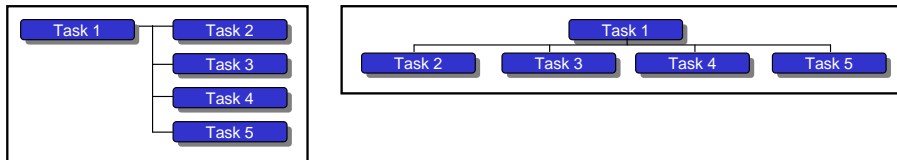


Figure 4.2: Depicting left-to-right versus top-to-bottom

tasks but no information about roles, actors or objects is given. Still a fundamental problem is that the possibilities for specifying time relationships remains problematic, see chapter 3.

Templates

Templates are a common way to represent concept properties. A template is a form and consists of fields for every property. Templates typically contain a mixture of properties and some relationships. In MAD and GTA, the template is used frequently to describe the task properties and relationships to mostly the parent task and roles.

A template is a simple and easily understood representation but it is only useful to represent detailed information. A large set of templates is very difficult to understand, even by experts. A template focusses on one concept and hence the relationships with other concepts are largely lost. Because of the lack of overview, templates are mostly used as a reference to detailed information of single concepts.

The Universal Modeling Language (UML)

In software engineering, UML (Rumbaugh et al. 1997) is one of the most influential modeling languages at present. It grew out of the intent to standardize the models that were used in Object Oriented Analysis and Design. Currently it is widely accepted in industry. UML was not designed with task modeling in mind, nor does it have an explicit ontology as a foundation. In UML, each diagram is defined both syntactically and semantically. Usage of terms between diagrams has been kept as consistent as possible.

Although UML was not designed for the purpose of task modeling, several diagrams can certainly be used in a task analysis context. The question is whether it would be useful to standardize on certain UML diagrams for task modeling. Since UML is a standard and many tools exist there are clearly benefits. However, if UML is used for task modeling the interpretation needs to be changed slightly. For example, states in an activity model now become tasks and objects now become roles³. UML has four representations that are directly relevant for task modeling:

³For the discussion of the different diagrams, this adaptation is assumed.

1. **The Activity Diagram.** This diagram can be used to describe the task flow in relation to events, roles and goals. Typically an activity is triggered by an event. As soon as a task is started, a goal consequently gets 'activated'.

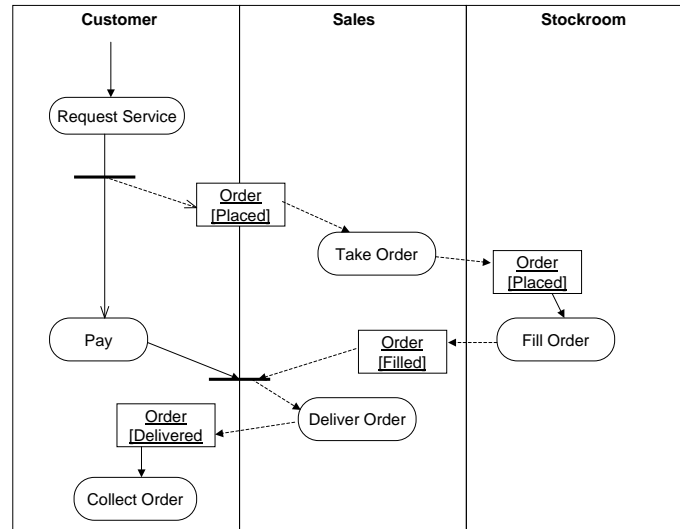


Figure 4.3: UML's Activity Diagram

2. **The Collaboration Diagram.** This diagram gives insight into the way different objects work together. The arrows show which roles communicate or work together in the exchange of objects or messages.

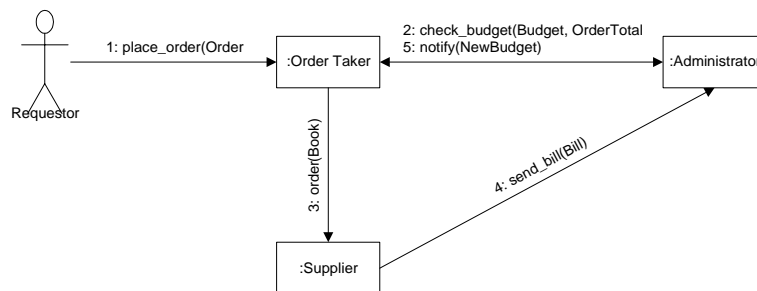


Figure 4.4: UML's Collaboration Diagram

3. **The Sequence Diagram.** The sequence diagram can show the sequence of tasks as they are performed between roles. Originally they are used to model method calls to objects but essentially there is no difference. Problems arise when calls are conditional or optional. Parallelism can also be modeled to a certain extent.
4. **The Use Case Diagram.** This diagram can be used to describe what is also known as scenarios. The exact difference is an ongoing dispute in the HCI com-

munity but at least they are both used to describe a particular set of tasks in a specified context. This representation is very informal.

Considering that the collaboration diagram and activity diagram are so related to each other, usage can effectively be restricted to using the activity view with swim lanes, i.e. no information would be lost since the collaboration diagram contains less information. The sequence diagram is probably less interesting for task modeling because of the problems with conditional and optional paths. Additionally, the method calls that are interesting in the object oriented sense have no important equivalent in task modeling. At most they could say something about “how” a task is started (by yelling or whispering commands?).

The use case diagram is useful for task modeling although there is no clear view on the differences between a use case and a scenario. One definition could be that a use case describes a specific “path” through a task tree under specified conditions. A scenario can then be defined as a more general description that also sets the context for a use case.

Using UML diagrams has the advantage that Software Engineers are familiar with them but other disciplines in the design team usually do not know them. The diagrams are fairly powerful but would require a small adaptation for task modeling purposes. Additionally, many software tools exist that support the designers to create UML diagrams.

Contextual Modeling

Contextual Modeling is part of the Contextual Design (Beyer & Holtzblatt 1998) method and consists of five *work models* to describe the task world. The models are built to describe work from the point of view of *one* person and they are not intended to represent everything that a person does. The five different views are:

1. **The Flow Model.** Represents the communication and coordination necessary to do the work.
2. **The Sequence Model.** Shows the detailed work steps necessary to achieve an intent.
3. **The Artifact Model.** Shows the physical things created to support the work along with their structure, usage, and intent.
4. **The Cultural Model.** Represents constraints on the work caused by policy, culture, or values.
5. **The Physical Model.** Show the physical structure of the work environment as it affects the work (objects and their locations).

These models as they are introduced are not entirely new. The authors claim that these representations have been tuned over time and are sufficient in most design cases. This

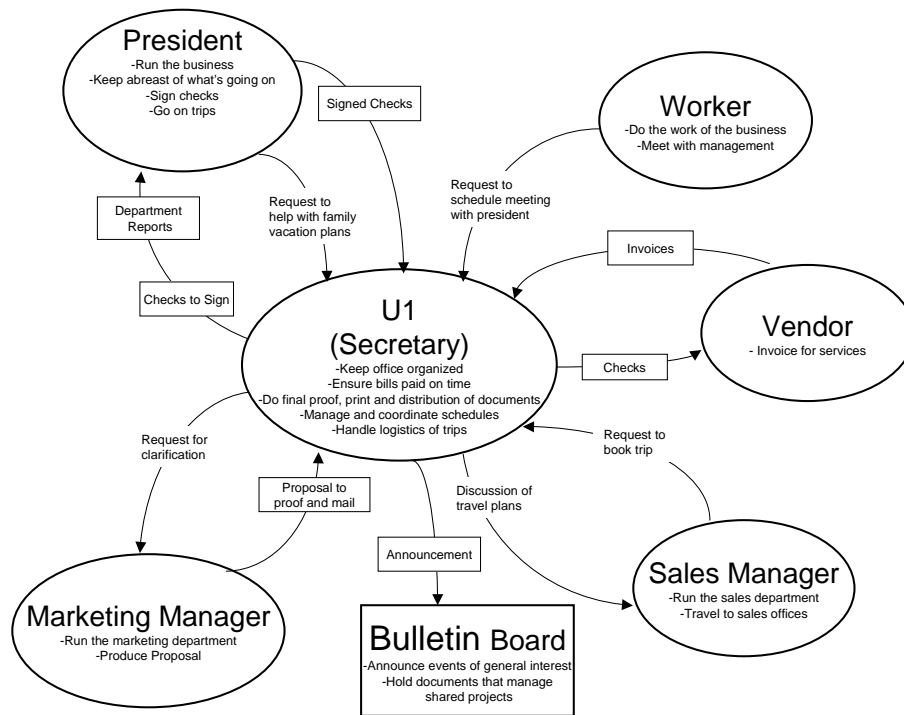


Figure 4.5: The Flow Model from Contextual Design

is questionable because none of the representations allows hierarchical building of representations. For instance, the Sequence Model is a linear sequence of tasks without the possibility of defining subtasks, choices, plans or strategies. Other representations such as the Flow Model are almost exactly the same as UML's Collaboration diagram, although a different notation is used. The Artifact model and Physical model are basically annotated drawings and not structured models. Even though the individual representations are not that new or special, the idea of using these "views" to describe work was not previously stated as such.

The contextual models use a somewhat different terminology than is commonly used. They speak about roles, tasks, and artifacts but they also use *intent* to indicate goals. Additionally, they speak about "triggers" as events that start tasks. Events are commonly found in work flow or process modeling but are somehow not often used in task modeling.

Contextual Modeling has shown that it is important to look at the work from different perspectives and work that out into practical models. Many have argued for a multiple perspectives view on work but none have worked it out in such detail.

As the authors say in their book, the models usually occupy a whole wall. The problem with the Flow model and the Sequence model is that they only work for small design cases and do not scale very well to larger cases. Other problems are the undefined

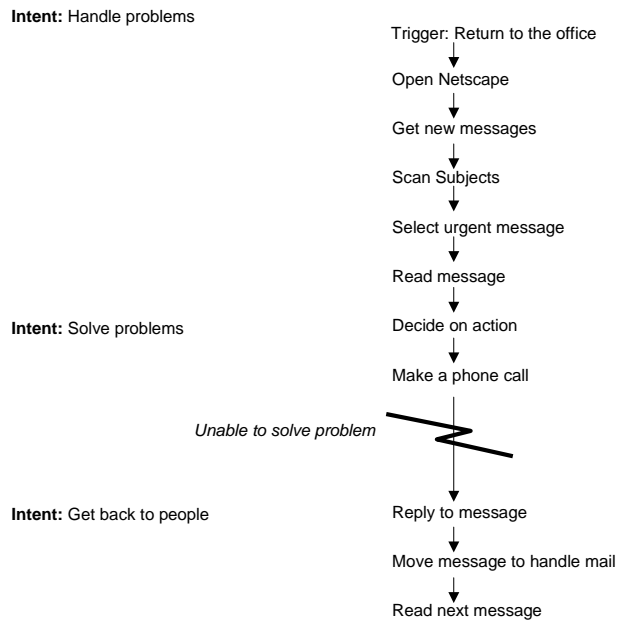


Figure 4.6: The Sequence Model from Contextual Design

semantics. The Flow model is actually a renamed UML collaboration model without distinguishing roles and actors.

Contextual Design also defines the process of gathering data and modeling steps. In time-boxed sessions the models are created and *consolidated* in a later session. Only in the consolidated versions are models worked out in detail. This illustrates that designers do not make an exact and consistent task model from the start but rather iterate and slowly consolidate the models.

4.2.2 A collection of ontology-based representations

In the previous sections, several common representations have been discussed. It is clear that some are more useful/usable than others and that improvements can be made. In this section, we define a collection of improved representations that cover the views as defined in chapter 3. This collection of coherent representations is an attempt to provide a more useful collection of representations for practitioners. The following views are covered:

- Work Structure
- Work Flow
- Work Artifacts

- Work Environment

For each of the views we will define one or more representations that form a useful “package” for that view. Together, the representations can form a practical tool set for the designer. The representations are based on existing representations but include some additions or modifications to make them more usable and useful for task modeling.

Constructing a set of Representations

The collection of representations that is discussed in the next sections combines several existing representations. Additionally, some modifications have been made. Compared to Contextual Modeling (CM), the main differences are:

- The CM sequence model is replaced by a work flow model similar to the UML Activity diagram.
- The CM sequence and CM flow model are combined into one representation.
- Decomposition trees are added.
- The CM cultural model has been redesigned.
- The number of concepts is larger than in CM.

Compared to UML, we use a modification of the Activity Model. We have added and event and goals lane as well as changed representations for parallelism and choice.

Modeling the Work Structure

The purpose of the work structure model is to represent how people divide their work into smaller meaningful pieces in order to achieve certain goals. Knowing the structure of work allows the designers to understand how people think about their work, to see where problems arise and how tasks are related to the user’s goals. The relation between tasks and goals helps the designers to choose which tasks need to be supported by the system and why i.e. which user goals are independent of the technology used.

For modeling work structure the task decomposition tree has proven to be useful and usable in practice. The tree is essentially based on the *subtask* relationship between tasks. Besides tasks, goals can also be incorporated. At the highest level a tree can start with a goal and subgoals and then proceed with tasks and subtasks, see Figure 4.7. In that case the *subgoal* and *has* relationship are also used. A task decomposition is modeled from the viewpoint of one role or goal. If complex systems are modeled, several task trees are needed to describe the work for all the roles. It then becomes difficult to see how work is interleaved.

Trees normally contain a time ordering using constructors from top to bottom or left to right, depending on the way the tree is drawn. The inclusion of time information can be

insightful but it is often also problematic as discussed in section 3.4. ConcurTaskTrees use operators based on LOTOS (ISO 1988) which are probably the best defined time operators. On the other hand, it is not always necessary to be very precise in everything that is modeled. Designers will also typically model that certain tasks occur *sometimes* or *almost never*. In our opinion, including some time information is useful but this kind of information is better represented in a work flow model if precision is required.

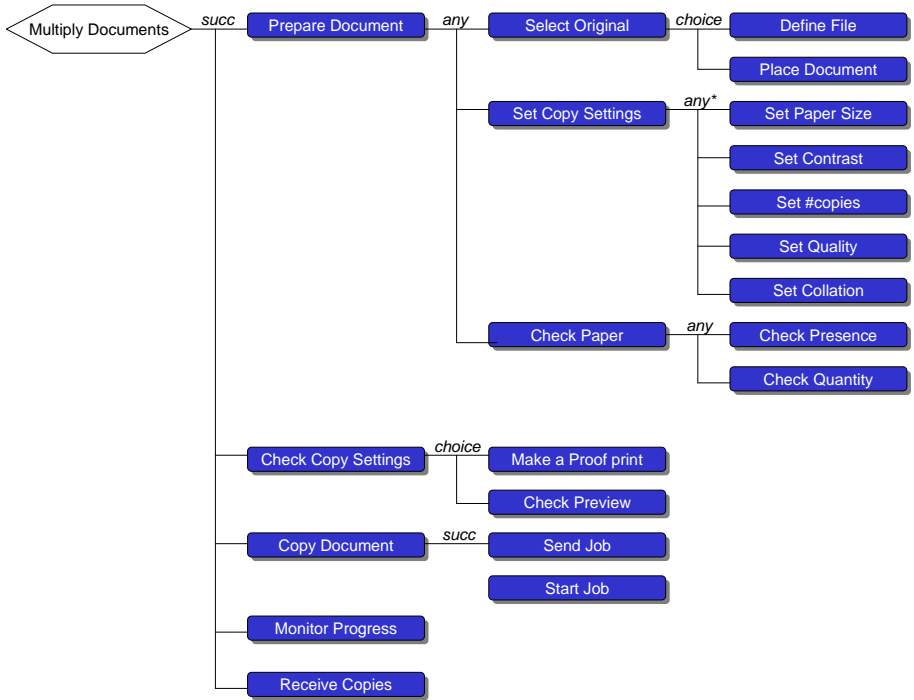


Figure 4.7: The Work Structure Model

If time is included, then a number of time operators are plausible. In our experience, it is useful to have a set of standard operators while also allowing designers to create their own operators when needed. For the average usage, the following time relationships have proven sufficient:

- **Concurrent.** The tasks occur concurrently.
- **Choice.** One out of a set of tasks is done.
- **AnyOrder.** All tasks of a set of tasks are done in no fixed order.
- **Successive.** One task is followed by another.
- **Any.** Zero or more tasks of a set of tasks are done in no fixed order.
- *** combined with other constructors.** Used to express iteration.

In the work structure model, the root of the tree is a goal with possibly some subgoals. Connected to goals are tasks which are represented as rounded rectangles. The tree is drawn from left to right instead of top-to-bottom for more economical use of space, especially when trees become large.

Other aspects of work structure include role structures and the relationships with tasks. For role structures trees can also be used. When used to show goal or role hierarchies the time constructors are not used.

Modeling the Work Flow

The purpose of the work flow model is to show work in relation to time and roles. The model gives the designer insight in the order in which tasks are performed and how different people are involved in them. Additionally, it can show how people work together and communicate by exchanging objects or messages. Typically, a flow model describes a small scenario involving one or more roles. This way, it shows how work is interleaved.

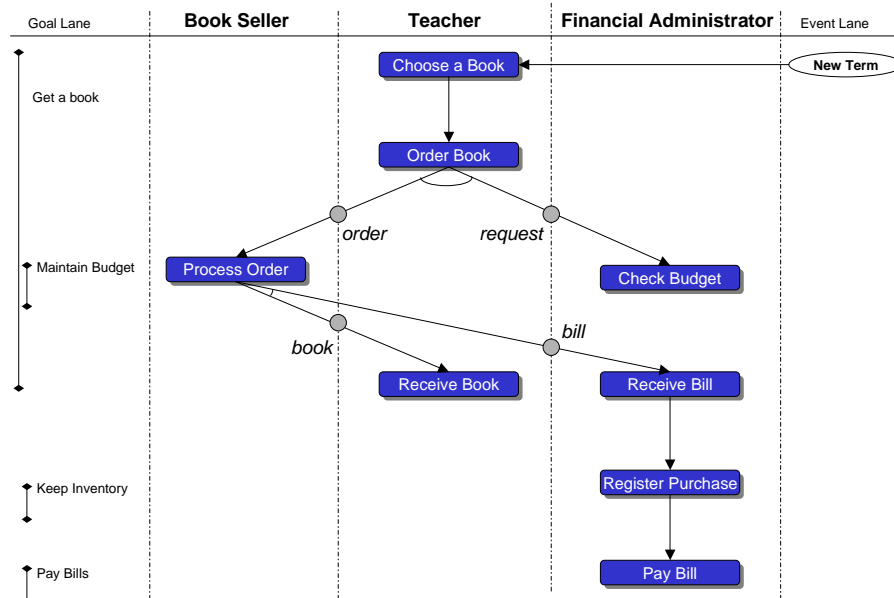


Figure 4.8: The Work Flow Model

The flow model specified here is a variation on the UML Activity graph. We included events and goals to make it more suitable for task analysis. Additionally, the representations of the time operators have been modified to be more appealing. This way the collaboration diagram (or Contextual Design's Flow Model) is not needed anymore since the information has been combined in one representation. Each flow model describes a scenario that is triggered by an event, see Figure 4.8. Work usually does not

start by itself but instead is often highly event driven (van der Veer et al. 1997). The event is represented by an oval which is connected to the first task. The sequence of tasks is given using a **Concurrent** operator or a **Choice** operator and not any of the other operators as suggested for the structure model. The concurrent operator is represented by an additional arc while the absence of the arc indicates the choice operator. Tasks can optionally be arranged in *swim lanes*, one for each role. Objects can be passed between tasks that have different roles and are drawn on the border of the adjacent swim lanes. When needed, goals can also be added to this representation. With a certain task a new goal can get “activated” until it is “reached” in a later task. The goals are written in the first column with vertical lines to show how long they are activated.

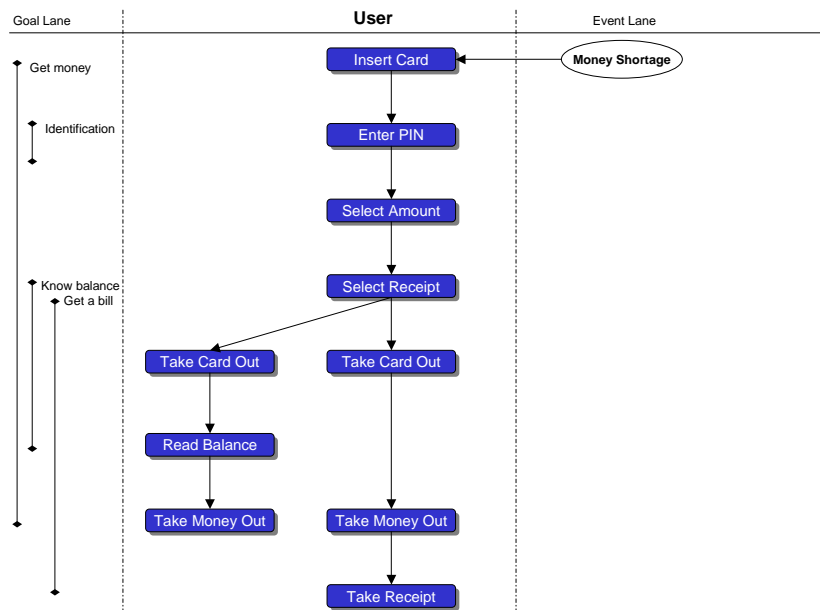


Figure 4.9: A Flow Model of a Dutch ATM

The flow model does not show hierarchical relationships between tasks and a flow model can only use tasks that are hierarchically on the same level. For subtasks, a new flow model needs to be specified. The addition of the goal lane can show many useful aspects when analyzing the work flow. For example, Figure 4.8 shows that once the “teacher” has received the book his goal is achieved but the scenario is not finished yet. Figure 4.9 describes the use of a typical Dutch ATM and shows why people often forget to take their receipts out. As soon as the primary goal has been reached, users lose interest in the remaining tasks. In this case, the task to take out the receipt is positioned *after* the users get the desired money. When the ATMs were first introduced the situation was even worse, the machines unfortunately gave back the card *after* the money had been dispensed. Since the user had already achieved the goal the user was much less interested in the remaining tasks and people consequently forgot to take out their bank card.

In terms of the ontology, the flow model is based on the concepts Event, Task, Object and Role. The relationships used are *triggers*, *responsible*, and *uses*. The operators are **Concurrent**, **Choice**, and **Successive** which are parameters in the *triggers* relationship. The **AnyOrder** and **Any** constructors are not valid in this representation and the **Successive** operator is implicit in the direction of the arrows. For objects that are being passed between roles it holds that each object must be associated to both tasks with the *uses* relationship. Note that the objects that are used in one task are not shown in the representations. For example, the PIN card itself is not shown in Figure 4.9.

Iteration is not specified in the flow diagram. If a task is done several times, an asterisk can be used to indicate that the task and its subtasks are done several times. However, usually iterations are specified in the Work structure model. Iteration is specified on subtasks and not tasks on the same level, which are shown in the Work Flow model.

Modeling the Work Artifacts

The artifact model shows two relationships between objects: the *containment* and *type* relationship. For both a tree diagram is used. The objects themselves can be annotated with their attributes or their visual appearance. In order to express containment and type, the UML notation can be used, see Figure 4.11. However, it is important to remember that we are only modeling objects that are relevant to the user and not any irrelevant internal system objects. To some this may suggest that the task models describes an object oriented system model, which is *not* the case.

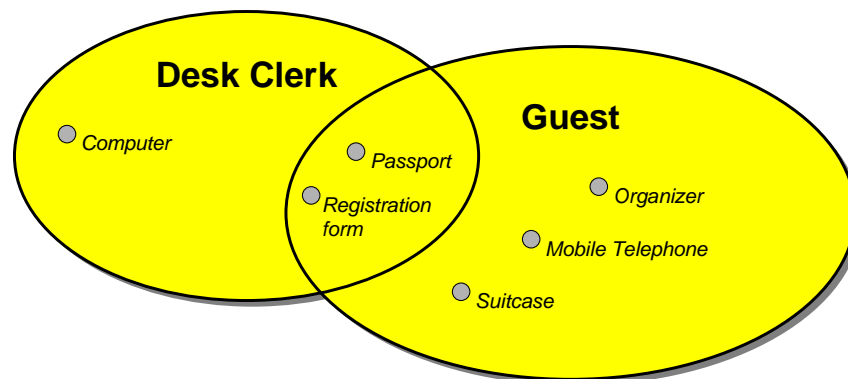


Figure 4.10: The Artifacts Model

The use of objects in tasks is partly covered by the Work Flow model. The Work Artifacts focusses on the structural aspects of the object. However, objects may also be connected to their users i.e. roles or agents, instead of the tasks where they are used. In such a diagram, the users are represented as ovals and the objects are labeled dots within the ovals. The ovals may overlap if more than one user uses the object, see Figure 4.10.

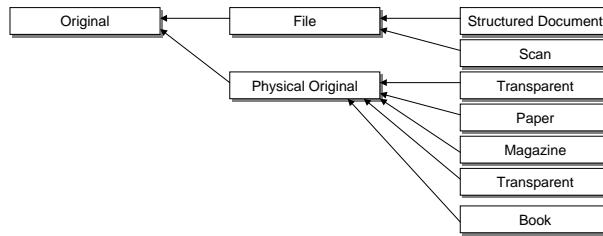


Figure 4.11: Example of a UML class diagram

Modeling the Work Environment

The environment model describes two aspects of the environment. Firstly the physical layout of the environment and secondly the culture within the environment. The **physical model** is simply described by one or more annotated "maps" of the environment. The purpose is to show where objects are located in relation to each other. The objects are those that are relevant for the work and also those who are in the same space. Figure 4.12 shows an example of a work place layout. Such layout diagrams can easily be drawn using commercial drawing software such as Visio (Visio Software 1999).

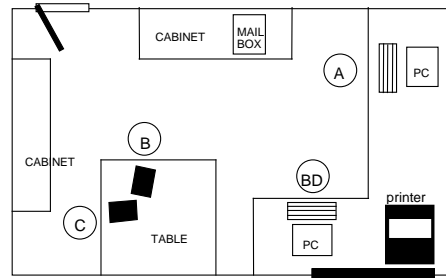


Figure 4.12: The Physical Layout Model

The other model is the **Culture Model**. The culture model we describe here is an adaptation of the culture model from Contextual Design. In Contextual Design the roles are represented in overlapping circles. However, overlapping of circles does not have any meaning although it suggests that there is one. Hence we adapted the model. We define the culture model as follows.

1. Roles are represented as ovals.
2. The ovals are connected by arrows if there is a **force** between roles. The relative strength of the force is depicted in the width of the arrow.
3. Forces are annotated with **attitudes** of the force relationship.

In some cases, a force applies to more than one role. By drawing an extra circle around roles, a force can indicate one-to-many forces which can typically be used to describe "corporate culture".

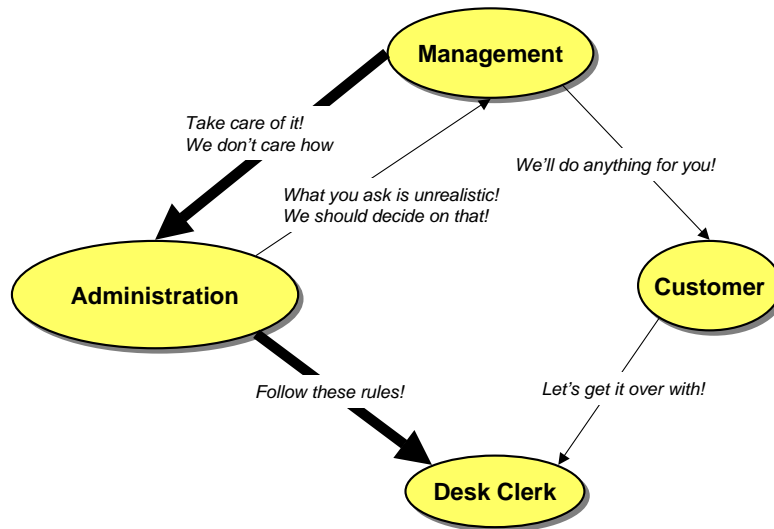


Figure 4.13: The Culture Model

4.3 Static versus Dynamic Representations

All of the representations discussed in the previous sections are static. However, representations can also be more dynamic. Traditionally, a representation is static i.e. it does not change after it is drawn and is designed for use on paper. However, it is often convenient to emphasize a certain aspect in the representation. When software is used to draw the representations, the representations can be changed dynamically. In (Card et al. 1999) they are called *active diagrams*. For example, one could easily switch between a flow model with or without swim lanes. Alternatively, it could be possible to add some extra information by *marking* tasks as "problematic" or "uses object X", see Figure 4.14. Such annotations are often done by designers to explain certain aspects to others during a presentation or in documentation. In software, we are already very much used to active diagrams and they occur in scrolling, zooming and syntax highlighting. This asks for a more flexible view on what constitutes a representation and when a representation can be modified. The dynamic aspects could be controlled manually by the viewer but could also be pre-specified using a function in which case we usually speak of animation. Now that it becomes increasingly more easy to create dynamic representations it is important to understand when and how they could be applied usefully in design.

In task modeling, animation is a way to create more dynamic representations. Ani-

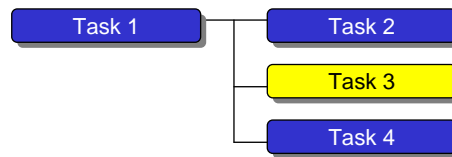


Figure 4.14: Coloring nodes to highlight particular tasks

mation can be used in simulations of scenarios or task models (Bomsdorf & Szwillus 1999a). Using simulations an analyst can step through a scenario and get a different feel for what goes on. Other purposes might be to “debug” a task model which is particularly useful for envisioned task models.

4.4 Analyzing the Task World

Representations help to represent the knowledge that is gained in the process of task modeling. During this process it is useful to analyze *what* is actually represented in the specifications. One aspect is to see if the specification correctly represents the knowledge and other aspects may focus on seeing the problems in the task world. For envisioned task models, it is important to make sure the specification is correct.

One frequent criticism of task analysis has always been the fact that it remained unclear what exactly to do with the data, “we have the data now what?” What should be done next is an analysis of the data in order to find problem areas and opportunities that relieve the problems in the task world. Those results then become the basis for designing an *envisioned task model*. Representations for envisioned task models can be largely the same as for current task models, only the interpretation is slightly different. Task analysis research has focused on data collection and modeling techniques but it has neglected research on data analysis. Naturally, during modeling activities some of the data is analyzed when models are being constructed and modified. However, much more structural analysis is possible, especially when the data is structured using the task world ontology (van Welie et al. 1998b).

When analyzing a current task model many problems can occur that differ from case to case. However, we found that many problems fall in the same categories and have a more general and domain independent nature.

- **Problems in individual task structures.** The task structure is sub-optimal because too many subtasks need to be done or certain tasks are too time-consuming or have a high frequency.
- **Differences between the formal and actual task performance.** In cooperative environments, usually regulations and work practices exist which are documented, for instance as part of ISO9000 compliance. In reality, tasks are mostly not performed exactly as described on paper and that “one way” of how the tasks

are done does not exist. When persons in a cooperative environment think differently about what needs to be done, problems arise.

- **Inefficient interaction in the organization.** Complex tasks usually have many people involved who need to communicate and interact for various reasons, such as knowledge about tasks or responsibility for tasks. This can be the cause for time-consuming tasks but also for irritation between interacting people.
- **Inconsistencies in tasks.** Tasks are defined but not performed by anyone or tasks are executed in contradictory order.
- **People are doing things they are not allowed to do.** In complex environments often people have a role that makes them responsible for tasks. Sometimes other roles actually perform tasks for which they did not get an official permission or they are using/changing objects they are not allowed to change.

Of course not all problems can ever be automatically detected. However using our model for describing task world models many characteristics can be detected semi-automatically by providing the analyst with a set of analysis primitives. Analyzing a work environment can be done when the data present in the model is transformed into qualitative information about the task world.

There are two ways of qualitative analysis. An analyst may search heuristically by looking at properties of a specification that might point the analyst to problems. Alternatively the data can be analyzed on a logic level by putting some constraints on the model, a form of verification. Constraints that do not hold may show interesting features of the task world. In the next sections, these ways of analysis will be elaborated and clarified with examples.

4.4.1 Heuristic model-based evaluation

In heuristic evaluation, we try to find out “what is going on” by looking at certain properties of the specification. The goal is to gain an understanding of the task world and to find the nature and causes of problems. Using a standard set of properties we can increase the chance of doing a successful analysis of the specification. For instance, looking at all tasks in which a certain role is involved may help to gain insight in the involvement of a role in the task structures.

Heuristic evaluation is done by checking specification properties that are not objectively right or wrong. It is up to the analyst’s interpretation whether they are reason for concern or not. These properties are more interesting for finding the actual *problems* in the task world. Using the ontology we can define a number of properties that deal with instantiations of the concepts of the ontology. Most properties concern the number of “links” between concepts, or the values of attributes. Possible properties related to problems are:

- The number of roles involved in a task.

- The rights a role or agent has for the objects used in the task they are responsible for or perform.
- The frequency of tasks.
- The frequency of events.
- The number of tasks a role is responsible for.
- The number of subroles a role has.
- The number of levels in subtasks of a task.
- The number of subtasks on the same level of a task.
- The objects used in a task.
- The roles involved in the task.
- The objects that are used by a certain role.
- Tasks that are delegated/mandated.

4.4.2 Model verification

Verification concerns only the model as it has been specified. Only a limited degree of verification of a task model can be supported due to the inherent lack of formal foundations for task models. There is no model to verify the task models against. However, it is possible to see if the task model satisfies certain domain independent constraints. The task world ontology merely defines the concepts and relationships without any constraints. This was done deliberately to give the analyst as much freedom as possible to specify what they find during data gathering. There are however constraints that we would like to be satisfied independent of the specific domain that is being studied. For example we would like that for each task there is at least one responsible role and that each task is really being performed by an agent. These constraints can be specified as logical predicates and can be checked automatically. Within model verification constraints we can distinguish constraints on *cardinality*, *type*, *attributes* and constraints *between* specifications.

Cardinality Constraints

Cardinality constraints concern the cardinalities of the relationships between the concepts. However, they have been defined irrespective of the specific study being done. They should hold in any domain. A task model where all constraints are obeyed may be considered "better" than one which does not obey all the constraints. In other words, the constraints allow us to denote classes of models which have an order of preference. Examples are:

- Each event should trigger at least one task
 $\forall e \exists t \{e \in Events, t \in Tasks \mid triggers(e, t)\}$
- Each agent should have at least one role
 $\forall a \exists r \{a \in Agents, r \in Roles \mid hasrole(a, r)\}$
- Each role should have at least one responsible task
 $\forall r \exists t \{r \in Roles, t \in Tasks \mid responsible(r, t)\}$
- Each object should be used in at least one task
 $\forall o \exists t \{o \in Objects, t \in Tasks \mid uses(t, o)\}$
- Each task should be performed by at least one role
 $\forall t \exists r \{t \in Tasks, r \in Roles \mid performs(t, r)\}$
- Each task should have at least one role that is responsible for it.
 $\forall t \exists r \{t \in Tasks, r \in Roles \mid responsible(r, t)\}$
- Each object should have an owner (someone with the *owner* right)
 $\forall o \exists r \{o \in Objects, r \in Roles \mid uses(o, r, Rights) \wedge owner \in Rights\}$

Type Constraints

These constraints deal with relationships between entities of the same type. These constraints are also of a general nature with the possible exceptions of the object constraints.

- An instance of an object cannot contain itself⁴
 $\neg \exists o \{o \in Objects \mid contains(o, o)\}$
- An object can not be of its own type
 $\neg \exists o \{o \in Objects \mid isa(o, o)\}$
- A task cannot have itself as a subtask
 $\neg \exists t \{t \in Tasks \mid subtask(t, t)\}$
- A task cannot trigger itself
 $\neg \exists t \{t \in Tasks \mid triggers(t, t)\}$
- A role cannot have itself as a subrole
 $\neg \exists r \{r \in Roles \mid subrole(r, r)\}$

⁴For classes of objects this can be allowed

Attribute Constraints

Other properties might be related to the *attributes*

- Missing goal attributes for the tasks that should have a goal.
- Check on empty conditions
- Objects with a specific attribute and/or value

If some of the attributes like duration and frequency are formally described, other properties could be checked as well. The question is if these properties make it worth to enforce a more formal specification of the attributes. In such a model, the following properties could be checked:

- The total duration of a task is less than or equal to the sum of durations of all subtasks. This holds only for sequential tasks that only have sequential subtasks.
- It is not desirable to have contradicting task sequence specifications. A after B after C and A after C after B at the same time. When analyzing a current task model this may be interesting to detect but in an envisioned task model it is undesirable.
- Conditions and states can be checked on their syntax. References to entities should be checked for existence.
- Taking all the formally defined conditions, will a certain task be executed? Under which conditions can the execution of a particular task occur (e.g. event Y needs to occur).

4.4.3 Comparing two specifications

The properties of the previous sections all concerned one specification. Another option is to compare two specifications in which case other properties are interesting. When comparing two specifications there are three ways to do so:

1. Comparing two current task models.
2. Comparing two envisioned task models.
3. Comparing a current task model with an envisioned task model.

In the last case, comparing specifications may say something about the design decisions taken when redesigning a task world. For example:

- Which tasks were reassigned to different roles?
- Which roles were reassigned to agents?

- Which tasks are removed or added?
- Which objects were added or removed?
- Which events are new and which have been removed?
- Which object rights have changed?
- Which task have become less complex?

In the ConcurTaskTree tool CTTE (Paternò 1999), a more statistical approach is taken and the designer can see differences in the number of abstract tasks, interaction tasks, objects and operators etc.

4.4.4 Model validation

Validation of *current* task models means checking if the task model corresponds with the task world it describes. In the process of validation one may find that certain tasks are missing or there are more conditions that are involved in executing a task. Often one finds that there are exceptions that had not been found in earlier knowledge elicitation. Consequently validation needs to be done in cooperation with persons from the task world and can not directly be automated by any tool. However, it is possible to assist in the validation process, for instance by generating scenarios automatically that can be used to confront the person from the task world. Such generated scenarios are in fact simulations of pieces of the task model. Recent work on early task model simulations (Bomsdorf & Szwillus 1999a) has shown promising examples of early simulations based on task models.

During a design process it is difficult to say when to stop doing task analysis. At a certain moment it may be considered adequate but later on in the design process new questions may arise that cause task models to be extended or revised e.g. more information is needed about object attributes or missing tasks are being discovered.

Validation of *envisioned* task models means checking whether the specified task models actually improve the task world. In this case, the model needs to be strict on aspects such as consistency. Techniques such as simulation can be very useful in the validation of envisioned task models.

4.5 Summary

This chapter discusses the "practical" side of task analysis. It discusses representations that designers can use as well as ways to analyze task models and represent the final results. Both topics are defined in relation to the task world ontology. The ontology defines the theoretical background for representations but representations are needed in practice. We propose a set of representations that together cover all the important aspects of the task world.

Besides task modeling, task analysis is also discussed from the ontological point of view. Analysis can focus on finding holes in incomplete specification but also for detecting problems. By focussing on specific aspects of a task model the designer can get a better understanding of the task model and the underlying problems or opportunities.

All together, it forms a complete "package" for task analysis and task modeling. For the practical designer there are many representations to choose from and the analysis primitives provide a practical start for analyzing task models. In the next chapter, we discuss the phase in task based user interface design where the actual technology is designed: *Detailed Design*. In the detailed design phase, an envisioned task model of the future situation is used to create an initial solution.

Chapter 5

Detailed Design

5.1 Introduction

In the previous chapters, task modeling and analysis are discussed. The main purpose of those activities is to gather as much relevant knowledge as possible about the users and their tasks. In the detailed design phase, this knowledge is used to design the user interface itself. This phase deals with the actual technology that is to be designed as far as relevant to the user.

As in many other methods, the user interface is created on the basis of an analysis. From the task and user data, an initial design is created which is then subjected to many incremental development cycles. Effective use of contextual data about the users and their tasks is crucial for the design of usable and useful systems. It gives designers the necessary knowledge to understand how users can be supported in their work. Both in the creation and evaluation activities, this knowledge plays an important role. In addition, the designers expertise and explicit design knowledge such as guidelines and patterns are used to create the user interface. This knowledge about what works and what does not is important, especially when resources such as time are limited but quality is still desired.

Designing the user interface means that many aspects such as functionality, dialog structure and presentational aspects need to be considered. The user interface is more than just some windows. This chapter discusses these aspects and explains how they are related to usability. Then user interface specification techniques are discussed along with their important role in the design process, both for communication and evaluation. Throughout the entire design process some form of usability evaluation is done. We discuss early evaluation techniques since those can already effectively aid in detecting usability problems. The possibilities and limitations of several techniques for early evaluation are discussed. All together, these methods and techniques aid in the systematic development of usable and useful interfaces based on contextual data.

5.2 The Gap between Analysis and Design

One of the difficult steps in the user interface design process is the transition from the analysis phase to the design phase. The results of the analysis phase are a detailed description of the problem domain and the identified areas for improvement that set the design goals (requirements) for the system. The purpose of the detailed design phase is to design a system that meets those design goals. The transition from analysis to an initial design is characterized by a combination of engineering and creativity in order to incorporate analysis results in a concrete design. This transition can not entirely be done by following a simple set of predefined steps and requires a certain amount of *creativity*. In UID literature, this transition is called the *gap* between analysis and design. The gap is concerned with questions like; what are the main displays? which data elements need to be represented and which are merely attributes? which interaction styles are appropriate? how should the user navigate through the interface structure? how will functionality be accessible? Besides the analysis results technological constraints and wishes of the client may complicate detailed design even further. However, in some occasions it may even be possible to create new technology that is needed for an optimal design solution.

In (Wood 1997) a number of methods and techniques are described that can be used to make this transition. In practice, bridging the gap means coming up with an initial design based on the analysis which then starts off an iterative development process. Naturally, the goal is to reduce the number of iterations in design by basing the initial design solution directly on the analysis.

5.3 Guidelines for Bridging the Gap

To overcome the difficulties of bridging the gap Dayton has developed a very concrete method called *The Bridge* (Dayton et al. 1998). It uses PANDA (Participation Analysis Design and Assessment) techniques in small teams of approximately 5 team members with at least one end user. The method consists of a number of steps in which an OO GUI prototype is constructed using a simple task model. The resulting GUI should be seen as an initial prototype which will need to be developed further in the following iterations. The basic steps are as follows:

1. Expressing User Requirements in Task Flows.
2. Mapping Task Flows to Task Objects
 - (a) Identify which task objects need to be included in the system.
 - (b) Identify the total set of relevant attributes of these task objects
 - (c) Identify relevant actions on the task objects. The actions can be ordered in menus.
 - (d) Identify groups of attributes so that only the task relevant task attributes are shown while performing a task. The resulting groups are called views.

- (e) Identify object containment relationships. These relationships need to be represented in dialog screens

3. Mapping Task Objects to GUI objects using a specific platform style.

The steps given by Dayton may not be appropriate for all types of systems. However, they may help in producing an initial solution which can then be elaborated in detail. Having an initial solution can be very important in the development process because it facilitates discussion and hence leads to new ideas. After all, the detailed design process needs to benefit from the designers' creativity. Dayton has developed his method for developing OO GUI systems as opposed to procedural systems.

Another set of guidelines is given by Mayhew (Mayhew 1999). She distinguishes two types of applications; *product* and *process* oriented applications. Product oriented applications (OO GUI systems according to Dayton) are applications where users individually create, change and store identifiable work products. Examples include word processors, spreadsheets and drawing applications but also ATMs and mobile telephones. On the other hand, process oriented applications (called procedural systems by Dayton) have no clearly identifiable work product and the main purpose is to support some work process. Examples are inventory tracking applications, financial management, and PDAs etc. Mayhew gives guidelines for both types of applications. In short her steps are;

1. Define the Conceptual Model as either product or process oriented.
2. Clearly identify products or processes.
3. Design presentation rules for products or processes.
4. Design rules for windows.
5. Identify major displays.
6. Define and design major navigational pathways.
7. Document alternative Conceptual Model Designs in sketches and explanatory notes.

Holzblatt describes another technique called *User Environment Design* that is part of the Contextual Design method (Beyer & Holtzblatt 1998). With UED the system is being structured using a mix of functional and object oriented focus. Using that technique the major displays are identified on the basis of the contextual analysis. From there on the system is again developed by iterative prototypes.

All of the guidelines above are based on identification of the major interface components, their structure and the navigational structure. The major interface components are directly derived from the task/object models built during analysis. The way the functionality is distributed over the UI components depends on the type of system that

is being built. The two main types mentioned (product vs. process) give a broad categorization but it may not always be easy to classify an application, for example interactive training applications have a bit of both. Although different types of applications can be distinguished, the high level process can be summarized as follows:

1. Develop an essential conceptual model of the task world. This model describes the task world without any reference to tools and systems being used.
2. Identify the major tasks and objects that need to be part of the system. These will become the high-level interface structure.
3. Depending on the type of application structure the application based on a process or product metaphor.
4. Create navigational paths in the interface structure depending on the task structure.
5. Design the presentation using a platform style.

After this short transition process, the iterative design activities are started to mature the system. The actual techniques used in these transition activities are reportedly very low-tech i.e. paper and pencil, sticky notes and flip charts for making all kinds of sketches. At this point in the design process the design solutions have the character of sketches and are hence informal. Nonetheless, using these sketches and paper prototypes already a lot of usability evaluation can be done, both internally and with future end users. Constant evaluation drives the design towards a more and more complete specification of a usable system.

5.4 Designing the User's Virtual Machine

User interface design consists of more than just designing some screens. Interfaces can become very complex and once an initial sketch of the interface exists many aspects need to be worked out in detail, including the interaction, the navigation structure and the system's behavior. We now take a closer look at the important aspects of the user interface. For the sake of the discussion, we use a different term that covers a broader range of aspects than is usually thought of when discussing the user interface. The term *User's Virtual Machine* (UVM) was introduced by Tauber (1988) and is used to indicate those aspects of a system that are relevant to the user. The user's attitude typically is "*Who cares what's inside?*". To the user, the interface *is* the system. The UVM is a useful concept to show which aspects of the user interface are important and hence need to be covered in the detailed design phase. These facets can be broken down following the Seeheim (Pfaff & ten Hagen 1985) model into:

- *Functionality Design*. The functionality as far as relevant to the users. Functionality includes the functional actions and objects that will be available to the user.

- *Dialog Design.* Structure of the interface without any reference to presentational aspects, the navigational structure and dynamic behavior of the interface.
- *Presentation Design.* The actual representation of the user interface including details such as layout, colors, sizes and typefaces.

All three activities are dependent on each other and they need to be kept consistent in order to form a coherent whole. Moreover, from a usability perspective there is also a forward dependency from functionality to presentation. If the functionality is not designed well enough the system will not be *useful* to users and therefore dialog and presentational aspects are irrelevant. In the same way, the dialog needs to be good enough before presentational aspects matter. However, for each system there will also be an emphasis on one of the three aspects because of the nature of the system. It makes a big difference whether a safety critical system or a mass-market consumer application is designed. This forward dependency may also offer an explanation for the fact that in practice, usability aspects are often not discussed until after the software design.

The UVM is user specific, or more precisely, a UVM belongs to *one* role, i.e. a role with the associate tasks/goals. Since systems are usually designed for multiple roles, several UVMs need to be designed. For the final design, these UVMs need to be integrated in order to design one system for all roles. This means that a design is always a compromise. Besides the different roles, certain user groups have specific needs concerning the dialog and presentation aspects, rather than concerning functionality. For example, elderly users may need larger font sizes or disabled users might need speech output. This also leads to a need for adaptable or adaptive interfaces. Adaptation should however never be used as an excuse for not making certain design decisions; "*let's just make it configurable*".

5.4.1 Designing the functionality

In task-based design, designing the functionality does not mean designing the application architecture including data models etc. It concerns the functionality as far it is relevant to the *user* and which is being presented to the user through the user interface. In performing tasks users execute functions and manipulate objects. How the objects and functions are internally represented in the system is a separate issue although often closely related to the way the user thinks about them. To perform tasks users usually perform certain actions on objects so when designing the functionality the designers have to choose which objects are part of the system, how they are structured or related, and what actions can be done on them. The tasks that are performed will provide the basis for high-level navigational paths expressed later in the interface structures. The functionality of the system needs to be well chosen in order to support the tasks from the task model in the most optimal way. For task-based design, this is where a large part of the contextual information about the task world needs to be used effectively.

Naturally, the functionality as relevant to the user needs to be supported by the system software architecture and certain aspects of the actual implementation are very relevant for the design of the user interface. For instance, functions may have side effects that

cannot be controlled by the user. Such functions can cause undesired or even dangerous situations in the real world that make them subject to extra care. In such cases, warnings or additional security measures may be appropriate. Vice versa, the functionality also influences the software architecture in the sense that specific functionality may pose certain constraints on the implementation. For instance, if multilevel Undo/Redo functionality is planned the software needs to maintain a command queue and each command has to be written in such a way that allows them to be undone and later redone. If this kind of functionality is not foreseen before the software architecture is defined, it may be very expensive to add it later. Other kinds of functionality that heavily influence the architecture include database functionality and real time behavior.

From a usability perspective the functionality needs to be well designed because otherwise the system cannot be *useful* for the user. Without the appropriate functionality the users cannot perform their tasks and will consequently not reach their goals. Ill-designed functionality can also confuse users and cause errors or lower performance. Even though presentation and dialog are also very important they can become completely irrelevant when the functionality is not designed well enough. In that sense, dialog and presentation depend on the functionality.

5.4.2 Designing the dialog

The dialog is concerned with the structure and dynamic behavior of the user interface without considering the exact presentation. It is not relevant on the dialog level what an interface component looks like but it is important to know if it is modal or not, what the content is and how the content is structured. Besides defining the major components of the user interface as a structure, the dynamics of the user interface need to be specified as well. For example, opening of one display may close another, actions of the user need specified feedback visually or auditory. It is mainly in the functionality and the dialog that provisions need to be made to make the user interface usable for all specified users. This can be achieved by providing shortcuts for advanced users, extra hints for novice users or separate sections for certain roles.¹

The term "dialog" already indicates the interplay between the user and the system. The user performs actions and the system responds. Interaction is not always of a simple stimulus-response type where the user always initiates interaction. The system can also take initiative and prompt the user for actions, for instance when new mail arrives or some kind of alarm goes off. Therefore the dialog is truly a two-way communication, both the system's behavior and the user's behavior need to be covered. However, it remains important that the users still feel to be in control.

The user's behavior includes the physical actions that are performed but also the mental actions that influence the user's performance. Mental actions take a considerable amount of time to perform and can make up a substantial portion of the total performance time of tasks. Shortening the time for mental actions can be facilitated by making the relevant information visible in the interface. The actual duration of the

¹Concerning accessibility, the provisions are usually made in the presentation.

mental or physical action depends on variables that are controlled by the designer. By providing appropriate navigational structure and presenting the relevant information according to the task, the duration can be considerably reduced.

From a usability perspective the dialog aspects influence a wide variety of usage indicators. Careful application of the usability means such as task conformance, flexibility, consistency etc, will greatly affect the dialog level and hence usability. All of these aspects must be tuned to get the desired level of usability. The functionality has to be right to make sure the systems is *useful* while the dialog contributes in making the system *usable*.

5.4.3 Designing the presentation

Presentation design is concerned with the way the user interface is presented to the users. The presentation are usually the most visible aspects of a user interface but auditory and tactile feedback are also communication techniques. Good presentation design helps to communicate where the user can find functionality, if it behaves as expected and how to control it. Colors, typefaces, layout, sound and tactile feedback, are import factors for the usability of a user interface. Well-chosen colors can prevent the users from making mistakes and make it easier to remember the semantics of interface objects. Typefaces have an impact on the readability of text both on screen and on paper (Kahn & Krysztof 1998). For example, on a screen sans serif typefaces are easier to read (because of the display resolution) than serif typefaces but for paper the opposite holds. Other typographic aspects such as size, rotation etc can create different effects on users. In the area of web-design, presentational aspects are an important part of the creating a "corporate image" and other branding aspects. Although this may not directly affect performance issues, it will influence the "experience" aspects of interaction which in turn influence user satisfaction.

Layout is another aspect that influences the users' performance. When interface objects are grouped meaningfully and laid out using a grid, users can find items faster because the information is more easily accessible and can be processed quicker (Fitts 1954). In (Mullet & Sano 1995) a large number of practical aspects of presentational design is discussed. These aspects include scale, contrast, proportion, organization, visual structure, grids, images, and style. Much knowledge in this area comes from industrial design and graphical design but also from ergonomics and cinematography (May & Barnard 1995). Not surprisingly, many designers have such a background while computer scientists rarely have this knowledge and skills.

Besides graphical presentation design (the visual channel), other output channels such as acoustics are also part of the presentation aspect. Often interaction is situated so that the hands and eyes are not available, for example during driving or for people with disabilities. In such cases, the other channels become very important. A newcomer in this area is haptic feedback, also known as force-feedback, which is often used in simulators.

From a usability perspective presentational aspects influence task performance times, chance of making errors and search times but also satisfaction. In (Tullis 1988) several

aspects of the presentation design are investigated with respect to their impact on user performance.

5.5 Cognitive aspects in UVM Design

In UID literature the cognitive aspects in dialog design have been widely recognized but this recognition has not been reflected in most common specification techniques. Exceptions are Mental Cognition Action tables (Sharratt 1990) and GOMS (Kieras 1996). Both categorize and use a set of mental actions so that it can be specified which mental actions the user needs to perform in order to complete his task. Specifying these actions gives insight in cognitive complexity involved in the task. Relevant questions for design are; which information should the user retrieve from memory? How much information does the user need to remember? How are search times affected by the presentational aspects?

Counting the number of mental actions and adding up their score would allow for some *comparative* analysis. GOMS even claims to make *absolute* predictions by calculating the time needed to perform the tasks. GOMS has a **M**(ental) operator that takes about 1.2 s. Examples of such mental operators are *decide* and memory related operators such as *recall* and *store*. Other mental operators can be defined by users/analysts and their duration may be different. The cognitive load of the specific operators may be dependent on their operands such as the number of items to choose from. In general, the mental load of an interaction largely depends on the number of objects that are involved in the interaction in relation to the user's working memory. During design this number can be influenced, for example by making relevant information available where it is needed thus reducing the need to remember objects. This issue is addressed by techniques such as CCT (Kieras & Polson 1985) and TAG (Payne & Green 1989). Based on (Card et al. 1983) and our own experiences we have defined the following list of mental actions:

- *find*(Item, List, Ordered?, Location) where Location can be *on screen*, *situation*, *ltm*, *wm*
- *choose*(Item, List, Ordered?, Location)
- *compare*(value1,value2, Location1, Location2)
- *compute*(Formula)
- *transform*(Input,Output)
- *memorize*(List, Memory, Ordered?) where Memory can be long or short term memory *ltm*, *wm*
- *forget*(List) This is the inverse of memorize and indicates that objects are not needed anymore.

- *recall*(List)

In comparison to *physical* actions it is important to note that mental actions take a considerable amount of time. This means that if only physical actions are modeled the predicted performance time may have a large error because of numerous mental actions that are involved. If designers do not recognize this, they might try to optimize only the physical actions which may still not solve the problem significantly.

Another cognitive issue directly related to usability is learnability. In learning all the aspects of interaction users make all kinds of decisions while doing tasks. CCT tries to describe the interaction in the form decision rules. The hypothesis is that the complexity of the rules influences performance and that the number of different rules influences the learnability. This is why it is usually good to strive for consistency in the user interface especially for similar tasks.

5.6 Specifying the User Interface

For several reasons the user interface needs to be specified. The possible forms can be anything from sketches to complete formal specifications. Reasons for specifying the user interface may include:

- *Communication* The design activities are done in groups and results need to be communicated throughout the group.
- *Evaluation* Actual prototypes or the actual system need to be evaluated against the intended behavior or requirements.
- *Creating Prototypes* Specifications can drive prototypes either manually or automatically.

Each of these purposes pose constraints on the suitability of the specification technique. For instance, when properties need to be proven it would be difficult to do so with sketches but easier with a formal specification. In the next sections, we briefly look at formal and informal techniques for specifying user interfaces.

5.6.1 Informal methods for detailed design

The most used techniques for detailed design are still the informal techniques. Sketching screens, windows or displays is the main method, either by manual drawing or using an interface builder. Such methods are excellent in the early exploration phases when the high level interaction needs to be created. Figure 5.1 shows a sketch that was made in the Seibersdorf case study, see chapter 9. It shows a sketch for the main display of a security system. Not many details are present but ideas for the main areas in the display are already visible.

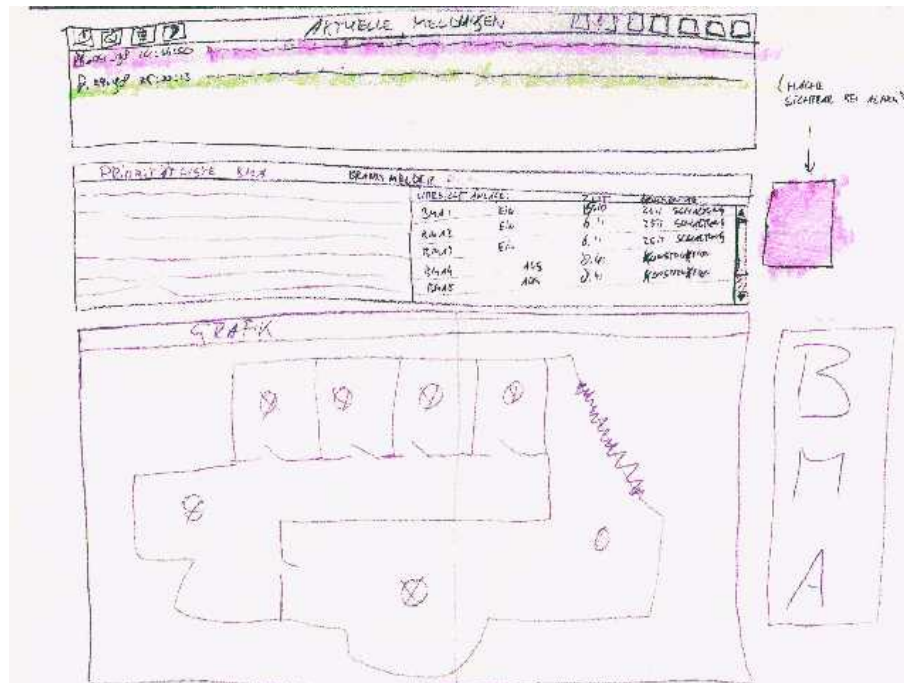


Figure 5.1: A sketch of a new system

Sketching is quick and does not require that details are exactly known. The main structural aspects can be drawn and they can almost be evaluated “on the spot”. Another big advantage is that no prior knowledge is needed to understand sketches which makes them very accessible. Besides sketching with paper and pencil, other tools also allow “sketching”. For example, chapter 7 describes story board tools which can be used for sketching. Interface builders also allow rough screen designs to be created rapidly. Screens can be printed out and put together as a paper prototype. Such designs are often called “paper mockups”.

Designing the hardware can also be done by “sketch” prototypes. Figure 5.2 shows some hardware sketches for a hand-held device. Such low-fidelity examples are very useful to quickly show and evaluate design ideas. In this case, size, weight and texture were explored.

Sketching techniques are good for the creative process because they allow a lot of freedom. It allows freedom in interpretation and even creative additions by colleagues, users or clients. They can simply take a pencil and change the design. A sketch clearly shows its status; it is unfinished, just a conceptual proposal and is still changeable.

However, when ideas start to stabilize the need for more precision increases. Sketches evolve in detailed and exact drawings showing all the visible details. Accompanying text usually specifies the interaction details. Such descriptions can contain a lot of text and easily leave some aspects unmentioned. Using natural language usually leads to



Figure 5.2: Sketching hardware

incomplete and imprecise descriptions.

5.6.2 Formal specification techniques

This section gives an overview of the most influential dialog specification techniques currently available. The overview is not meant to be exhaustive but is meant to be representative for what dialog specification techniques have to offer. Formal specification techniques usually offer precision and great expressive power. A disadvantage is that most techniques do not scale very well; they can only be used for describing portions or simple systems. Large specifications tend to become very hard to understand.

ConcurTaskTrees

ConcurTaskTrees (Paternò et al. 1997) (CTT) is a specification technique developed by Fabio Paternò and colleagues. The CTT's are task decomposition trees combined with a set operators, based on the LOTOS (ISO 1988) operators, to specify time constraints between tasks. Time constraints can be defined for tasks on the same level and also for subtasks. In addition, tasks are typed and are always one out of the possibilities; user, computer, interaction, or abstract. Basically the task types are used to indicate the "type" of entity that performs the task. When the type is "abstract" it means that it cannot be determined uniquely which is usually for all tasks that have more than one type of subtask. Besides tasks, roles and objects are also modeled. This was a later addition enabling to describe systems with more than one user. Usually, when CTTs are used they are *prescriptive* task models. They model the system and user interaction

with a task model. In terms of our philosophy this is the future task model or task model 2.

The CTT specification technique was essentially intended for describing the system and lately also for model-based design approaches (Paternò 1999). It is also part of a method called Task Lotos Interactor Models (TLIM). Construction of CTT diagrams is supported by a tool. Withing the TLIM approach other tools are also available for certain activities. In chapter 7 the different tools are described in more detail.

State Transition Diagrams

State Transition Diagrams (Dix et al. 1998) are very old but still often used for the specification of user interfaces. The interface is described by labeled state and actions that cause state changes. Figure 5.3 shows a small example from a drawing application.

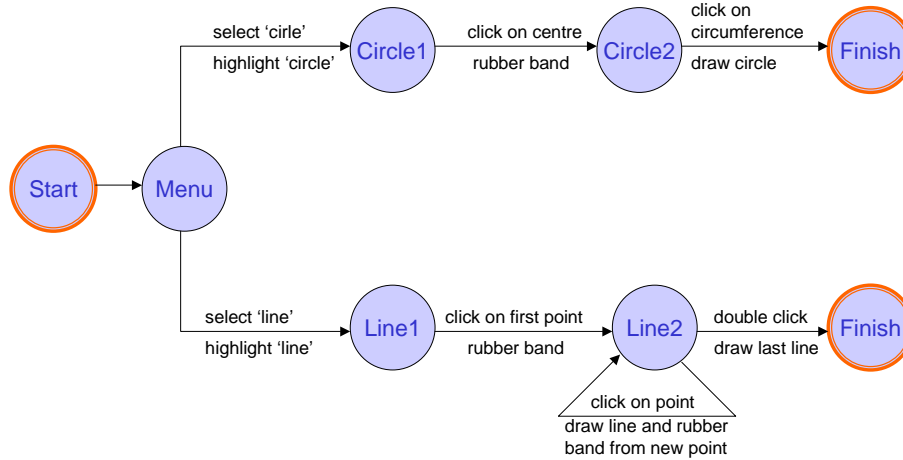


Figure 5.3: An example of a State Transition Diagram

For small examples this technique works fine but for real applications the number of states increases exponentially while not all states are equally important to describe. Finding meaningful names for each state becomes increasingly difficult. Using a hierarchical variant may help coping with increasing complexity but at the same time it makes the diagrams more difficult to understand.

STDs are system oriented instead of task oriented. Internally systems work with states but users are goal directed. When going from a task model to a detailed design specification, a STD is not the most appropriate technique.

Extended Task Action Grammer

ETAG (Tauber 1990, de Haan 2000) is another technique to specify the user interface. It is a task centered specification technique that describes both the interaction from a

task perspective and the interface structure itself. From a different viewpoint it can also be seen as a technique to describe the knowledge users need for interaction and how it is presented in the dialog. Therefore, it is not a technique for modeling real users. ETAG describes the lexical, syntactic and semantic aspects of a user interface. However, it does not address the presentation aspects. In Payne & Green (1989) Payne states about TAG *"as far as TAG is concerned the screen could be turned off"* which is also directly applicable to ETAG. ETAG uses a propositional representation for representing knowledge. A specification is built up using *sentences*. In addition ETAG uses feature grammars which allow *rules* to be defined. Such rules directly relate to learnability aspects. In an ETAG specification both the objects of the user interface are defined as well as the tasks that can be performed on them by users. Although the notation is very powerful it is usually too complex to use. It requires considerable skill to master the technique.

User Action Notation

User Action Notation (Hix & Hartson 1998) was developed out of the need of communication between implementers and designers. The technique consists of two types of diagrams; interaction templates and composite templates. Interaction templates are used for describing the actual interaction in detail using four columns (user action, system feedback, interface state and connection to computation). The composite template is used to describe a hierarchical decomposition of interaction templates.

UAN focusses on the interaction between user and the system instead of the system's states as STDs do. This makes UAN more suitable for task-based design. Compared to ETAG there are a lot of similarities but the table format of UAN is much easier to understand than the ETAG sentences. However, coping with increased complexity is also a problem for UAN tables. One table can already describe complex interaction but it is not uncommon that several dozens of tables are needed to describe a realistic application. Understanding such a large number of tables becomes problematic.

5.7 NUAN: New User Action Notation

In this section, we describe the New User Action Notation (NUAN). It is a variation of User Action Notation (Hix & Hartson 1998) and intended to overcome some of the problems we encountered when applying UAN. The main problems concern the specification of event-driven interfaces and mental aspects during interaction. Nowadays, systems are highly event-driven which cannot be well described in the original UAN. One small change is that in NUAN the cryptic shortcuts are replaced by short statements². Although this notation is less compact it was easier to understand by the students that used it, see appendix A for a complete list of the commands. Each of the following sections describe one of the other extensions that have been made.

²Instead of `~[fileicon]Mv^` we write `POINTERTO(fileicon), CLICK(button1)`

NUAN is a task-based technique and we have defined the relationships with a task model. Figure 5.4 shows the relationship between the task world ontology and NUAN. The NUAN ontology shows that the task modeled is continued from the point of basic tasks. The events and object are representatives of their equals in the task model.

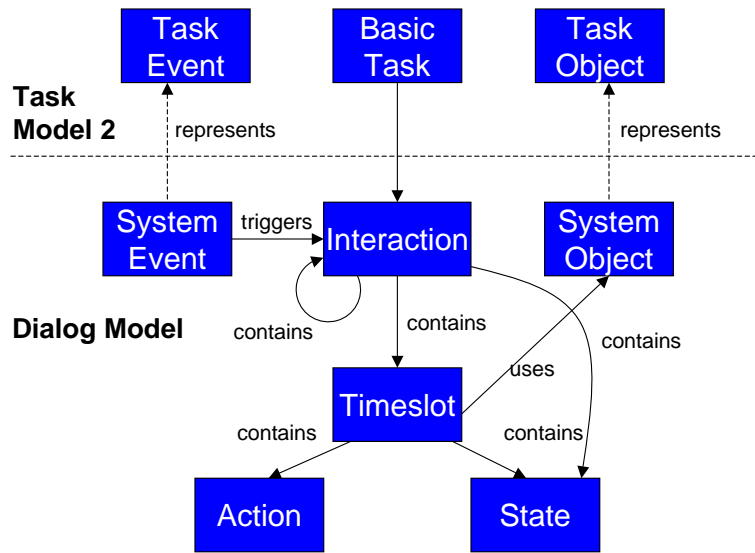


Figure 5.4: Connection between the task ontology and NUAN

5.7.1 Adding an interface pre-state column

The first extension is the addition of an optional new column, the pre-state column. It can be used to describe state conditions that need to be true before the described interaction can take place. The interface pre-state column makes it explicit what the state conditions are for the interaction to succeed. For example, when deleting a file it needs to be selected first. Selecting objects is an often-used action and is hence described as a single interaction, facilitating reuse. The deletion interaction can then be defined *without* reference to the selection mechanism. Such modeling helps the user to design more consistency throughout the interface, in the example concerning selection. The pre-state column is placed at the right-hand position in the first row of the NUAN table. The first left-hand position in the first row is used for annotation purposes, see Figure 5.5. In the original version of UAN there was no standard field for annotations explaining the interaction. The new comment field allows designers to explain in one sentence what the interaction describes and makes it hence more understandable for other designers.

5.7.2 A modified interface feedback column

Interaction takes place between the system and the user. In the original version of UAN the interface supposedly only provided *feedback*. Nowadays event driven systems often take the *initiative* in interaction. Therefore the name of the column was changed to "Interface Actions" in order to show that interaction takes place between two "equal" parties. Typical examples of systems where the user mainly reacts on signals include security systems, email system and workflow systems. Therefore, instead of always modeling interaction on the initiative of the user, in NUAN the system's action can be the first action to occur. Such an interface action is marked by an exclamation mark to indicate that the interface takes the initiative.

Interaction: New Email			
About		Interface Pre-state	
This interaction describes the arrival of a new email		emailer = running minimized	
User Actions	Interface Actions	Interface State	Connection to Computation
	! MESSAGE("New email has arrived")	mes_wnd = visible	unread_mail = true
	! ASK("Read now", [Yes,No])		
DECIDE([yes,no])			
POINTERTO(<yesbutton>) CLICK(<yesbutton>)	MOVEPOINTER(<yesbutton>) SHOW_EMAIL([latest])		unread_mail = false
POINTERTO(<nobutton>) CLICK(<nobutton>)	MOVEPOINTER(<nobutton>)		unread_mail = true
	HIDE_MESSAGE()	mes_wnd = hidden	

Figure 5.5: An example of a NUAN table

5.7.3 Expanding time capabilities

In order to make timing a bit more exact, in NUAN lines are needed to indicate that interactions occur after each other. Actions on the same line occur at the same time. In the original UAN, this was undefined and often actions on the same line were intended to occur *after* each other. This makes it impossible to specify actions that occur simultaneously. If actions cannot be specified in the space of the cell, it can be specified using multiple lines in *one* cell. As long as it is in one cell, it occurs at one moment in time.

5.7.4 Mental actions

In some cases it is very important to model the mental actions of the user. GOMS was one of the first techniques to do so. A limited set of mental operators has now been introduced, see section 5.5 for an overview of the mental operators. If mental actions are specified in the NUAN table there can be interface actions at the same time. For

example, an animation that is evaluated by a user. When used, the mental actions can give insight into the cognitive aspects that are needed. For example, it can be evaluated using NUAN whether the user needs to make a decision using elements visible on screen or from working memory. In figure 5.5 the mental action *decide([yes,no])* is used. In this case the user has to make a decision from a set of options that have already been presented by the system.

5.7.5 Generic interaction diagrams

Some interactions occur frequently and can be reused using generic interaction diagrams. In fact, most standard actions also use variables to make them more generic. In NUAN, *generic interactions* can be defined to reuse generic interactions in other tables. Reuse of generic actions is desired because it can improve the learnability aspects of the design. With reuse, the user only needs to learn the interaction once and can apply it in other contexts. Generic interactions can be recognized by their own label and the fact that they have arguments. In any generic interaction at least one user action needs to occur.

5.7.6 Parallellism

Parallellism is an issue in complex interactive systems where many users work together, synchronous or asynchronous. In (Sage & Johnson 1998), this kind of cooperation is modeled by "joining" two UAN tables into one table. However, from the users' point of view this is not relevant and all aspects can adequately be modeled using one table. Conditional events and the corresponding actions can be described using the interaction pre-state column together with the system events. Therefore, parallelism is *not* explicitly modeled in NUAN.

5.8 Evaluating Design Alternatives

The main goal in the detailed design phase is to create a design that maximizes the level of usability. Using the analysis results to develop initial design solutions, may give already some form of guarantee for an initial step in the good direction. However, evaluation is always needed to determine the usability of the system in the actual *context of use*. A good analysis does not automatically lead to usable designs.

Testing with users using software or paper prototypes are the traditional ways of evaluating usability. A disadvantage of testing with software prototypes is that it can only be done late in the design process when a lot of design choices have already been made. Testing with paper prototypes has the disadvantage that it still does not "feel" like a real system and data obtained in testing can only be used to evaluate general concepts. Ideally, designers should be able to evaluate their design solutions *early* in the design process when only high level and abstract specifications exist. However, if no users are involved, evaluation should be done carefully with regards to valid interpretations.

User centered design methods mainly use iterative prototyping as the main driving force. In itself, this is a very *poor* method and is characterized by the "trial and error" principle. No systematic reuse of design knowledge is done and the same mistakes could be made repeatedly. In real-life design projects, there are many factors that influence the design process and "threaten" attention for usability. Time and money are often constraints that prevent iteration in the design process, making the evaluation of usability during the *early* design stages even more important. When a project is running out of time, the last activities in the design process such as user testing are likely to be skipped. Moreover, maximizing usability may not always be the main design goal. Marketing strategies may dominate the constraints which may lead to less attention to the usability aspects. In each design project there will be conflicting requirements. Therefore, it is important that the design which is produced and that can not be evaluated as much as desired, is still as good as possible.

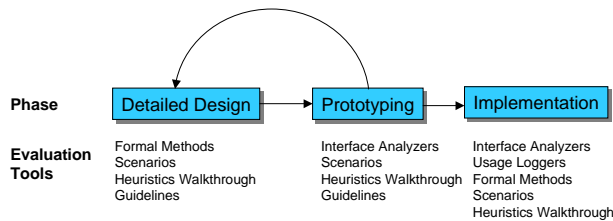


Figure 5.6: Evaluation tools in different design phases

Evaluating design alternatives can be done in several ways, depending on how detailed the design is. Figure 5.6 shows the possible evaluation tools for the basic phases in detailed design. For *early* evaluation only a few techniques can be used. In the next sections, we discuss these techniques for early evaluation of design alternatives.

5.8.1 Scenarios, guidelines, and patterns

Scenarios are often used for high level evaluation (Carroll 1995). A scenario is a description of an interaction sequence in a particular context. By interpreting them in a cognitive walkthrough (Polson et al. 1992), a heuristic evaluation (Nielsen 1993) session, or by acting them out, certain *claims* about the design can be tested. Hence, scenarios help designers to focus on specific aspects of the design. During such an evaluation design knowledge in the form of guidelines, heuristics, ergonomic criteria or design patterns, is used to determine potential usability problems. Walkthrough methods are typically applied by a group of designers and other stakeholders. This evaluation process is still subjective and the designer's expertise is needed to determine the correct application of guidelines, patterns and heuristics.

An advantage of these informal techniques is that "contextual" knowledge about the users and their tasks is usually incorporated. In (Carroll 1995) it is stated that scenarios should *always* include contextual information. Scenarios can be developed on the basis of a task model and the user interface specification is then evaluated in that context.

During the evaluation session the results can be explicitly compared to the task model. This also makes it easier to determine whether guidelines have been applied correctly. A well-chosen combination of these techniques can be effective in detecting usability problems, especially concerning the main task and goals of the users.

Guidelines or patterns can also be used apart from use in a walkthrough technique. They can already be used during the design activities as a source of design knowledge. Some attempts have been made to formalize design knowledge and to build tools that support designers in the use of guidelines, see EXPOSE (Gorny 1995) and DIADES-II (Dilli & Hoffmann 1994). The knowledge is usually a mix of dialog and presentation aspects with a strong focus on presentation aspects. Such tools can be very useful in assisting designers very early in the design process, during design and evaluation activities.

5.8.2 Prototype evaluation tools

Several tools and techniques exist for the (semi-)automatic evaluation of software prototypes. Most of them use a rule-based knowledge base that contains ergonomic rules, guidelines, or some other form of design knowledge. Such tools can theoretically lead to objective evaluations. For example, a technique such as ERGOVAL (Farenc et al. 1995) allows a semi-automatic evaluation of a prototype using ergonomic rules.

Tools like AIDE (Sears 1995) can evaluate a user interface based on an interface description file. AIDE uses metrics to make a quantitative analysis of the interface. The values of the metrics still need to be interpreted by the designer. An other approach is to extract the actual interface description out of an executable application. For instance by analyzing the Windows Resources of an application (which describe all the application widgets), constraints on layout or consistent usage of fonts, widget sizes, and colors can be checked (Mahajan & Shneiderman 1995). Such tools are a step in the right direction but many usability aspects simply cannot be evaluated automatically. For instance, aspects such as choosing good colors depends on culture and other aspects of the context. Determining the appropriateness of icons and labels is also difficult to do automatically. Although such tools require a minimal amount of time to apply, it can be argued that the gains of these tools are relatively small. Without considering the context of the application only a limited set of guidelines is still more or less valid. Since the focus is mainly on presentational aspects, many other usability aspects also remain unchecked.

Other tools such as EMA (Balbo 1994) and USINE (Lecerof & Paternò 1998) use a combination of a task model or interface model and actual usage logs. The actual usage logs are analyzed against a task model. The outcome of the analysis is an annotated user log that still needs to be interpreted by an expert. The properties that are found are mainly related to deviations of user actions compared to the prescriptive task model. Although such an evaluation is better because it includes more context, it is still highly subjective and the method may not provide direct clues on causes of usability problems nor on possible improvements. That has to be done by an expert designer. Additionally, a requirement for these approaches is a prototype where logging code has

been added, which may not always be feasible. An advantage of such an approach is that the models used are models that contain both dialog and presentational aspects as well as contextual information. However, such an increase in model richness may also complicate finding the causes of sub-optimal usability metric values.

5.8.3 Formal usability evaluation of the user interface

An other technique for early evaluation of user interfaces is by formal analysis of models that describe aspects of the user interface. Evaluating the user interface using formal specifications has some potential benefits. Claims can be defined very precise and they can be determined objectively and automatically. Theoretically a standardized set of claims could be evaluated for every user interface in order to find usability problems. If formal models are used for usability *evaluation* purposes, the challenge is to determine *meaningful properties* of the model that are *valid* indicators of usability. With the availability of such properties, tools could be constructed to develop design alternatives based on task models, hence the interest in model-based user interface design. The tools discussed in the previous section increasingly rely on such usability properties as well.

Formal specifications of the user interface usually only describe the user interface on the functional or dialog level. Contextual information is not used and claims regarding usefulness and appropriateness can therefore not be evaluated. In (de Haan et al. 1991) a number of formal techniques is evaluated including GOMS, (E)TAG, CCT, ETIT, and CLG. Other formal techniques such as Petri-nets, STD's and Z specifications can also be used and focus more on the functionality aspects. Other less formal techniques such as UAN and MCA tables also take functionality into account as well as the users' physical and mental actions. While these formal techniques may not be suitable for analysis of entire user interfaces they can be useful for analyzing certain small aspects.

Formal models are notoriously hard to build and their value has often been criticized, mainly because of their limited usefulness compared to the required effort (de Haan et al. 1991). For usability evaluation, there is a lack of well defined properties that relate to usability (van Welie et al. 1999b) and they are usually limited to dialog aspects and they ignore presentational aspects. For both general usability related metrics for user interface descriptions and usability properties in dialog models, it holds that their values are tentative and can only be used to *compare* design alternatives, which may still be useful.

Usability Properties in Formal Models

Formal evaluation of user interface models can only address usability on the level of means, as described in chapter 2. Several properties can be given for each means. The properties should be measurable, either relatively or absolutely. Usability properties are not generally agreed on and some have tried to develop such properties, also called metrics by some (Rauterberg 1995). Evaluation is then done by determining the values

of the properties using the user interface specification. However, as long as it is not known how the value of the property influences usability, no valid claims can be made.

Many techniques are based on counting the number of steps needed to perform a task or the number of rules the user needs to know. However, conclusions about such a number are tentative with respect to usability. In some cases, less steps is better but in other cases, when it comes to error-prevention, more steps is better. It depends on the kind of task at hand and type of function. The properties defined by Abowd (Abowd et al. 1995) are derivable from a state based dialog model and concern aspects such as state reachability. In our opinion, this is not directly related to usability but more to the correctness of the specification.

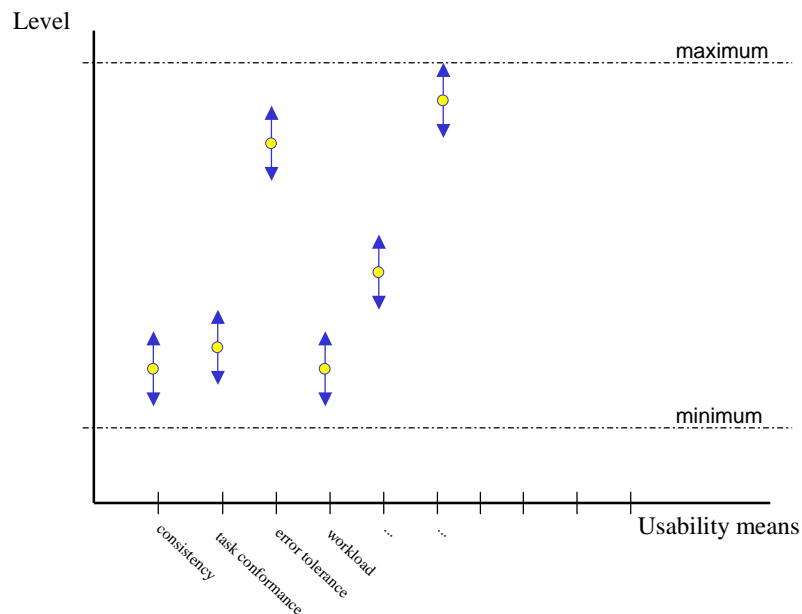


Figure 5.7: An example of optimal levels of usability means vs. their limits

Although the usability properties typically only have a meaning in the *context* of that design, it is possible to define properties that are only marginally dependant on the context and hence are valid for usability evaluation. Some properties may have a general meaning, e.g. the property, "Percentage of Undoable functions" should in any design be closer to 100% than to 50%. Unfortunately, such properties only help to define a *minimum* level of usability and help to detect just the most trivial usability problems. However, for finding the *optimal* level of usability the contextual information about the context of use needs to be incorporated. For each property, a minimum or maximum level may be defined on a value range but the optimal value lies somewhere in between. Consider an interface where a function is always accessible by pressing a function key or keyword only. This could result in a high level of flexibility but it is a highly undesirable situation if the user does not know this function exists at all or how to access it. This is the typical case for command line interfaces. Similar problems can arise if

interaction paths are short but because of the layout hard to notice by the user. These problems are mainly caused by the fact that such usability properties are *not orthogonal*. Optimal usability leads to a compromise over all properties. Figure 5.7 illustrates this phenomenon.

Using Contextual Models in Formal Evaluation

Formally verifying a dialog model alone takes a design out of its context. The design is verified without looking at the tasks users need to do, conventions and styles that are posing constraints, or specific aspects of the intended user group. If contextual models are included, properties could theoretically give a more valid indication of usability. The two most probable contextual models to use are a *user* model and a *task* model.

Checking a dialog against a user model comes down to looking at the dialog aspects where cognitive and motor skills and limitations are involved. Some aspects of user modeling are easier to deal with than other aspects. Considerable research into user modeling has been done to capture relevant aspects of user behavior when interacting with systems. PUMs (Young et al. 1989) is a technique that uses both a user model and a system model to evaluate usability. The user model is only a contextual model in the weakest sense since it does not include individual differences as found in the context of use. PUMs does not clearly define any general usability properties and the actual formal proof of properties remains difficult (Butterworth et al. 1998).

Verifying a dialog model against a task model is not straightforward neither. First of all, because of the diversity of task models it is not guaranteed that they model the same thing. An important issue in discussions about task models is the question what exactly they describe. Task models for model based systems and other methods like GOMS (Kieras 1996) and ConcurTaskTrees (Paternò et al. 1997) are usually *prescriptive*³ task models. Other task models such as TKS (Johnson et al. 1988) and GTA (van der Veer, Lenting & Bergevoet 1996) are mainly *descriptive* and focus on modeling the user's task knowledge. A consequence of this distinction is that the meaning of task and object is different. In a prescriptive task model the objects are mostly part of the system which is not necessarily true for a descriptive task model. Also, a prescriptive task model usually focuses on one user interacting with the system instead of taking into account other users and stakeholders, other roles and the environment in which a user may interact with a system. If a task model is included as a contextual model only the *descriptive* model should be used since it is then a valid reference to the current situation. If a *prescriptive* model is used, usability evaluation becomes an evaluation of a correct "implementation" of an "envisioned" situation.

It is clear that a task model's most obvious contribution is in checking task conformance, although this may not be easy to do. Currently the situation is such that neither task modeling nor dialog modeling is mature enough. Moreover, even if they were mature, finding valid usability properties that use the available contextual information is not trivial at all. It would require a thorough understanding of how usability "means"

³Although a notation in itself may not force the resulting task model to be prescriptive or descriptive, specific notations are often used in only one sense.

need to be based on the contextual information *and* how the different means together lead to optimal usability. Neither of those two relationships is well understood and forms one of the big challenges in HCI research.

5.9 Summary

This chapter discusses the detailed design phase. In this phase, analysis results are used to develop the actual user interface design. Making the transition can be done using the given guidelines. Once an initial design has been created, the incremental process of iterations starts.

The user interface consists of more than some buttons or menus on a window. We discuss which aspects need to be designed and which techniques can be used for that. The term User Virtual Machine is used to discuss the importance of the *functionality*, *dialog* and the *presentation*. Each of these is in some way important for the usability when the system is used in practice.

The user interface also needs to be specified. We discuss several informal and formal techniques. The informal techniques are easy to use, popular and usable but lack precision. The formal techniques are difficult to use but allow preciseness. These techniques should not be used in isolation and can complement each other when used together. In particular, we discussed NUAN. It is an extension we developed to improve UAN.

When the specifications are available, some form of evaluation can start. In order to use the specification techniques for early evaluation, they must allow valid usability indicators to be determined. For formal techniques we argued that such indicators can hardly be produced since no valid indicators depend *only* on the model of the user interface. Only if contextual models of the users and tasks are incorporated, more indicators can be determined.

Chapter 6

Interaction Patterns in User Interface Design

6.1 Introduction

Bridging the gap between analysis and detailed design is one of the difficult steps in design. Additionally, detailed design is also difficult when it comes to designing for usability. Besides the analysis results, design knowledge also plays an important role. This chapter discusses interaction patterns. Patterns are reusable and proven solutions that can be applied for design and evaluation activities. Patterns are focused on documenting and reusing proven design knowledge.

Guidelines have since long been used to capture design knowledge and to help designers in using that knowledge when designing user interfaces. Such design knowledge should help the designer to make the right design decisions and prevents the designer from making the same mistakes over and over again. However, applying guidelines is not without problems. Usually guidelines are numerous and it is difficult to select the guidelines that apply to a particular design problem. Guidelines are usually very compact but their validity or appropriateness always depends on a *context*. Software tools for working with guidelines (Vanderdonck 1999) can help but do not address the core problems of guidelines.

As an alternative to guidelines, we propose patterns as a solution to some of the problems of using guidelines. Patterns explicitly focus on context and tell the designer *when*, *how* and *why* the solution can be applied. The solutions are concrete so that they can be applied directly. Because of such a different focus, patterns can potentially be more powerful than guidelines as tools for designers. Inspired by the work of Alexander (Alexander et al. 1977), patterns have become popular in software construction (Gamma et al. 1995). Interest in patterns for user interface design (UID) goes back to 1994 (Bayle et al. 1998, Rijken 1994) but a proper set of such patterns still has not emerged. Some attempts have been made to create patterns but there appears to be a

lack of consensus about how patterns for UID should be written down, which focus they should have and how they should be structured. Consequently, a potentially even more interesting pattern language for UID has not been established since it is necessarily preceded by the development of a sufficiently large body of patterns. First, we take a closer look at why patterns can be more effective than guidelines. Then we look at the definition of patterns and how that translates to patterns for UID. We propose a template for UID patterns focused on usability and will discuss and illustrate the template with some examples.

6.2 Guidelines or Patterns?

Since HCI already has a long tradition developing design guidelines, it needs to be clear what the differences between patterns and guidelines are. The purpose of guidelines is to capture design knowledge into small rules, which can then be used when constructing new user interfaces. A pattern is supposed to capture *proven* design knowledge and is described in terms of a problem, context and solution. Since they have more or less the same purpose, the format may seem the only difference. On the one hand it is true that the design knowledge of a guideline could also be written down using a pattern template. On the other hand, the fact that a template is used to write down the guideline does not necessarily make it a pattern.

For patterns it is important that the solution is a *proven* solution to the stated problem and that designers agree upon the fact that it is a proven solution. Designers share values and ideas so the pattern must relate to their experience. In a sense, patterns represent pieces of good design that are *discovered/uncovered empirically* through a collective formulation and validation process, whereas guidelines usually are defined *normatively* by a closed group. Each pattern has a name and a collection of patterns forms a vocabulary for designers. Guidelines are mostly presented without explanations or rationale. In the Smith and Mosier guidelines (Smith & Mosier 1986) some guidelines have a short rationale in the comment field but they are often simply defined without any argumentation whereas some are just style definitions and not generic guidelines.

It has often been reported that guidelines have a number of problems when used in practice (Dix et al. 1998, Mahemoff & Johnston 1998a), e.g. guidelines are often too simplistic or too abstract, they can be difficult to interpret and select, they can be conflicting and often have authority issues concerning their validity. One of the reasons for these problems is that most guidelines suggest a general absolute validity but in fact, their applicability depends on a specific *context*. This context is crucial for knowing which guidelines to use, why and how they should be "translated" for the specific context. For many design decisions, it is simply required to know the whole *context of use* (ISO 1991c) including the tasks, the users, the environment etc. Without that knowledge, the design problem cannot be solved adequately. Guidelines have no intrinsic way of stating the context for which they apply and at most, it is briefly mentioned.

Another problem of guidelines is that it is often difficult to see what the *problem* is and why the guideline is like it is. For example, consider a very simple guideline saying, "Left align labels in dialog window" (Mullet & Sano 1995). What is the real problem being addressed by this guideline? It is not "how to layout labels" because that would be a problem of the UI designer. But what is the benefit for the user? In our opinion, the real problem should be concerned with understanding information on a display with aspects such as scanning time and readability which goes back to Fitts' law (Fitts 1954)¹. A pattern makes both the *context* and *problem* explicit and the *solution* is provided along with a *rationale*.

Compared to guidelines, patterns contain more complex design knowledge and often several guidelines are integrated in one pattern. Guidelines usually exist in a positive and a negative form; do or don't do this. Patterns however focus on "do this" only and are hence constructive. Furthermore, solutions need to be very concrete and should not raise new questions surrounding the core of the solution. Although they can point to other "problems" that often occur in the same context or in the context generated by this solution.

The last difference between patterns and guidelines is that patterns can be grouped and connected to form a pattern language. Each pattern can reference other patterns that solve sub-problems related to the same problem or to the solution. This way, design knowledge becomes semantically connected and designers can more easily access the knowledge concerning a particular problem. With guidelines this is not possible.

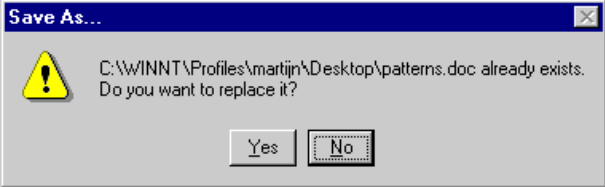
6.3 An Example

Table 6.1 shows a very simple pattern that illustrates what an interaction pattern can look like. Just as patterns in software engineering, the pattern has a specific structure that is adapted to interaction design. It is focused on the use of warning messages to protect the user against unintended actions. It is based on the idea to require two consecutive mistakes instead of one. Note that the pattern does not address *how* the protection layer should be implemented. The example merely shows a screenshot of an existing application where this pattern was applied. The pattern references another pattern, THE WARNING, that deals with how to design warning messages. Together these two patterns can solve the whole design problem but they each describe separate issues.

This example still resembles a guideline. However, other patterns may describe solutions that are less similar to guidelines. For example, the HELPING HANDS patterns describes the use of two handed input in entry tasks and the WIZARD pattern describes how users can be guided through a complex task. See appendix B for more patterns.

¹Fitts' law says, among other things, that the time to point at an object is proportional to the logarithm of the distance to the object.

Table 6.1: An Example of an Interaction Pattern

Name	The Shield
Problem	The user may accidentally select a function that has irreversible (side) effects.
Usability Principle	Error Management
Context	The user needs to be protected against unintended or accidental actions that have irreversible (side) effects. The (side) effects may lead to unsafe or highly undesired situations. For example the unintended deletion or overwriting of files. Do not use for actions that are reversible.
Forces	<ul style="list-style-type: none"> – The user needs to be protected but normal operation should not be changed. – The user is striving for speed while trying to avoid mistakes. – The severity of the (side) effects i.e. how undesirable is the effect if it was unintended?
Solution	<p>Protect the user by inserting a shield.</p> <p>Add an extra protection layer to the function to protect the user from making mistakes. The user is asked to confirm her intent with the default answer being the safe option.</p>
Usability Impact	Increased safety, less errors and higher satisfaction. However, it requires extra user action which leads to slower performance.
Rationale	The extra layer causes the user to require two repetitive mistakes instead of one. This way the user is also made aware of the safety aspects of her actions. The safe default decreases the chances for a second mistake.
Example	 <p>A copy of the file already exists at the specified location. Overwriting it will result in loss of the copy. The default is "No" so that the speedy user has to take the effort of saying "Yes".</p>
Known Uses	Microsoft Explorer, Apple Finder, Sony Vaio's Smart Capture
Related Patterns	THE WARNING

6.4 Patterns as Design Tools

Patterns are potentially better tools than guidelines because they are explicitly related to a context and are user problem centered. Although this may conceptually be true, in practice creating patterns for UID is not that easy. A pattern for UID is not necessarily structured in the same way as an architecture pattern and it is important to find a format and focus that has been designed for UID and has the right view on the important issues in UID. Suitability for describing usability related problems is an important issue for UID patterns. In this section, we will look at some definitions of patterns and we will propose a format for UID patterns.

6.4.1 Defining a pattern

As the name already suggests, a pattern is concerned with repeating elements, problems and solutions that emerge. Alexander (Alexander et al. 1977) defines a pattern as follows; "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution". He goes on explaining the nature of a pattern; "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over". From these explanations it shows that patterns are very practical, they describe instances of "good" design and not vague principles or strategies. Furthermore, they have been proven and are hence not theories or speculation. It is therefore necessary that a pattern contains a rationale why the solution works and proof by referring to examples where the pattern was successfully applied. Patterns are both *descriptive* and *prescriptive* and they help designers construct new instances. Alexander said a pattern should describe the *core* of a solution. Other related issues concerning the context are therefore dealt with by other related patterns that are being referenced. Patterns for different purposes usually do not have exactly the same template and for each purpose an adaptation is needed. The main fields are always *problem*, *context*, *solution*, and *forces*. The solution describes how the forces are balanced in the specific context. The remaining fields are extensions which should help make the knowledge even more clear.

6.4.2 Anti-patterns

Within the Software Engineering community, the success of patterns led to the development of anti-patterns (Brown et al. 1998). Anti-patterns focus on why things are not going right and then a solution is given. It can be seen as a pattern that is preceded by an example of bad design. It shows how not to do it and then how to solve it. Therefore, anti-patterns are *descriptive* and reflect on a particular design choice. In user interface design, many examples of bad design have been documented. Seeing bad designs may be very inspiring but it does not directly help to solve problems. Patterns and anti-patterns can also be combined by extending the normal pattern with an example of what is likely to happen if the pattern is not used. What is the danger of not solving the problem right? In particular for UID this may be very illustrative because the "danger" can often be shown with a single screen shot. For example, Figure 6.1 shows a screen shot that clearly asks for a pattern that addresses layout of dialog elements.

6.4.3 Types of patterns

Many types of patterns exist besides architectural patterns and software engineering patterns. Even within the field of HCI there are many possible uses for patterns. For example, organizational patterns describe reoccurring solutions in organizations and ethnographic patterns describe how humans act.

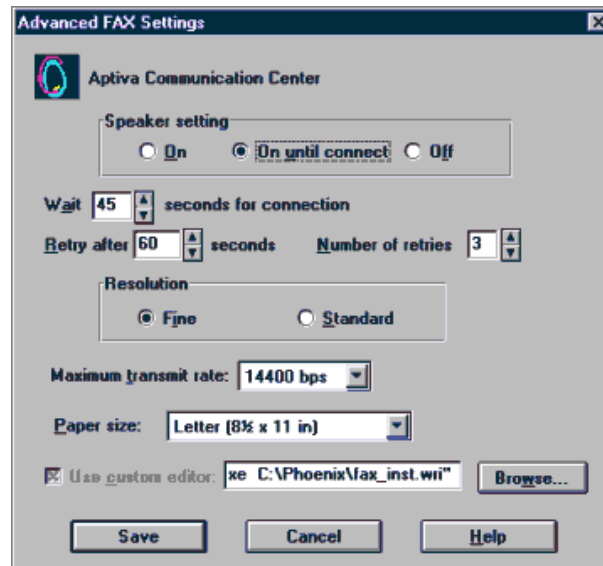


Figure 6.1: Chaotic layout of dialog elements

When doing a task analysis, task patterns (Paternò 1999) can be used to describe recurring task structures. Later on when doing detailed design, interaction patterns can be used to describe detailed design aspects of a user interface. Patterns help bridge the gap between an envisioned task model and the UVM specification in at least two important ways. First, task patterns such as described in (Paternò 1999) can be used to recognize common task structures. Using those patterns is can become easier to recognize tasks and even to see problems in the current task situation. When (re)designing, successful task patterns can be applied directly. The second important use of patterns, is to do detailed design. Task patterns will have a corresponding detailed design that describes the actual elements of the user interface. This solution will have a high level of usability for that task. In this chapter we focus on interaction patterns instead of task patterns. The interaction design patterns are also task related and form the most directly usable patterns in practice.

6.5 Interaction Design Patterns

Interaction patterns are different from the Gang of Four (GoF) patterns (Gamma et al. 1995) in that they describe proven solutions that solve problems *end-users* have, not technical construction problems. In this sense, a "design" is used to indicate the user interface from the viewpoint of the user. Therefore, patterns in user interface design are more related to Alexander's patterns (Alexander et al. 1977) and share the same intentions i.e. striving for quality for the users of the "things" we build. For interaction patterns, quality can be defined as *quality in use* or *usability* (Hartson 1998, Nielsen

1993, ISO 1991c). These patterns concern the static structure, the appearance and dynamic behavior of the user interface but not the implementation in terms of coding. They include the "look and feel" of the interface as far as it goes beyond mere style. The solutions described help users in the tasks they perform in day-to-day usage. In contrast, the GoF patterns describe solutions that solve construction problems of designers which are typically "invisible" to end-users.

The patterns presented in this chapter are part of an ongoing effort to describe successful solutions that benefit the users of the system. Consequently, they are solutions most of us are acquainted with. Other collections such as Common Ground (Tidwell 1998) or the Web patterns collection (Perzel & Kane 1999) do not make the explicit distinction between the user perspective and the designer perspective. In fact, they are primarily aimed at designers and make the user's problems secondary. Although some of those patterns indeed benefit users, they lack the proper focus and rationale. For example, the problem statement in Tidwell's patterns is typically of the form "*How can the artifact best show*" which does not explicitly relate to a usage problem. Generally speaking, each pattern that focuses on the user perspective is *also* usable for designers but not vice versa. Therefore, our patterns should benefit both designers and end users.

Interest in patterns for user interface design (UID) goes back to as early as 1994 (Rijken 1994) but although several pattern collections exist, an accepted set of such patterns i.e. a pattern language, has not yet emerged. There appears to be a lack of consensus about the format and focus of patterns for UID. Consequently, a pattern language for UID has not been established since it is necessarily preceded by the development of a sufficiently large body of patterns written with the same focus or "view". It is our opinion that patterns in UID require a special format which is focused on *usability*. Just as patterns in Software Engineering (SE) needed an adaptation of Alexander's format, patterns in UID will need their own format and view. Only with a clear view can we write and discuss patterns in UID which will ultimately lead to a pattern language.

6.5.1 Taking the user perspective

When taking the user perspective it becomes important to put emphasis on the argumentation for how and why the usability is improved. Without such a rationale it is impossible to see whether or why the solution is actually a good and accepted solution. Certain solutions in UID solve problems designers or marketing people have but not necessarily problems users have. E.g. banners and splash screens are accepted designs, but hardly for usability reasons. Others have focussed on HCI patterns that emphasize the problems that information providers face when designing web sites (Perzel & Kane 1999). In such patterns, marketing goals are the frame of reference that determine what is considered *quality* in proven solutions.

It is not difficult to identify patterns in user interfaces but it is hard to identify those patterns that really benefit the user, and explain the usability aspects. There are simply many examples of bad user interface designs, making it hard to identify good solutions. On the other hand, bad examples can be very useful to motivate designers to

use patterns, similar to the idea of anti-patterns. The Interface Hall of Shame² is a nicely commented collection of such bad examples. In other cases it is tempting to describe solutions that are minimizing usability related problems of users. For example, validating data after a user has entered it is not always the best solution, although frequently used; the user should not be allowed to enter syntactically incorrect data in the first place! The UNAMBIGUOUS FORMAT (see the appendix) pattern can be used to achieve this goal.

User problems are concerned with what the user wants to do with the system and how the interaction takes place. In the field of Human Computer Interaction (HCI) we speak of usability problems to indicate user/usage problems. It is recognized in HCI that usability is not an attribute of a system itself but it is only relevant in relationship to specific users who need to perform specified tasks in a specified environment. This stresses the importance of the context description of a pattern and allows for a natural mapping to patterns as originally intended by Alexander. The context section of a pattern should describe all characteristics of the users, tasks, and environment for which the pattern applies and serves as a selection criterion. The problem section then describes what fundamental usage problem the user is having for which the solution is intended. In other words, patterns in UID are typically task oriented.

6.5.2 Categorizing user problems

For patterns in user interface design several organizing principles/categories have been proposed (Mahemoff & Johnston 1998b). Mahemoff proposes the following categories: task related patterns, user related patterns, user interface element patterns and system based patterns. Our patterns are task/user related and in our collection we categorize patterns according to the kind of usage problems they address. In (Norman 1988) several user interface principles are suggested. The principles give an indication of the kind of problems and questions (McKay 1999) users may have when interacting with a system. Norman's principles are:

- **Visibility.** Gives the user the ability to figure out how to use something just by looking at it. *Hmm, I think this feature might do it*
- **Affordance.** Involves the perceived and actual properties of an object that suggest how the object is to be used. *Now how does this object work? Oh, I get it*
- **Natural mapping.** Creates a clear relationship between what the user wants to do and the mechanism for doing it. *To perform my task, I need to select this option, enter that information, and then press this button*
- **Constraints.** Reduces the number of ways to perform a task and the amount of knowledge necessary to perform a task, making it easier to figure out. *Oh no, what do I have to enter here? Ok, I just have these choices*

²<http://www.iarchitect.com/shame>

- **Conceptual models.** A good conceptual model is one in which the user's understanding of how something works corresponds to the way it actually works. This way the user can confidently predict the effects of his actions. *To perform the task, I provide the necessary information and give this command and it seems to work as I expected it to...*
- **Feedback.** Indicates to the user that a task is being done and that the task is being done correctly. *Great it worked!*

These principles above assume that users always exhibit fully rational behavior. In practice, users make mistakes and do things they do not really wanted to do. Therefore, additional principles are:

- **Safety.** The user needs to be protected against unintended actions or mistakes. *Oops! I made a mistake and here is how I correct it. Now I understand and I'll try again.*
- **Flexibility.** Users may change their mind and each user may do thing differently. *Now that I think about it, that parameter should have been ... Cancel it, I want to change the order.*

These categories of user problems and the questions they lead to, are quite general and the context description is important to distinguish the situations in which to use the pattern. Additionally, these problems can often be solved by several solutions which makes it even more important to be precise and concrete.

6.5.3 A focus on usability

If we focus on usability problems of users, we need to work out what the implications are for the way we write patterns. A pattern for UID should focus on solutions that improve the usability of the system in use, which must be measurable in usage indicators. Usability can be measured with the following usage indicators; learnability, memorability, speed of performance, error rate, satisfaction, and task completion, see Chapter 2. Each pattern should therefore state the impact on these usage indicators. In short, if a "UID pattern" does not improve at least one usage indicator, it is not a UID pattern. We believe that UID patterns always use a certain ergonomic principle and the rationale section should explain how the ergonomic principle as used in the solution leads to an improvement of the usage indicators. Most elements of the GoF patterns can be used directly for UID patterns as well. However, it is important to write them down from the right point of "view".

- **Problem.** Problems are related to usage of the system and are relevant to the user or any other stakeholder that is interested in usability. In contrast to SE patterns, problems in UID patterns should not be focused on construction problems designers are facing. Hence, problem descriptions should be user task oriented and fall into one of the categories as defined by Norman (Norman 1988).

- *Context.* The context is also focused on the user. What are the characteristics of the context of use (ISO 1991c), in terms of the tasks, users and environment for which the pattern can be applied?
- *Solution.* A solution must be described very concretely and must not impose new problems. However, only the core of the solution is described and other patterns might be needed to solve sub-problems. Other patterns relevant to the solution should be referenced.
- *Examples.* The example should show how the pattern has been used successfully in a system. An example can often be given using a screen shot and some additional text to explain the context of the particular solution. It is preferable to use examples from real-life systems so that the validity of the pattern is enforced. If a writer cannot find any real-life example, the pattern is not a pattern.

The fields and "view" needed to write UID patterns are important. For example if the view is taken wrongly, one might write patterns on "how to use tab controls". This is very tempting to do especially when guidelines are rewritten into pattern format. However, such views take on the perspective of the designer and not the user. Moreover, the design knowledge about "how to use tab controls" depends on the context of when it is applied, the users and their task. In other words, it is looking at the problem from the point of the solution without knowing the problem. Another addition we use in several patterns is a "Counterexample" section. This is an example of a real application when the pattern should have been applied but was not applied. It creates a kind of anti-pattern effect that serves as an extra motivation for use of the pattern.

6.5.4 A template for interaction patterns

As argued before, patterns in UID need their own format. We have chosen to use a template format in a table-style. It is also possible to combine all these elements in a more "story-like" format as Alexander did but since no studies have been done on which format is best suited for UI designers, we follow our own preference. For our current collection we have used the following structure and definitions:

Pattern Name

Each pattern should have a meaningful or catchy name that relates to the problem or the solution. The name allows one to use a single word or short phrase to refer to the pattern, the knowledge and structure it encompasses. Good pattern names form a vocabulary for discussing design problems and solutions.

Problem Description

Users can have all sorts of problems while using a system. The description states the user problem the pattern is trying to solve and objectives it wants to achieve within the

given context and constraints of the problem. User problems are typically task related and concern things users want to do, need to know, have to give or must control.

Usability Principle

Interaction patterns usually use a 'principle' on which the solutions are based. A complete set of principles is not known yet but an initial set can be given. The following list of usability principles is used grouped according to Norman's (Norman 1988) problem categories:

- *Visibility*: User guidance, Grouping, Incremental Revealing
- *Affordance*: Metaphors
- *Feedback*: Immediate Feedback, Legibility
- *Constraints*: Minimizing actions, Self-explanation
- *Conceptual Model*: Significance of codes, Compatibility, Adaptability
- *Safety*: Error Prevention, Error Correction, Forgiveness
- *Flexibility*: Explicit control

Context

A pattern solves a problem in a certain context. The context is the set of conditions under which the problem and its solution seem to recur, and for which the solution is desirable. Besides the problem description, the context also provides criteria for determining when this pattern should be applied.

Forces

This description represents relevant forces and constraints of the problem and how they interact/conflict with one another. Forces make clear the intricacies of a problem and define the kinds of tradeoffs that must be considered. A good pattern balances the forces in an optimal way.

Solution

The solution section describes the core of the solution. It may consist of diagrams, sketches and prose that identify the pattern's structure, dynamic behavior and their visualization, and shows how the problem is solved. A solution is formulated in an abstract way and does not specify specific instantiation details. The formulation of the solution starts with a one-liner that summarizes the solution.

Rationale

This section describes how the pattern actually achieves its goal, why it works, and why it is good. The solutions section describes the visible structure and behavior of the system, while the rationale provides insight into the deep structure and key mechanisms that lie under the surface of the system. The rationale should provide a reasonable argumentation for the specified impact on usability when the pattern is applied. The argumentation can be based on psychological, sociological, ergonomic, or organizational findings etc. that have been accepted in the disciplines involved.

Additionally, the impact the pattern has on usability when it is applied must be described. It describes what usability aspects have been improved, and which ones have become worse. Usability literature has identified the following measurable aspects of usability, see also Chapter 2:

- *Performance Speed.* How fast users can work with the system.
- *Learnability.* How easy the system is to learn.
- *Memorability.* How well users remember how to use the system.
- *Satisfaction.* The satisfaction users get when using the system.
- *Task Completion.* How much of the task could be completed.
- *Errors.* The number of errors users make.

It is usual that a pattern optimizes one or two aspects while other aspects have to suffer. Each solution tries to provide the right balance in the specified context.

Example

This section gives one or more examples of interfaces where the pattern has been used. Examples help readers to understand the pattern and enforce the fact that the pattern describes a proven solution. Examples should be taken from real applications and typically include screen shots or animations.

Counterexample

This optional section gives an example of a system where the pattern should have been used but was not applied. The counterexample serves as an extra argument in favor of the pattern.

Known uses

This section just mentions other applications where the pattern was used. If this pattern is really a re-occurring solution it should be possible to give some examples.

Related Patterns

This section references other patterns that address related problems.

6.6 Towards a Pattern Language

At the time of writing, our collection contains thirty patterns that have been formulated using the template given in the previous section. The reason for starting a new collection is that we wanted a collection of patterns that is strictly focused on problems of the end-user and not problems of designers. The patterns must be considered candidates since the process of reaching consensus is still in progress. Anyone can submit new patterns and the patterns are being discussed by a small group of researchers and practitioners. The site is built using XML in order to create a consistent and standard format for publishing patterns. Additionally, the use of XML facilitates several ways of automatic indexing or categorizing patterns. Pattern writers can submit a pattern using the pattern DTD³ while a style sheet causes all patterns to be rendered in a consistent way.

A representative set of patterns can be found in appendix B and the complete collection can be found online at <http://www.cs.vu.nl/~martijn/patterns/index.html>. The following list gives an indication of the what kind of patterns are in the collection. Only the name, problem and solution are given.

- **Wizard**

Problem: The user wants to achieve a single goal but several decisions need to be made before the goal can be achieved completely, which may not be known to the user.

Solution: Take the user through the entire task one step at the time. Let the user step through the tasks and show which steps exist and which have been completed.

- **Progress**

Problem: The user wants to know whether or not the operation is still being performed and has not stopped as well as how much more the user will need to wait.

Solution: Show that the application is working and give an indication of the progress.

- **Grid Layout**

Problem: The user needs to quickly understand information and take action depending on that information.

Solution: Arrange all objects in a grid using the minimal number of rows and columns.

- **Hinting**

Problem: The user needs to know how to select functions.

Solution: Give the user hints for other ways to access the same function.

- **Unambiguous Format**

Problem: The user needs to supply the application with data but may be unfamiliar with which data is required or what syntax to use.

Solution: Only allow the user to enter data in the correct syntax.

³Document Type Definition, a definition of the structure of the document

- **Preview**

Problem: The user is looking for an item in a small set and tries to find the item by browsing the set.

Solution: Allow the user to preview the item.

- **Automatic Mode Switching**

Problem: Users change their minds and may switch to new goals.

Solution: Detect the user's switch of goals and automatically set the new mode.

- **Helping Hands**

Problem: Users need to enter many different types of objects.

Solution: Use one hand to enter the data while the other hand is used to switch entry modes.

When a collection of patterns is agreed upon by a community, it is possible to speak of a pattern language. Patterns are usually related to each other and consequently a network of patterns constitutes a pattern language. The development of a pattern language is the highest goal in pattern research. However, before we can speak of a pattern language for user interface design it is necessary to develop good patterns. We have outlined a format for UID patterns and illustrated the format with an example. The next step is to develop a substantial amount of patterns and to start evaluation of the patterns to create the very important agreement. Validity and agreement are requirements for a pattern language. Not anything written down in a pattern form is a pattern and we should not accept them as such. Up till now, our work has focused on the development of patterns but in the near future the pattern approach needs to be validated to see whether they are indeed more effective than guidelines in their usage and in their effect on the end product.

6.6.1 Structuring a pattern collection

Our pattern collection presented in the appendix is a starting point for a pattern language. The patterns in the collection are linked and hence form a network of patterns, often visualized using a graph. Besides the fact that patterns reference other patterns, patterns can also be categorized. The GoF (Gamma et al. 1995) give a categorization in *Creational*, *Structural* and *Behavioral* patterns. However, for interaction patterns such a categorization is not appropriate and alternatives are needed. A categorization gives structure to a pattern collection and facilitates both *selection* and *understanding* of patterns.

Patterns are intended to be used as reference material when there is a need for them. The reason why a designer wants to search for a pattern determines the optimal structure of the pattern collection. For our collection, several possibilities exist. For example, the designer might want to address a particular problem or she might be looking for a pattern that closely matches a specified context. Other indices are the usability principle, the usage indicators that are involved, or patterns that only address presentational aspects. Table 6.2 shows an example of a structuring based on the type of end user problem.

However, not many interaction design patterns exist and it seems premature to make assumptions about how they will be used in practice. It seems likely that some categorizations will be used more often than others but in general there is no need to choose one fixed categorization. When XML is used it is possible to see a pattern collection as a database that can be queried in many ways. Therefore, we will not present a fixed structure for our pattern collection.

Table 6.2: Patterns and User Problems

User Problem Category	Patterns
Visibility	Command Area, Wizard, Contextual Menu
Affordance	Mode Cursor, Like in the real world, Setting Attributes
Natural Mapping	Like in the real world, Preview, List Browser, Continuous Filter, Navigating between spaces, Container Navigation
Constraints	Unambiguous Format, Focus!
Conceptual Models	Grid layout, Like in the real world
Feedback	Progress, Hinting, Contextual Menu
Safety	Warning, Shield
Flexibility	Preferences, Favourites

6.6.2 Developing a pattern language

The patterns in this collection have mainly been developed by examining often-used applications and by going through guidelines. Most of our efforts have concentrated on writing down the patterns with a consistent focus as described in the previous sections. The patterns presented here are a collaborative effort; patterns were distributed by email and discussed by a small group of researchers and practitioners.

In this process, we came across several problems which are perhaps specific for interaction patterns. First of all the formulation of the problem proved difficult because it is often hard to pinpoint what problem the user has and if the solution actually solves it. The problem field served as the first criterion to see if the spotted solution was solving usability problems. Additionally, the formulation of the solution was the part that was iterated most often. At first the solution tends to be formulated closely to the instance of the example found but it can usually be formulated more abstract.

After we had written around fifteen patterns we started to link the patterns in order to move on towards a pattern language. By doing this, we discovered some "holes" which led to the discovery of new patterns. For example, when a pattern is written only the essence is described which means that certain aspects are abstracted from or unaddressed. Such aspects can then be a topic for another pattern if it concerns a user problem. Although most of the patterns in the collection have been revised several times, the next development is to validate the patterns further in the community of user interface designers and end-users.

6.7 Summary

This chapter discusses patterns as an important tool in user interface design. Most directly, patterns can be seen as a better kind of guidelines. Patterns represent proven design knowledge in a much richer context than guidelines. Patterns are problem and context oriented which makes them potentially more usable for designers than guidelines.

The concept of a pattern is not new and has been popularized in architecture and software engineering. As in each domain where patterns are applied, patterns for user interface design require their own format and we defined a "mapping" specifically for user interface design. The format is based on the other formats as used in architecture and Software Engineering but applied with a focus on designing for usability.

Developing a collection of patterns is difficult. In order to validate the concept of interaction patterns, we developed a collection of two dozens of patterns. The patterns have been validated by a small group consisting of researchers and practitioners. Only when a body of patterns has been accepted by practitioners, we can work towards a real pattern language for UI designers. The defined format and focus for UID patterns should contribute to the development of such a body of patterns. However, much more effort is needed to develop and validate patterns in practice.

Chapter 7

Tools for Task-based Design

7.1 Introduction

Nearly all software development nowadays involves a graphical user interface (Myers 1995). Designing, implementing and improving the user interface is a complex and difficult process that greatly benefits from tools. Virtually all user interfaces are created using tools that significantly reduce the implementation effort. However, the early phases in software design that deal with analyzing the current situation and determining the functionality of the new product, are not so well supported as the implementation phase. In particular for task analysis only a few tools exist and those are not commonly used in practice.

Task analysis is often criticized as being time consuming and difficult to do. It is therefore an activity that could greatly benefit from tools that reduce the effort needed to do a task analysis. Currently, mainly tools such as sticky notes and blackboards are used but those are not usable for all task analysis activities. Tools could also improve the effectiveness of a task analysis if the quality of the models and produced documents is structurally guided by the use of a tool. A tool can help designers in modeling more accurately or performing a detailed analysis of the models. For the industry, it is simply a requirement that a modeling technique is supported by tools. If not, the technique will not be used at all.

This chapter describes the status of the current tools and set out requirements for future tools for task-based user interface design (TB-UID). Since tools have not had a big impact on task analysis yet, it is necessary to determine the requirements for successful TB-UID tools. One crucial aspect is that tools are used in a *process* and the role they have in the process is important to understand. Another important aspect is that TB-UID is a group activity which has implications on the way tools are being used.

7.2 User Interface Design Tools

Myers gives a survey (Myers 1995) of user interface tools and he defines categories such as tool kits, Interface builders, User Interface Management Systems, Integrated Development Environments, Design Exploration Tools, and Model-based Generation tools. Such tools are all intended for the implementation phase when a lot of design already has taken place. The tools allow prototypes to be built quickly and they facilitate evolutionary prototype improvements. Some tools are more high level such as the model-based development environments which start already earlier in the design process. Model-based interface development environments (MBIDE) often even use task and flow models and can be used earlier in the design process than other UID tools. However, MBIDEs are still focussed on reducing the implementation effort.

In this chapter we will focus on tools that support TB-UID activities such as *knowledge elicitation, creating story boards/scenarios, task modeling, and task model analysis*.

7.3 Supporting Task-based Design

Supporting task based design can mean many things. Some activities will benefit more from tool support than others. At the CHI '99 workshop on *Tool Support for Task-Based User Interface Design* both the current use of tools and the possible opportunities for tools were discussed (Bomsdorf & Szwillus 1999b). The group approached the issue by looking at the process of task analysis. The group organized the task analysis into five phases and for each phase the activities, the current tools and possible future tools were discussed, see Table 7.1.

Table 7.1: Tools and the Task Analysis Process

Phase	Getting started	Data Collection	Data Analysis	Evaluation	Envisioning
Activities	<ul style="list-style-type: none"> - Identify stake holders - Define goal statement - Define constraints 	<ul style="list-style-type: none"> - Collect data from stake holders - Additional research - Preliminary interpretations - Encoding (during collection) 	<ul style="list-style-type: none"> - Summarize raw data - Identify entities (tasks, roles, objects, events, agents) - Structure data - Scenarios - Elaborate entities - Clean up - Validate Accuracy 	<ul style="list-style-type: none"> - Evaluate effectiveness - Identify problem areas - Simulations 	<ul style="list-style-type: none"> - Optimize in response to goals - Improve - Establish design goals - Create and evaluate design alternatives - Innovations
Current Tools	<ul style="list-style-type: none"> - Flip charts - Word processor - E-mail 	<ul style="list-style-type: none"> - Audio/video recording - Post-it notes - Pen/paper forms - Bibliographies - Annotation tools - P.D.A. 	<ul style="list-style-type: none"> - Flip charts - Paper + post-it - Word processor 		<ul style="list-style-type: none"> - Paper/pencil - Flip charts - Word processors/outliner - Prototyping
Future Tools			<ul style="list-style-type: none"> - Multi-user modelling editor - Database support - Problem database - Consistency analysis 	<ul style="list-style-type: none"> - Scenario Simulator - Problem database 	<ul style="list-style-type: none"> - Story boarding tools

One of the first remarkable aspects was the fact that there was a considerable amount of agreement on the general activities that needed to be done during a task analysis. In some activities tool support was considered unnecessary and in other activities the "old

fashioned” tools such as white boards seemed very adequate. Especially during the *modeling* and *analysis* phases tool support was found to be lacking but much needed. Modeling tools were wanted and especially tools that support *cooperative* work since modeling is usually an activity done by several persons. Looking a bit more into the future, the group envisioned *simulators* that would allow designers to validate their models or in the future test their future task models.

7.3.1 Support throughout the process

In the ideal case, TB-UID should be supported throughout the whole process. Tool support could cover most activities from knowledge elicitation, task analysis, scenario development, sketching, interface modeling to prototype development. However, in practice this is not possible yet. In model based user interface design, the hope is that indeed the whole design process is covered, up to the generation of initial prototypes. By transforming one model into another, the process progresses from task model to implementation model. Such tools are called Model Based Interface Development Environments (MBIDE). MBIDEs exist for almost ten years now but they have not matured as much as one would hope. Most well known projects such as MASTERMIND (Neches et al. 1993), ADEPT (Johnson et al. 1983) have been stopped and MBIDEs never got beyond the level of research prototype. One of the fundamental problems is that the transformation of models is not trivial and can only be partially automated. Another problem is that the generated prototypes often do not have the desired level of usability. For certain kinds of application, such an approach has proven feasible but the approach cannot be generalized to all kinds of applications.

One of the latest MBIDE projects, called MOBI-D (Puerta 1997), is showing a more general approach that involves far more designer involvement. Instead of a high level of automation, the tool environment takes on the role of intelligent adviser and allows the designer to make the choices. The models are then used to create useful advice and design options instead of generating other models and finally prototypes. This kind of research is still very much in development and the availability of these tools remains problematic, which makes them hard to compare and test.

For the moment, it seems that tools are more effectively used for assisting designers in the design process as opposed to automating activities. When focussing on assisting designers, supporting all sorts of modeling and analysis activities are the main goals.

7.3.2 Integrated modeling and modeling Purposes

In TB-UID, several models are being developed throughout the process. Task models, user models, flow models, object models and scenarios play a prominent role. Most of these models serve a specific purpose and usually models are not explicitly related to each other. For example, scenarios and a task model are usually used as input for developing the detailed interface model but the link between them is not defined explicitly. Without such a link, models tend to be used in isolation which complicates the

development of tool support. This may be one of the causes why MBIDE's have not been successful yet.

Modeling in TB-UID is usually for one specific purpose and the integration with other models is not the main *intent*. Presently, the link between a task model and dialog model is a very problematic one but the link between a scenario and a user model is also undefined. Even if these would be defined, another issue is concerned with the *content* of each model. If a task model is used as the basis for a simulation, the models need to meet certain execution constraints. Such constraints may not always be desired for a more informal task model. In a similar way, it remains unclear what exactly a model should contain if it is used for the purpose of evaluation on a specific aspect, for example safety or performance.

It seems that the models that are used now for different purposes, are difficult to integrate. see Figure 7.1. Consequently, tool support is also fragmented and coupled to the model purposes. If integrated modeling is to be supported by tools, the models need to be integrated first while their "content" is specified in relation to a purpose, such as simulation. An ontological approach may help to reach that goal. By defining on a metalevel how the models are related, the models can be linked.

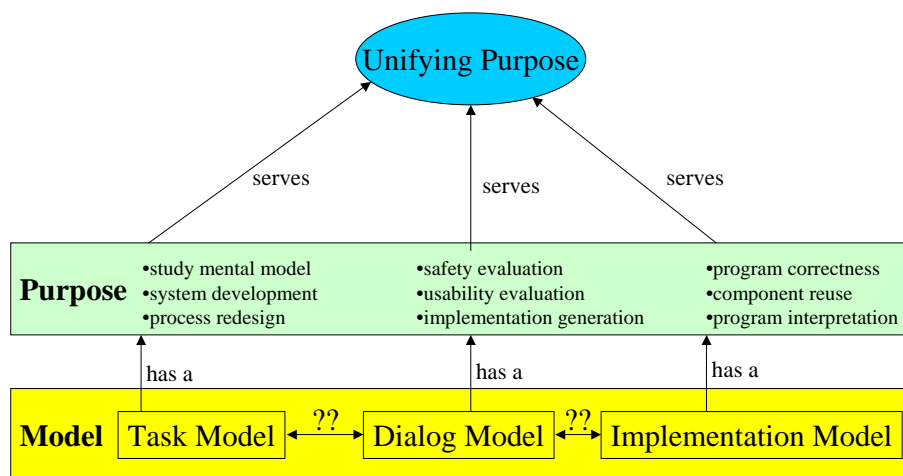


Figure 7.1: Model purposes

Although the ideal tool support for task-based design is not yet feasible, there are already interesting tools that can be used now. The next sections discuss the current state of the art of such tools.

7.4 An Overview of Current Task Analysis Tools

This section presents a brief survey of task analysis tools that have been described in task analysis literature or that are commercially available. Each tool will be briefly

outlined in order to get an idea what it can do. The purpose of this section is to illustrate the current state of the art in tools for TB-UID. Many tools have never been developed beyond the prototype stage but still they emphasize interesting aspects. We will use those ideas in the next section to outline requirements for future tools for TB-UID.

7.4.1 Commercially available tools

A limited number of commercially available tools exist that can be useful for the early phases in TB-UID.

- **Elicitation:** The Observer
- **Modeling:** WinCrew, ErgoWeb, Visio
- **Analysis and Simulation:** The Observer, WinCrew
- **Envisioning:** Storyboard Quick

WinCrew

WinCrew is a tool for constructing system performance models for existing or conceptual systems when a central issue is whether the humans and machine will be able to handle the workload. WinCrew can be used to predict operator workload for a crew given a design concept. What separates WinCrew from other workload models is this direct link between task-induced workload and the effect on system performance. WinCrew can predict how the human will dynamically alter his behavior when he or she encounters high workload situations. WinCrew can simulate the following as a function of high workload: dynamic allocation of tasks between humans and machines, dropping tasks based on task priority, task time and accuracy degradation. This tool originates from the US DoD and was intended for analyzing tank crew behavior. Since this tool is intended for a very specific purpose, it may not be generally useful in TB-UID.

The Observer

The Observer (Noldus Software 1999), see Figure 7.2, is a professional system for the collection, analysis, presentation and management of observational data. It can be used to record activities, postures, movements, positions, facial expressions, social interactions or any other aspect of human or animal behavior. Data can be entered directly into a PC or hand held computer, or code events from videotape. During an observation session, key-presses are used to log events and the time at which they occur. During observations, notes and comments can be added, which are stored together with the data. A complete system consists of several tightly integrated software and hardware components. It reads time codes directly from video tape or media file, which allows accurate event timing at the playback speed of your choice. Once data collection

has been completed, analysis options allow exploration of the data in time-event tables and plots, or generate reports with statistics on frequencies and duration, the sequential structure of the process or the co-occurrence of events. For additional calculations and inferential analysis (hypothesis testing), you can export the summary tables to spreadsheets, databases or statistics packages.



Figure 7.2: The Observer

ErgoWeb

ERGOWEB is a prototype software tool that offers support to the human factors specialist using hierarchical task analysis (Annett & Duncan 1967). The tool provides a direct manipulation diagram editor for HTA and provides high-quality output of analyses, in both graphic and automatically generated tabular form. The tool is implemented in Smalltalk-80 and currently runs on Macintosh computers. Apparently, this tool has

led to other tools because ERGOWEB is now a company that offers specialized software for ergonomics.

Storyboard Tools

In the film industry story boards are since long being used. Many tools exist for creating story boards such as ScriptWerx (Parnassus Software 1999) or StoryBoard Quick (PowerProductions Software 1999). They allow a designer to quickly create story boards using a multitude of standard elements and drawing tools. The story boards can then be animated to see the "basic" movie. Figure 7.3 shows an example of a storyboard tool.



Figure 7.3: Storyboard Quick screen shot

In TB-UID, such tools can be used to develop scenarios and initial sketches of prototypes. In the latter case, interface sketches are the main figures of the storyboard.

Structured Drawing packages

Most of the tools that are discussed in this chapter are special purpose tools. However, more general tools such as structured drawing packages can also be useful. Tools such as Visio contain an extendable set of building blocks with which the designer can create drawings. Typically, such tools already contain building blocks for business modeling and sometimes also for creating an office layout, see Figure 7.4. Such drawings can be used to model the physical environment in the task world but if new building blocks would be defined, object models or task models could also be made using such tools.

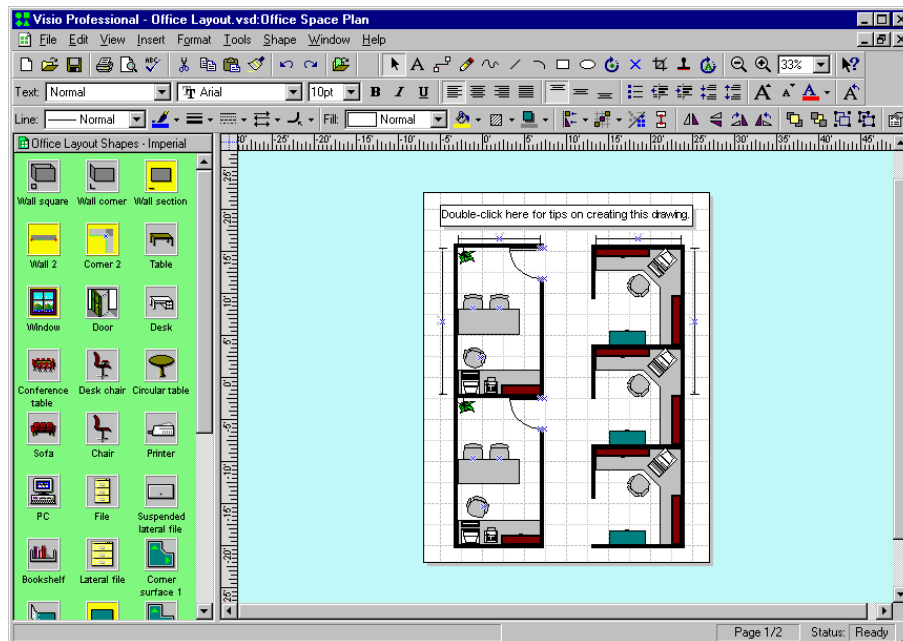


Figure 7.4: Visio

7.4.2 Research tools

In this section we discuss tools that originate from research institutes, typically universities. The tools are often the result of Ms.C. or Ph.D. projects but sometimes tools are also the product of more long term projects. The tools cover several phases:

- **Elicitation:** EL-TaskModels and U-TEL
- **Modeling:** CTTE, QGOMS, GOMSED, ALACIE
- **Analysis and Simulation:** CTTE, QGOMS, GOMSED, VTMB and TAMOSA
- **Envisioning:** Scenario Browser

CTTE tool and EL-TaskModels

Task Lotos Interactors Modeling (TLIM) (Paternò 1999) is a modeling method which includes ConcurTaskTrees (Paternò et al. 1997) for building task models. ConcurTaskTrees are hierarchical task models enhanced with LOTOS-based operators for specifying time constraints between tasks. The CTTE tool is a graphical editor for drawing ConcurTaskTrees written in Java and is available on the web, see Figure 7.5. Tasks have a certain type, which is visualized using different icons, and the LOTOS syntax is

used for specifying the time constraints. The tasks and the objects used in the tasks can be described further using templates. Besides editing functionality, the tool also offers consistency checking and the possibility to apply a transformation algorithm. At the time of writing, the tool was still under development.

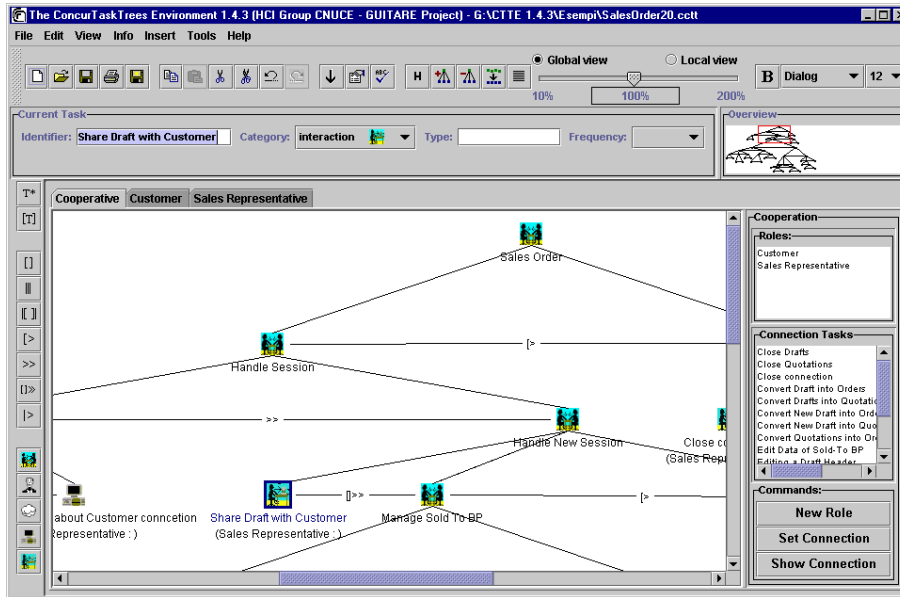


Figure 7.5: The ConcurTaskTree Environment

EL-TaskModels, see Figure 7.6 is a tool for elicitation of task models. A scenario is loaded into the tool and the analyst can use the tool to identify roles, objects and tasks. Identification of these entities is usually the first step towards a complete task model. The entities can be linked to each other and the scenario itself. The output of the tool is a ConcurTaskTrees description which can be elaborated with the CTTE tool.

U-TEL

U-TEL(Tam et al. 1998) is a tool that assists in elicitation of user task models from domain experts by natural language processing. It is part of the model-based user interface development environment MOBI-D (Puerta 1997) where it is the first tool in the design process. A piece of text, for instance a transcription of an interview can be analyzed in order to identify the tasks and objects that are being mentioned. The resulting task model will be refined by other tools later on in the process. However, the task models that can be specified are very simple and do not allow roles and responsibilities to be specified since they are designed for generating the user interface automatically.

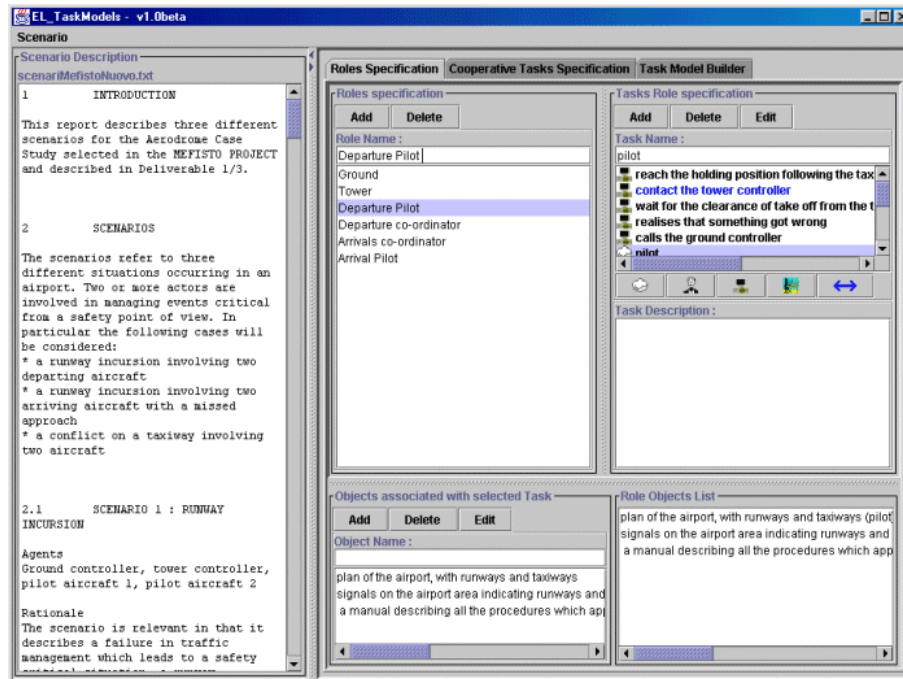


Figure 7.6: The EL TM tool

The Scenario Browser

The Scenario Browser (Rosson & Carroll 1995) is a tool to develop scenarios, see Figure 7.7 It is part of a larger environment intended to fill the gap between scenarios and the implementation objects, in this particular case in Smalltalk. A scenario is displayed in natural language and while other windows can be used to structure the information in the scenario. One window shows several "points of view" for a specific task, usually for different roles. Other windows can be used to specify the object structure relevant for the scenario and the *claims* that are used for analyzing the scenario.

ALACIE

ALACIE (Gamboa Rodriguez & Scapin 1997) – Atelier Logiciel d'Aide la Conception d'Interface Ergonomiques – is a workbench that includes two tools supporting IMAD* and ISSI as well the interaction between these models. IMAD is tool following the (Scapin & Pierret-Golbreich 1989) methodology developed at INRIA Rocquencourt France. It allows you to make MAD models and includes task templates and constructors. It is a graphical tree editor with support for verification of the descriptions. The tool checks every change in the description and automatically updates all logical and visual structures. Models created by the IMAD tool can be translated to interface ele-

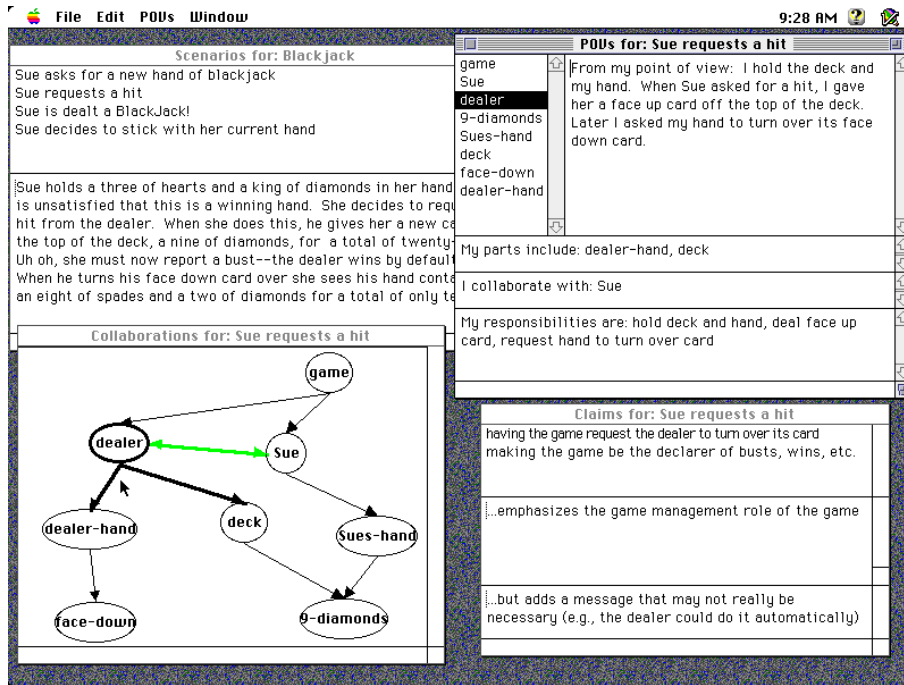


Figure 7.7: The Scenario Browser

ments by the ISSI as the next step towards a prototype. Except editing facilities, also some verification and validation of the structures can be done. The tool is however completely in French and is not freely available.

IDEAL

IDEAL (Hix & Hartson 1994) – Interface Development Environment and Analysis Lattice – is a tool environment that supports task analysis, qualitative and quantitative data collection, cost/importance and impact analysis. IDEAL consists of a number of separate tools and the output of each tool is made accessible to other tools via hyperlinks, the development lattice. IDEAL claims to incorporate hardware such as a videotape deck connected to a workstation. This is meant for videotape analysis during task analysis. One of the tools of IDEAL is QUANTUM – Quick User Action Notation Tool for User interface Management – which supports automated writing of UAN (Hix & Hartson 1998) task descriptions. Another tool is a Usability Specifications tool which allows the designer to specify usability attributes along with a Measuring instrument, Value to be measured, Current Level, Worst Acceptable level, Planned Target Level, Best Possible Level and Observed Results. For Data Collection and Data Analysis there are two tools as well. The data collection tool is based on the video setup and allows the designer to record data observable in the video along with comments. The

data analysis tool allows the designer to record problems along with their effect on user performance, importance, solution, cost, and resolution. Unfortunately work on the IDEAL project has stopped and the tools are not available to the public.

QGOMS editor

QGOMS (Beard et al. 1996) is a direct-manipulation tool for simple GOMS models developed at Idaho State University. QGOMS is a variation on GOMS and is referred to as "quick and dirty" GOMS. It reduces the number of modeling constructs and uses modified selection rule mechanisms. The tool visualizes tree models and allows direct manipulation of it, see Figure 7.8. During model construction, the task-completion times are automatically adjusted. Unlike GOMS the tool does not do simulations, only static task time calculations. The tool is a Visual Basic application and is available on the web. The tool was developed in 1996 and has not been developed further since.

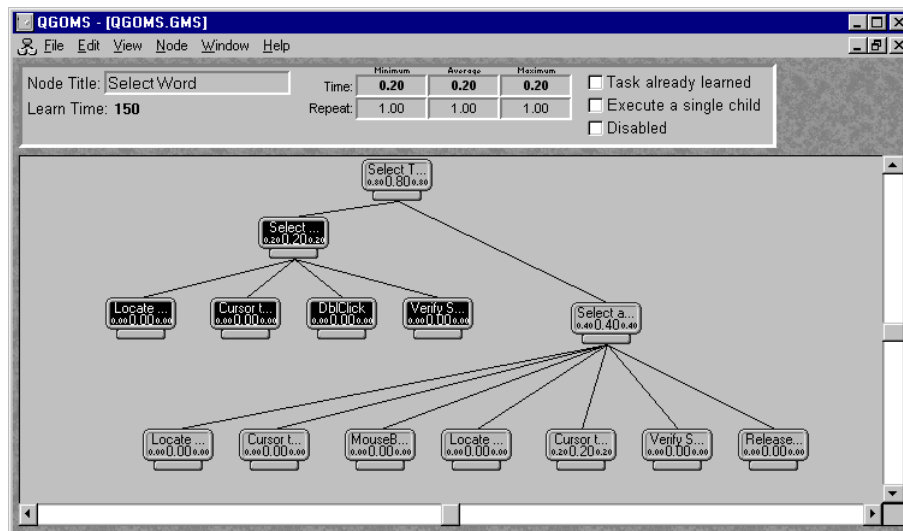


Figure 7.8: The QGOMS tool

GOMSED

GOMSED (Wandmacher 1997) is a tool to build GOMS models and automate the calculations. The tool has been developed at the Technical University of Darmstadt. The tool gives a tree representation of the GOMS models and allows the timing details for each method to be specified, see Figure 7.9. The tool can calculate the total time needed for the task. Similar to QGOMS, a simplified GOMS model is used. It allows only one operator and selection rules are not incorporated. The main focus of this tool is to decrease the time needed to make GOMS models.

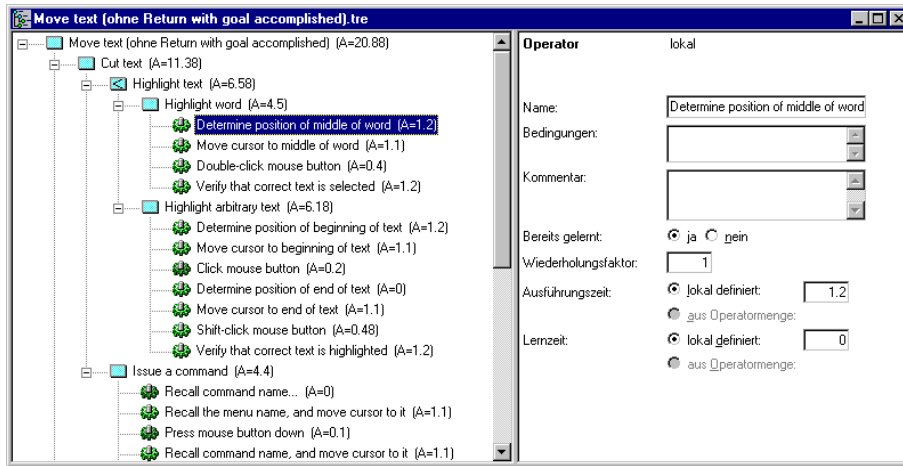


Figure 7.9: The GOMSED tool

TADEUS

TADEUS (Stary 1999) is a methodology and an environment for task-based design and prototyping of interactive software. The TADEUS approach provides a model-based framework for representation, a methodology, and a corresponding environment for user interface development. The inputs for this approach have been provided by techniques from work flow modeling as well as by user interface description languages, aiming at task-based and user-oriented development of interactive software.

VTMB and TAMOSA

VTMB (Biere et al. 1999) – Visual Task Model Builder – is a tool for creating task models and simulating task models. The task models contain temporal relations between tasks as well as conditions for task execution. Objects can be used in the task and can be created, used, modified, or deleted during the tasks. VTMB provides a graphical tree builder and allows the task model to be simulated for validation purposes, see Figure 7.10. The user can step through each task and gets feedback about which tasks are possible to execute. TAMOSA – Task Model Simulator and Analyser – is a second tool that explores simulation, see Figure 7.11. Both tools use the ODSN (Szwilius 1997) simulator for the simulation. Both are Ms.C projects and are not being developed further.

Discussion

From the short descriptions of the discussed tools, several things can be observed. First of all that commercial tools are very specific and that research tools are often quite general. The commercial tools are focussed on doing one specific thing, which

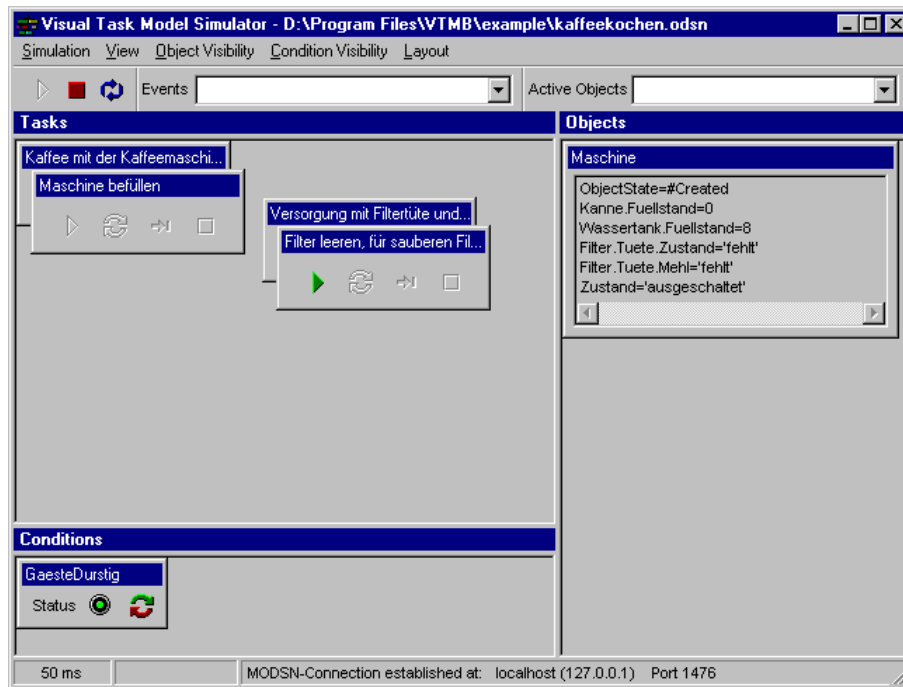


Figure 7.10: The VTMB tool

makes it possible to develop a product with well-defined functionality. Research tools often originate from Ms.C. projects or Ph.D. work and are usually part of theoretical work done. Consequently the research tools have the status of prototype and are not maintained after the project ends nor are they always available to the public. Table 7.2 shows all the discussed tools and the activities as listed in the workshop on *Tools for Task-based User Interface Design*.

Table 7.2: Tools in the TB-UID Processes

Data Collection	Model Specification	Analysis	Envisioning	Specifying Envisioned Model	Analyzing Envisioned Model	Early Prototyping
The Observer U-TEL EL-TaskModels	WinCrew CTTE ERGOWEB Visio	The Observer WinCrew CTTE	ScriptWerx Storyboard Scenario Browser	GOMSED QGOMS ALACIE CTTE Visio	GOMSED QGOMS ALACIE CTTE	VTMB CTTE

A research tool is either developed to support a modeling method or technique, or it is intended to support a specific activity. The TLIM editor, QGOMS editor and ALACIE are good examples of tools that support a modeling method. The U-TEL and IDEAL tools clearly support one or more specific activities. Also apparent is the lack of an integrated approach. IDEAL tried to provide such an approach but never succeeded.

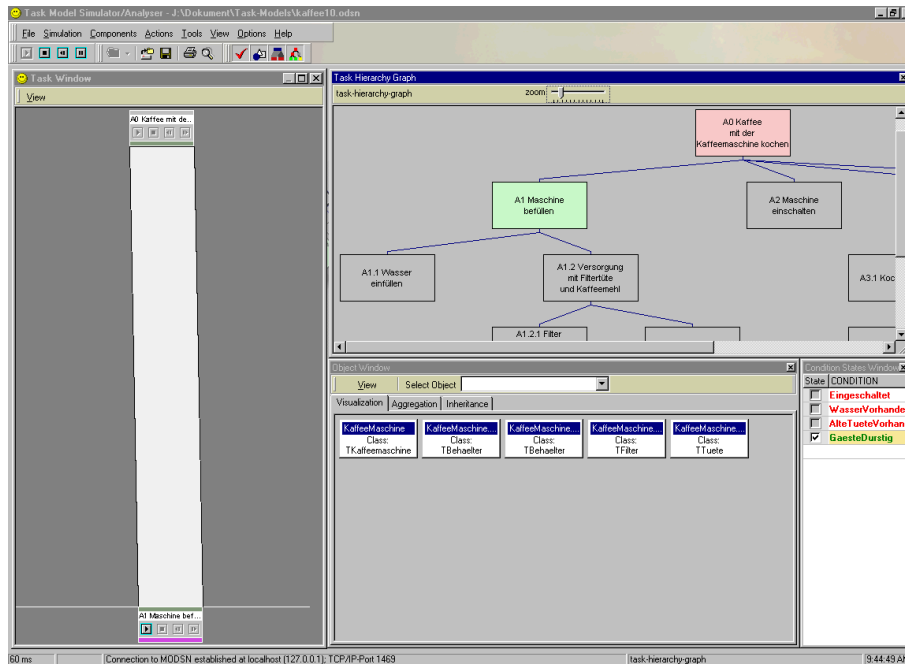


Figure 7.11: The TAMOSA tool

However, the early design process contains several activities that can benefit from tool support. Since tool availability is a serious problem, it is also difficult to combine several tools into an integrated set.

Practically speaking, the commercial tools can only be used for very specific activities and the research tools are hardly available and remain in prototype stage. In practice, paper and pencil or Post-It notes still seem to be the most useful tools now.

7.5 Requirements for Task Analysis Tools

Having seen that task analysis tools are still in their infancy, it is important to see how this situation can be improved. In this section a number of requirements are defined that may contribute to the development of better tools. Some requirements can be traced back to the discussion of the current tools in the previous section. Other requirements are related to requirements of CASE tools in general and apply to task analysis tools just as well. The requirements discussed here are not concerned with the main functionalities tools should have. These usually include creating, editing, and checking specifications. Tools that meet the requirements are hopefully more useful in real life projects and could improve the quality of the task analysis done.

7.5.1 Base the tool directly on a conceptual framework

Task analysis is more than just making task trees. Although a tree editor can already save you a considerable amount of time, a task analysis tool can be much more than that. Many more representations exist that focus on other important aspects. When a tool needs to offer multiple representations and provide analysis support, a conceptual framework needs to provide the basic concepts and relationships that are being studied, especially when the representations need to be consistent with each other. The conceptual framework will be the glue that should keep everything together and will be the basis for formal analysis. Before you can start building a tool, you need to have a clear conceptual framework. Adding such a conceptual structure to an existing tool is bound to fail.

7.5.2 Offer consistent and coherent representations

In task analysis there are already many representations that are being used and no doubt more are to come. A tool should be prepared for changes or new representations. When a tool offers more than one representation it becomes very important that the representations are all consistent with each other and coherent. Changes to a certain task specification should be automatically visible in all representations that include the task and definitions should be stored only once. In other words all the representations should be based on a repository of information from which the information needed for every representation is extracted. Basing a tool on a repository is also very commonly done in other CASE tools. Models could be used in more than one tool in the process and it is therefore important that the specifications can be interchanged.

7.5.3 Support cooperative task analysis

Task analysis is usually done by more than one person. One person may be scanning through documents while others are on location doing interviews while others are doing an ethnographic study. All the data that is obtained from different activities needs to be combined into one analysis. A tool is thus a groupware product and needs to be able to handle common issues in groupware applications such as version and change management, working online or offline, merging specifications etc. When a repository is being shared over a network, the repository needs to be accessible to all group members with the appropriate authorizations etc.

7.5.4 Support documentation including multimedia

After completing a task analysis the results have to be communicated to other members of the design team or to perhaps the client. If the knowledge the analysts have gained about the task world is not properly communicated, this knowledge easily gets lost and the task analysis loses its value. Task analysis results can be represented as task trees, detailed templates or process diagrams but multimedia fragments such as

images of objects, audio fragments of interviews or short video clips are also part of the results. Not every representation is always suitable for communicating results to others but a tool should be able to handle all these kinds of results and it should offer a coherent way of producing the documentation a one product. Especially in cooperative design teams more care needs to be taken that the documentation is easily available to all team members.

Integration with other documentation tools is also important. Most designers use word processing packages to write their documentation and they often need to put several of the representations in the documents. Therefore, tools need to be able to produce output in formats that can be used in the typical office suite applications.

7.5.5 Support design tracking

In large projects where the design group consists of several persons it becomes important to be able to track how the design is evolving. Decisions that have been taken between different versions of the specification, and important comments and remarks should be recorded so that a design rationale is being built up during the design. Some of the decisions can be documented by annotating the representations but more structural techniques such as QOC (Questions Options Criteria, (MacLean et al. 1991)) are also available.

7.5.6 Offer stability, robustness and product support

In the past many tools had the problem that they were a research prototype and therefore often implemented with a quick prototyping tool, such as Smalltalk. Some of those have become obsolete now and those tools cannot be used anymore. Because the tools were implemented with such a prototyping tool, they could not be easily distributed and used by others. In our opinion a tool for task analysis should be publicly available in an easy distributable way so that others are stimulated to use the tool or to design their own tool if they do not like it. In industrial settings the stability and robustness of a tool are important factors for choosing a tool. It has to be clear whether it is an unsupported tool that is not maintained or a tool that will be developed further and is well supported. For research prototypes it would be useful if the source code would be available after the project ended.

7.6 Discussion

The requirements show that task based design is a group activity which needs to be supported as such. Designers need to show each other their models, modify them or consolidate them, possibly even in a distributed environment. Such activities ask for a solid conceptual bases that integrates all the modeling aspects. Our task world ontology is a step in that direction.

Supporting the process also involves interfacing with other existing tools such as typically found in an office suite. No tool should ignore the other tools the designer works with and similar to other tools, tools for TB-UID should also be mature products themselves. This aspect is nowadays problematic at least.

Although it is perfectly understandable why the current tools are not up to par, the main problems are still on a conceptual level. The conceptual aspects of models, their integration, their representations and their usage are slowly beginning to become clear. In our opinion, the tools will only succeed when their conceptual problems have been solved.

7.7 Summary

In this chapter, the current status of tools for task based design were discussed. Tools nowadays form an essential element in user interface design. In task-based design, tools are often closely related to methods or techniques and can substantially reduce the effort in modeling and analysis tasks. While there has been some interest in tools that *automate* the design of user interfaces, there is now a tendency towards tools that *support* designers in specific tasks. Usable interfaces require the effective use of techniques but also human creativity for making the appropriate design decisions.

As shown in this chapter, tool support for task-based design is still in its infancy and requirements for future tools were proposed. There is clearly a need for tools that support modeling and analysis activities in terms of a group activity. A clear conceptual basis, usable representations and analysis techniques are the essential ingredients for more effective tools. In the next chapter, we discuss a new tool EUTERPE that has been designed with these requirements in mind.

Chapter 8

EUTERPE, a design workbench

8.1 Introduction

In the previous chapter, the state of the art with respect to tools for task-based user interface design (TB-UID) is discussed. It shows a clear need for tools that support task-based design in the broadest sense. This chapter discusses our tool EUTERPE. It is a research prototype intended to support task-based design. It is based on the task world ontology as discussed in chapter 3 and was designed to meet the requirements as discussed in chapter 7. In a way, the tool is an exploration into what kind of tool support is needed, since not many other tools existed at the start of the project. Task modeling and task analysis are in itself not clearly defined which made the development of tool support challenging.

EUTERPE has been developed over a period of three years and has been used for at least two years. Since the first version of EUTERPE, it has been publicly available and has been used by many people. EUTERPE was used mainly in education at several universities but it was also used in industry for actual projects. Feedback on the usage of the tool and new research ideas have been the primary forces in the development of the tool. During the project, it proved necessary to improve both the modeling concepts in task modeling and the way a tool fits in the design process.

The main focus has been on supporting task modeling and model analysis using a central repository of task and user data. Several representations allow the designers to structure the data, interpret and document the gained knowledge for other designers. Later on, support for other design activities has been added such as an editor for UAN tables.

This chapter starts with an overview of what EUTERPE is and what it does. Experiences of using the tool are reported together with indications for further development of the tool. The second half of the chapter discusses implementation issues concerning the architecture and the use of the task world ontology.

8.2 The Project's Context

The main goal of the project was to develop a workbench for task analysis. A prototype developed by a master's student had shown to be promising enough to develop a more elaborate tool. The new tool was intended to support multiple representations. Initially the focus was on editing task trees and templates but multimedia support was also a requirement. The tool was intended to deal with modeling itself but also with assisting the designers in detecting inconsistencies or other (un)desired properties of the created models.

An important choice that was made at the beginning was *not* to focus on automatically generating user interfaces based on task models. That would certainly have affected the models we were using and considering the fact that other researchers have not achieved convincing results (Puerta & Eisenstein 1999), we explicitly abandoned this path. Instead, we wanted to focus on modeling aspects, assisting designers where there was a need to model and analyze.

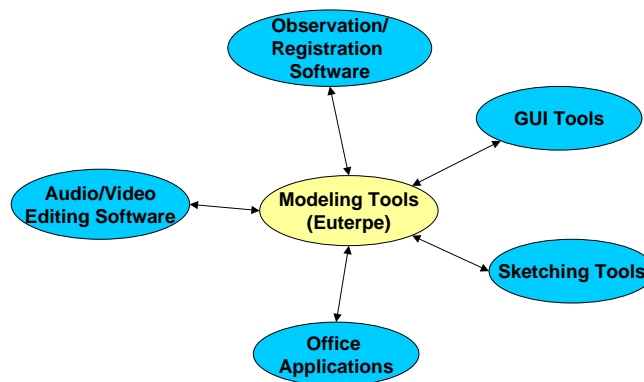


Figure 8.1: The Context of EUTERPE

EUTERPE was intended to offer support for several activities but it would certainly be part of a suite of tools for designers. Figure 8.1 shows the context of EUTERPE. Several tools for specific tasks are needed with which EUTERPE needs to communicate by being able to use the "products" or produce information needed in other tools. The main focus was on supporting the modeling activities and not the data gathering or design generation.

The development of the tool was done by iteratively developing and testing prototypes. Since we needed to define the "task analysis tasks" ourselves we used an exploratory design approach. Based on our experiences with doing TB-UID and after looking at other software tools, the basic development ideas were formed. In iterative cycles functionality has been added and modified. Although not all foreseen functionality has been implemented, the tool has shown to be usable enough for actual usage.

The implementation has mostly been done by the author. The NUAN editor has been done by a master student under supervision of the author. Other master students have

explored the workflow editor and the support for design space analysis. Unfortunately, not all explorations were mature enough to include in the public version of the tool. In most cases, the limited resources for doing programming constrained what features could be integrated. Nonetheless, we will discuss all our efforts in creating a workbench for task-based design.

8.3 An Ontology-based Tool

EUTERPE is an ontology-based tool. Internally, the task world ontology has been implemented very literally using a variation of Prolog. The ontology defines the structure of the data as it is stored in a repository. The representations are in fact views on the data in the repository, see Figure 8.2.

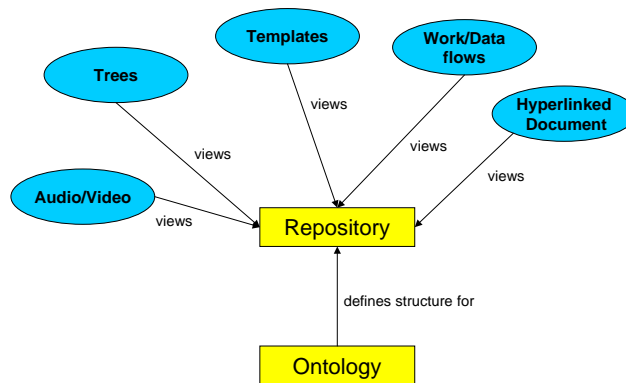


Figure 8.2: Conceptual Structure of Euterpe

The ontology only defines a structure for the task model data and does not define any particular representation. EUTERPE offers several representations and all these representations can be kept coherent because each representation is built up on the fly out of the same pool of information specified using the ontology. For instance, a task tree representation does not exist in the logical model but the structure is derived from the specified *subtask* relationships of tasks. By issuing queries to the Prolog engine, all the relationships can be inspected. Of course, EUTERPE allows most representations to be modified as well in which case the views need to assert the right facts in the Prolog engine. For instance when a new subtask is added by editing the task tree view, a new fact *subtask(X,Y)* is asserted. This way, designers who are the users of EUTERPE can work with the representations without having to deal with the logic representation underneath. In fact, the user of EUTERPE is never confronted with any of the logic underneath the system. The views that are offered are typically overlapping and hence partially dependent on each other. Each representation uses a subset of all concepts and relationships and consequently changes in one representation may affect others. By regeneration of the views the user is always shown the most up to date representation. In

chapter 4 several representations have been defined using the task world ontology and all of them can in principle be supported in EUTERPE.

8.4 Supporting Task-based Design

The main functionality of EUTERPE is intended to support task modeling. Representations include task trees, templates, and other hierarchical representations. Ideally, EUTERPE would be a workbench that supports designers during task-based design. For each activity that could benefit from tool support, a component would be present. Support for task modeling, dialog modeling and documentation is present but many other components could be added e.g. support for simulation, design rationale and sketch based prototyping. In this section, we discuss the functionality that has been implemented in the latest version of EUTERPE.

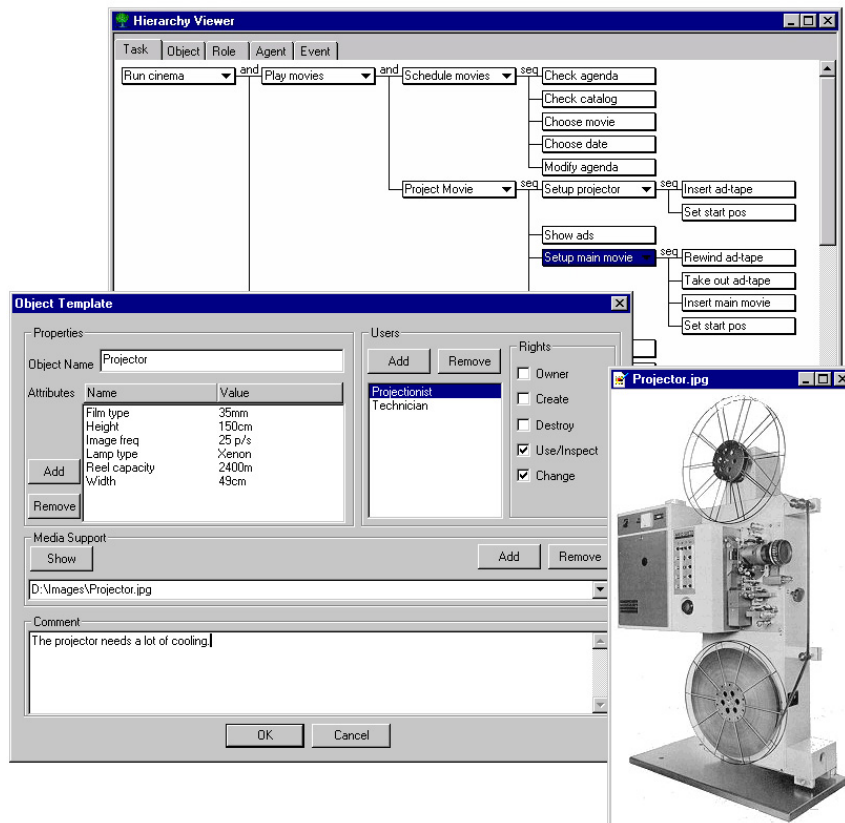


Figure 8.3: The Hierarchy Viewer, an Object Template and a Picture.

8.4.1 Supporting task modeling

The most complete support is for task modeling. When the tool is started, the designer can choose to create a new task model after which a new *HierarchyViewer* is shown. The *HierarchyViewer* is a window that contains tab sheets for each of the ontology's concepts, except for the goal concept. The *goal* concept was added later to the ontology and hierarchical representations of goals have not yet been included in the tool.

The first sheet shows a task tree and the designer can build a task tree by inserting child nodes or sibling nodes. For objects there are two hierarchies, one for the containment hierarchy and one for the type hierarchy. Events and Agents are not hierarchically structured and are therefore shown as lists. Besides the *HierarchyViewer*, some work has also been done on the development of a workflow viewer. The version shown in Figure 8.4 is still in prototype stadium and has not yet been integrated in the public version of EUTERPE. The prototype is intended to support work/dataflow diagrams as outlined in chapter 4.

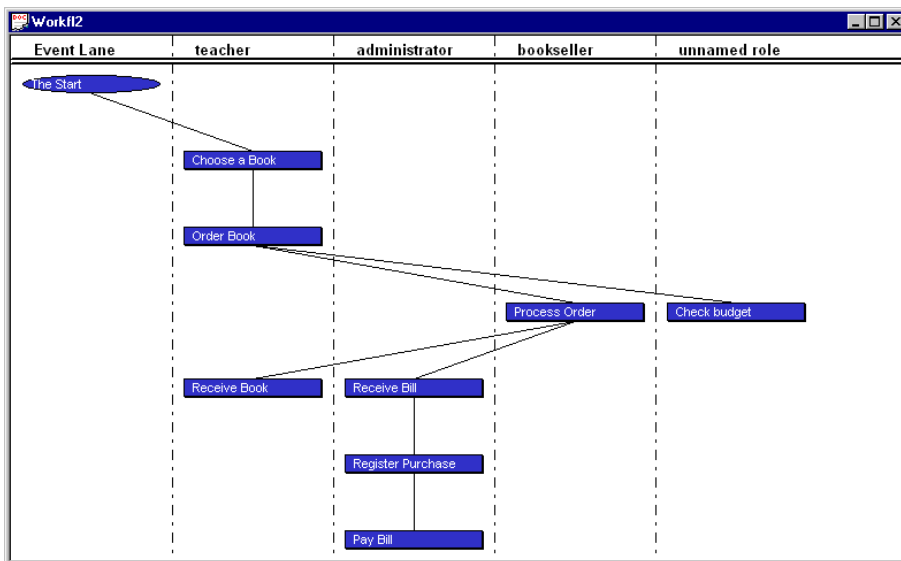


Figure 8.4: Prototype Workflow Viewer

For each concept a template exists that appears when double-clicking the node. The task template allows the designer to specify task details such as timing information and task conditions. Additionally, some relationships with other concepts can be established and viewed. The task template has become rather full of fields and after user testing it was decided to initially show only the most used fields and present the other fields on request. For templates of other concepts this was not necessary because of the small number of fields. Figure 8.3 shows the hierarchy viewer, an object template and an image.

Task trees tend to become quite large, a hundred or so tasks is not uncommon. There-

fore, trees can be (partially) collapsed to give the designer more overview. Additionally, it is possible to zoom in/out so that the visible part of the task model can be optimized. This feature proved very useful during modeling but also during presentations where low resolution displays were used.

Editing the models has been implemented conform the Windows style guidelines and includes cut and paste functionality as well as drag and drop functionality. Using the editing functionality the designer can move nodes, copy sub trees, delete nodes etc. This is actually the most used functionality of the tool. All functionality is also accessible through the keyboard. All user actions can be undone using the multilevel undo function.

8.4.2 Supporting model analysis

Another activity that is supported by EUTERPE is task model evaluation. As soon as some form of task model exists, the model can be evaluated. Evaluation can be done for several purposes as discussed in Chapter 4. An important question is how model evaluation could be supported best. All of the evaluation properties defined can be done using the internal Prolog system but we decided to take a designer's perspective first and see how evaluation could be used in practice. Instead of starting at the Prolog level implementation, the user interface aspects of doing evaluation were considered first.

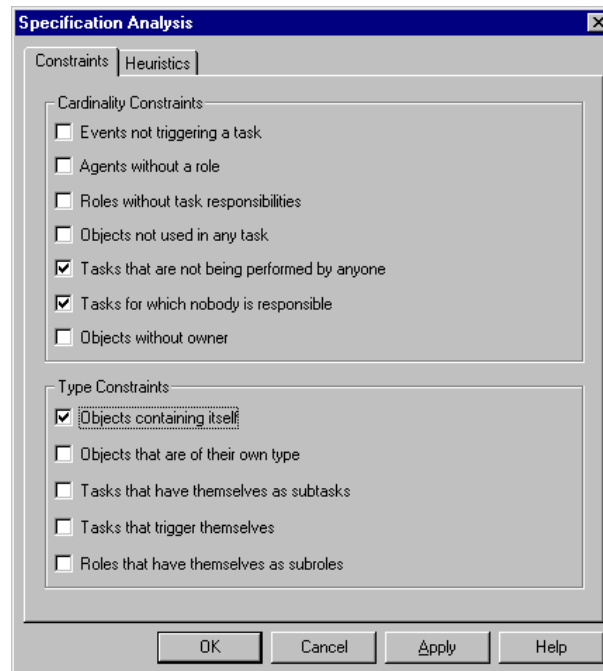


Figure 8.5: Evaluation constraints

One of the things we noted when design students held presentations about their task models, was that they tended to mark certain tasks. Sometimes because these were problematic or incomplete and sometimes because these needed to be optimized in the future. This led to the idea of coloring nodes in a tree as a way to *visualize* the results of an analysis. If coloring was the visualization, we still needed a way to specify *what* needs to be shown. In section 4.4, we discussed some of the questions designers might have. Some of them were questions that surfaced when observing designers in case studies and others were added to complete the possible set of questions. These questions were directly used in the tool to specify what needs to be shown. Figures 8.5 and 8.6 shows the dialog screen for specifying the questions the designer wants answered. The questions often need some parameters to be specified which is done using a form filling dialog style. Internally, the question is automatically translated to a Prolog query and the user never has to build any complex queries. The constraints have been split into *constraints* and *heuristics*. Constraints apply to every specification and should ideally have zero results, see Figure 8.5. Heuristics can be used to analyze a specification, to find inconsistencies or problems, see Figure 8.6.

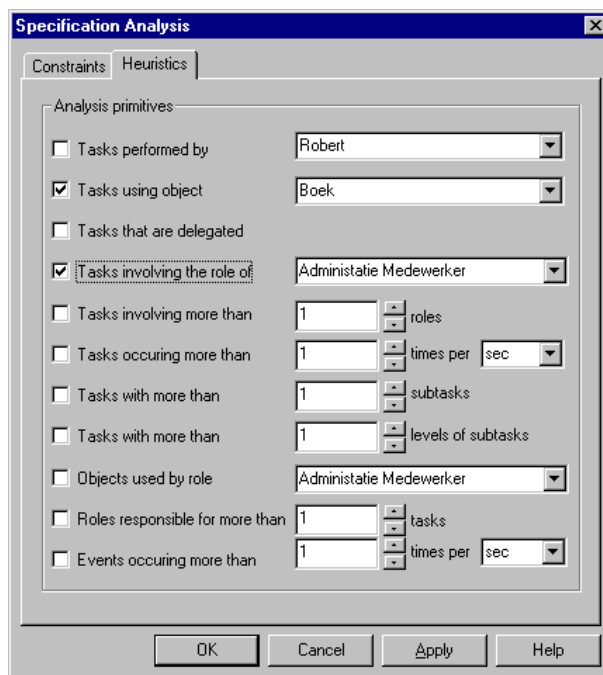


Figure 8.6: Evaluation using Heuristics

EUTERPE can process several queries in parallel. Each query is evaluated using the internal node database, and the nodes found by the query are colored, see Figure 8.7. If a node is selected in more than one query, it gets the color belonging to the last query. While this is not a satisfactory solution, in practice this was not considered a problem and therefore not addressed further. The ability to analyze a specification was regarded

useful by the users. One possible extension that came up when it was used was to see multiple instantiations of the same query, for example, all tasks by X and all tasks by Y.

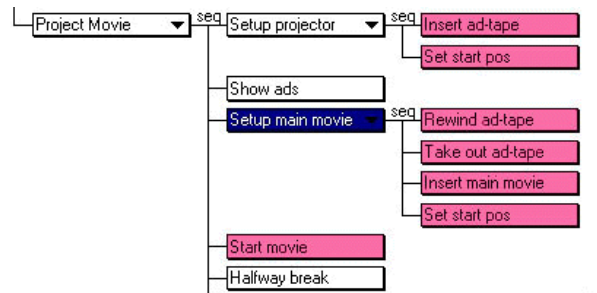


Figure 8.7: Coloring for evaluation

8.4.3 Supporting cooperative design

Usually design is a team effort and even a task analysis is mostly the combined job of several persons. During a task analysis some analysts may be going through documents while others are processing video tapes and still others are modeling. Sometimes task analysis requires “on the spot” observation and analysis may be done at several different locations. Such conditions demand a tool with multi-user support where users can cooperatively work on models from different geographical locations at the same time. Without multi-user support the analysts have to meet in order to update the specification with their findings. Multi-user support for EUTERPE would mean that the specification (the logical model) is edited by several persons from different locations. Designing such a tool needs to deal with the same problems that many groupware applications try to solve, i.e. keeping track of changes and merging several specifications. We explored a client-server version of EUTERPE in order to create multi-user support. The Prolog engine was decoupled from the application itself and ran as a server application. Although the server worked, we never completed the work for the “client”.

To support cooperative design without multi-user support, EUTERPE allows merging of documents by copying model elements. Merging documents is often necessary when several persons work on a model. This can be done using the clipboard. EUTERPE can open multiple documents at the same time and by copying nodes or trees via the clipboard, specifications can be merged and restructured. This way partial models can at least be reused and do not need to be entered again manually.

8.4.4 Supporting documentation

EUTERPE has two ways of producing documentation. First of all, by using the printing functionality. Task trees and object hierarchies as well as lists of events and agents can

be printed. If a tree does not fit on one sheet of paper printing is automatically done tiled on multiple pages.

The second way of producing documentation is by exporting the specification to HTML. EUTERPE can generate a set of HTML pages including an applet containing a task tree (or object/agent/role/event tree) that allows browsing of the task analysis results, see Figure 8.8. As a result these pages are a "read-only" view of the model since changes are not propagated back to the Prolog engine. All the pages together can be seen as a hyperlinked task analysis document because for each concept that is referenced a hyperlink is added. For instance, a reference to an object used in a task becomes a link to the description of that object and vice versa. Links to images and video fragments are also generated. The applet shows the trees graphically and when a node is selected the browser jumps to the corresponding entity. When large design teams actually used EUTERPE, the produced HTML documents were put on a web-server and these constituted the main reference document for the other members of the design team.

The screenshot shows a Netscape browser window titled "hotel Templates - Netscape". The address bar shows the URL "http://www.cs.vu.nl/~s3aan/Euterpe/". The browser displays a task model for a hotel. On the left, there is a list of tasks under the heading "Tasks". The main content area shows a detailed view of the "Task : Make Reservation" task. Below this, there is a "GTA Tree Navigator" applet showing a hierarchical tree of tasks and sub-tasks.

Name	Make Reservation	Initial state	
Goal		Final state	
Task type	user_action	Start condition	
Constructor	or	Stop Condition	
Sub tasks	Call Local Agent Use Web page	Used objects	
Performed by		Duration	
Involved roles	Booking agent Room Booker	Frequency	
Triggers			
Triggered by			
Comments			

Figure 8.8: A task model in HTML with Java applet

Another important aspect is the integration with common Office applications. Design-

ers typically write reports in which they need to include some of the design representations such as task trees or parts of UVM specifications. A tool must therefore be able to produce output in formats that can be used in typical office applications. In EUTERPE, we use Windows Metafiles as the exchange format for representations. Such representations can be arbitrarily scaled or modified in office applications without loss of image quality.

As mentioned in the requirements, the outcome of task analysis could be much more than text documents with some graphical representations such as trees. In addition to the hyperlinked documentation, EUTERPE also offers support for multimedia elements such as images, sounds and videos. Each concept instantiation can have a list of media objects connected to it. Scanned documents or video fragments can be added to clarify certain tasks, objects or any other concept. We found that especially video fragments turned out to be very useful because they are very effective in showing other members of the design team, who were not involved in the task analysis, what the task world looks like.

In the projects we did, video recordings of ethnographic studies were scanned and broken up in short video clips that show some particular "hot spots". Such video clips are typically between 1 and 3 minutes long and converted to MPEG 1 format, which makes them 10 to 30 Mb in size. Typically 10 to 20 of such video fragments were added to a specification which made it feasible to use video in terms of size and added value. Any computer that is sold nowadays can play MPEG 1 files without hardware support and a CD-ROM can contain up to 1 hour of MPEG 1 video. When HTML pages are created the media files are optionally included so that they can be viewed online as well.

8.4.5 Supporting dialog modeling

The first step towards support for dialog modeling in EUTERPE is the NUAN editor. NUAN is a variation on UAN that supports generic templates, mental actions and event driven interfaces. See Chapter 5 for an in depth discussion of NUAN. The task model is intended to model until the level of basic tasks. Delegation of the basic tasks to the system can then be described more precisely in NUAN. Therefore an action in NUAN is linked with the task model at the level of a basic task. Figure 8.9 shows an example of the NUAN viewer. In theory, the NUAN editor is also ontology based as is discussed in (van der Veer & van Welie 2000). We also defined the connection with the task world ontology but this link has not yet been implemented using the underlying Prolog system of EUTERPE. Hence, it is not possible to evaluate constraints that combine task model aspects and dialog model aspects.

The NUAN viewer allows NUAN tables to be edited and tables to be structured into a hierarchy. The hierarchy is shown on the left side while the right side shows the selected table. Tables can be copied and moved in the hierarchy. The NUAN table viewer uses syntax highlighting for standard actions and mental actions. Tables can also use variables and can hence be used as generic templates for actions. Like the task model views, the tables can be exported as Windows Metafiles for use in other

office applications. For other documentation purposes, the tables can also be exported to HTML.

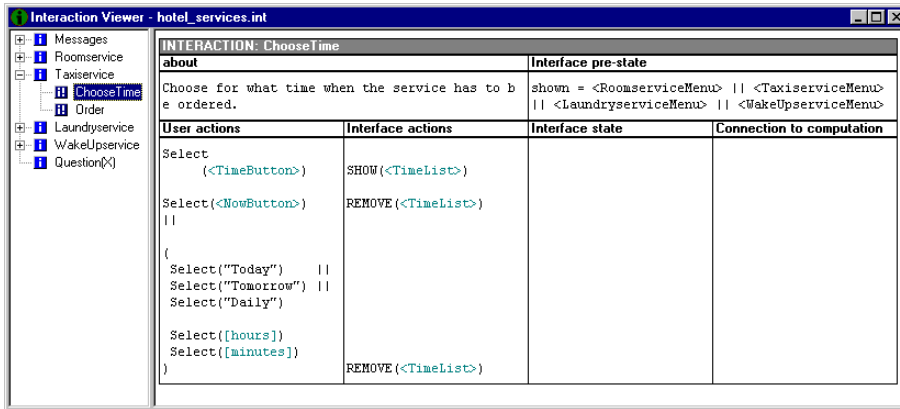


Figure 8.9: Nuan Diagram

8.5 Extending EUTERPE

Euterpe was designed to support multiple views and only some have been implemented. Extending can be done by adding new representations or adding new design task support. In the architecture of EUTERPE, the flexibility is largely related to the use of the task world ontology. An arbitrary number of views can be added provided that they are implemented as additional views on the ontology. One of the views that has been added in addition to the Hierarchy Viewer is the Workflow viewer. In the same way, other views such as a scenario viewer could be added. Other functionality extensions could be independent of the repository and could hence be seen as a separate "module" within the tool. One example of such an extension would be to include a task simulator. Such a simulator could read a specification and then execute it. Another example is to include support for design rationale. Both of these extensions are discussed in the next sections.

8.5.1 Adding support for design space analysis

Design Space Analysis (DSA) is used in TB-UID to systematically deal with design options with criteria to evaluate them. When important design decisions are made a technique such as Questions Options Criteria (QOC) (MacLean et al. 1991) can be used to document the design rationale. In this area, some work has been done on supporting QOC diagrams. Such diagrams aid in choosing options by systematically confronting all possibly relevant options with all relevant criteria. Figure 8.10 shows a screen shot of a study prototype. It shows an editor for questions, options and criteria

as well as a table editor to relate the different options and criteria for a certain question. Apart from a table representation of the design space, a graphical representation of the Q-O-C relation, see (MacLean et al. 1991), has been developed in the prototype. Unfortunately, the work has not yet been included in EUTERPE.

The screenshot shows the 'QOC GRAPHICAL ENV.' window within the 'GTA TOOL' application. The window displays the following table:

OPTIONS/CRITERIA	Criterion 1.2	Criterion 1.3	Criterion 5
Option 1	High Satisfaction	Middle Satisfaction	Low Satisfaction
Option 5.1	Low Satisfaction	High Satisfaction	High Satisfaction
Option 5.2	High Satisfaction	High Satisfaction	Low Satisfaction

Below the table are two buttons: 'Add Row' and 'Add Column'. The window also shows the question 'Question 4.1.1' and the 'Best Option : Option 5.1'.

Figure 8.10: Diagram

8.5.2 Adding support for simulation

So far, EUTERPE mainly focuses on editing representations. The analysis features still only use the traditional representations such as trees. A more dynamic way of analyzing or evaluating a task model is simulation. Simulation allows designers to see the proportional timing of tasks which is otherwise only indicated in task attributes. Designers can use simulation to test their models but sometimes other stakeholders can also benefit from simulations. For example, to get a dynamic representation of the processes in the task domain. When using an appropriate visualization, domain experts or clients can use simulations to evaluate the models.

In EUTERPE, simulation has not been added yet. Some initial investigation has been done to see whether a task simulator such as ODSN (Szwilius 1997, Bomsdorf & Szwilius 1999a) could be used in EUTERPE. The translation of EUTERPE's models to ODSN is possible although some task attributes would need a more precise definition to specify the exact timing semantics. Currently, EUTERPE allows designers to specify additional time constructors which causes problems in the translation process because there is not yet a way to define the semantics of them. If a restricted set of predefined constructors was used, the translation to ODSN would be straightforward.

8.6 EUTERPE in Use

Over the years EUTERPE has been used by quite some people. This section discusses our experiences in using EUTERPE in practice. Our primary source for experiences is in education at universities. In the Netherlands, the tool was used for several years at the Vrije Universiteit, the Technische Universiteit Delft, the Technische Universiteit Twente, and the Dutch Open University. Additionally, it was used at the University of Cluj in Romania. In all cases, the tool was offered as a possibly useful aid and the usage was not always mandatory. Most students quickly understood its usefulness especially for time consuming design activities such as tree editing. In all cases, at least several students used it. Although EUTERPE has also been used in industry it is difficult to assess the experiences.

The tool has been evaluated using a short questionnaire. Students reported their opinions and gave suggestions for improvements. However, speaking with the students and analyzing the complaints they reported, proved more useful for the development of the tool. The following issues came up in all evaluations and most versions of the tool:

- **Usability of the tool, more is better..**

When people use the program, the first complaints after bug reports concern the usability of the tool. Users basically demand a fully developed bug free tool including extensive editing facilities, undo and exporting mechanisms. Although there were no fatal usability problems in the sense that users could not perform their tasks, they always came with things that could be improved. As soon as those were implemented, they found new issues...

- **Multiple Representations**

People really need multiple representations, especially when the tool is used by people who do not have a Computer Science background. The latter type of designers frequently apply other representations such as process or flow diagrams. Hierarchical diagrams are not easily understood by everyone. Additional wishes were to include scenarios and simulation. Concerning representations for dialog modeling, it showed that a NUAN editor helps to describe the dialog structure but it is not a completely satisfactory solution. For many people, sketches turned out to be more useful to document detailed design ideas, despite the lack of precision.

- **Keeping up with our latest research**

Ideas for better representations have to be developed before a tool can support them. In some cases, it was cumbersome that the tool only realized a subset of the ideas we had. For example, goals as a separate concept did not make it into the tool while it was important in many studies. The same holds for the addition of workflow representations.

- **Improved Modeling**

Since our initial design already contained a broad range of concepts and relationships, there was no lack of constructs. Only in a few cases it turned out that there

was still room for improvement in terms of modeling. For example, being able to have multiple references to the same task within a task structure. Although this is supported by the Prolog implementation, it was not possible in the interface itself. Another issue that came up was that some designers wanted to build a hierarchy of events, for example, to model several kinds of alarms. This is not (yet) part of the ontology.

- **Extending Dialog Modeling**

Although the dialog modeling support was considered useful, it is not completely satisfactory in general. (N)UAN is powerful but certainly has its problems and limitations (Coutaz et al. 1993). Moreover, visual design is not addressed which is very important in the design process and works well for communicating design ideas. One idea would be to integrate sketches of the visible aspects of interaction with the tables. Additionally, large NUAN specifications become difficult to understand and some form of state transition diagrams may help understanding the complex structure of a complete interface.

These issues give a direction for future developments of EUTERPE. See Chapter 10 for a discussion on future research.

8.7 Implementation of Euterpe - a logic approach

The architecture of EUTERPE has been heavily influenced by the development of new ideas about task analysis. One of the things that became clear at the beginning of the project, was that the exact representations and the fundamental structure of the stored data (what later became the ontology) were going to change during the project. The theoretical aspects of task modeling were not fully stable and new ideas were constantly being developed. It was foreseen that the tool had to be flexible so that changes or new ideas would not require a major rewrite of code. Considering the fact that we wanted to use our task world ontology as the basis of the tool and the fact that consistency checking was also a goal, a *logic* approach was considered. Logic languages such as Prolog are very flexible when exploring different knowledge structures. When procedural languages such as Java or C++ would be used, changes to data structures would have a big impact on other code causing a higher risk on stability of the tool. For example, adding a relationship would require only a couple of lines of code in Prolog but in C++ it would require relatively much more code.

After an initial survey, it was decided to use a Prolog variant called Object Prolog, an extension offered by SICStus Prolog. At the time the work on the ontology started, the object oriented variant was the most obvious solution to operationalize the ontology. The Prolog engine could then serve as a database in normal use but could also facilitate more complex model checking. Since a logic approach was preferred there were several options for dealing with the graphical user interface (GUI) that should be connected to the Prolog component. The options that were considered are:

- Using Prolog with GUI additions.

- Using Java for the GUI and communicate with the Prolog system through native interfaces.
- Using C++ for the GUI and communicate through the C API of Prolog.

The first option was discarded because of the complex GUI aspects that were planned. Prolog with GUI extensions is not suitable for complex user interfaces with multimedia facilities. The second option was explored extensively but also discarded. ¹After some prototyping it turned out that Java was not mature enough for such a project² and that a stable development environment was needed. Since the tool was targeted for the MS Windows platform, the natural choice was then for a mature C++ development environment with advanced GUI support: Microsoft Visual C++ 5. The Prolog system that was chosen was SICStus Prolog because it was the only system that contained Object Prolog, an object-oriented variant of Prolog that allowed for a natural mapping of the task world ontology. The Prolog system contains a runtime system which is a library that can be used with any other application.

The next sections discuss the architecture of the tool, the embedding of a Prolog system in a C++ environment and the mapping of the task world ontology to Object Prolog.

8.7.1 A model-view-controller architecture

The main architecture of EUTERPE is based on the Model-View-Controller pattern (Gamma et al. 1995). The *"model"* part consists of the document class which uses the Prolog engine directly. The Prolog engine contains all the real model data and the document class maintains temporary runtime data on the C++ side. Everything the designer creates is stored in the Prolog engine and the Prolog engine also contains the main IO routines. The designers see *"views"* of the data that can be edited as well, the *control* part. Such an architecture is easily built when the Microsoft Foundation Classes are used. Visual C++ creates all the necessary classes when a so called MDI (Multiple Document Interface) application is created.

The main classes that are generated by Visual C++ are the document class (CGTA-Doc) and the view/control class (CHierarchyViewer), see Figure 8.11. The document class manages all document functionality such as serialization, view management, and adding/modifying elements. The HierarchyViewer class contains the TreeViewer class that shows the task trees as well as other tree structures. Drawing the tree is partially done by the treeviewer and partially by the NodePainter class. The NodePainter only draws the nodes of the tree and the appearance of the node is determined by the values of the NodeStyle structure. The TreeViewer class is not only used to show the task trees but is also used the other hierarchical structures as well as for list structures, for example the list of agents.

¹The results of the Java exploration were re-used in the development of the Java tree applet.

²This was when JDK 1.0 had just been released in 1996

Logical View

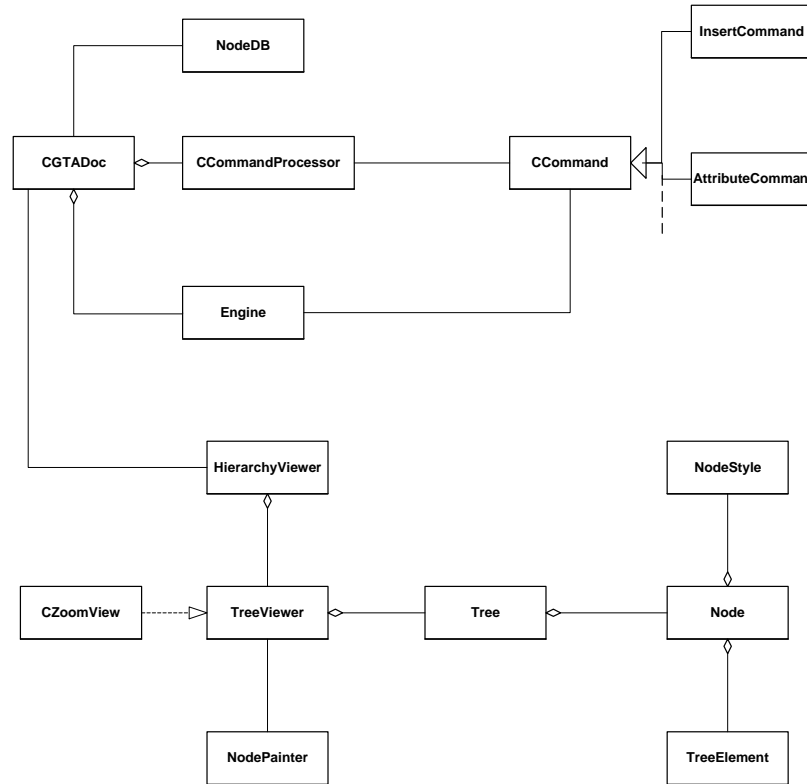


Figure 8.11: A Partial Class diagram of the main Euterpe components

8.7.2 Embedding a Prolog engine

Most of the Prolog systems on the market have been designed to be used as stand alone systems. All of them have a C API that can be used to access the Prolog engine externally. Such an API deals with both external access as well as the implementation of new Prolog clauses. However, the C API is very low level and deals with single terms and sub terms. An arbitrary term needs to be built up by adding each atom or variable using function calls. To create a term with several sub terms consequently requires numerous C function calls. Once a query has been executed, the term has to be peeled off to access the result variables. Whereas in each Prolog system the user can enter terms on the command line, this is not possible in the C API³. Another important issue was support for multiple engines. If a Prolog system is embedded it would be very convenient to have multiple engines i.e. multiple query stacks. That way each process can use its own query engine without interfering with others. However, this is

³In 1999, SWI Prolog started to add such functionality to the engine.

not possible with the current Prolog systems⁴.

```
class EngineException {
public:
    EngineException (char *);
    EngineException (char *, char *);
    char *toString (void);
};

class Engine {
public:
    Engine (void);
    ~Engine (void);

    // Feed the engine
    int    Load(const char* fileName);
    int    consultFile (const char *filename);
    void    Restart();
    void    Initialize(void);

    // Querying function behave as printf functions
    int    Query (const char *query_text, ...);
    int    openQuery (const char *query_text, ...);
    int    queryCutFail(const char *query_text, ...);
    int    nextSolution(void);
    int    closeQuery();

    // Getting results
    int    getVarIndex(const char *var);
    void    printVariables (void);
    char*    varToString(const char *var);
    char*    copyVarToString(const char *var);
};
```

Figure 8.12: The Engine class

The C API problems were addressed by wrapping the C API in a C++ engine object that supports more high level interaction with a Prolog system. It was decided to make a string based interface so that arbitrary terms can easily be entered. For getting results, the values of variables can be directly accessed using the variable name in the original query term. In total this has led to very short code for doing queries. Figure 8.12 shows the latest version of the **Engine** class. The most used calls are *Query()* to do a query and *varToString()* to get the results. The *Query()* call was implemented using some built-in Prolog clauses that can parse arbitrary terms. This way the terms did not have to be built up using the C API calls. The *varToString()* call gives the results by merely specifying the variable name used in the last done query. This eliminated complex code for unpacking terms. For iterating on solutions variants of the *Query()* call can be used. In order to make debugging easier, the **EngineException** class was created. Unfortunately, the runtime system often crashed when compiled Prolog code was buggy and the exception class could not help much. Because the Prolog system

⁴In 1999, SWI and AMZI Prolog also added multiple engine support.

was embedded, debugging was more difficult than usual. Nonetheless, the **Engine** class was very easy to use and led to short code. For example, for building a task tree the subtask term only requires minimal code:

```
// find children of this node
Engine *e = new Engine();
e->openQuery ("%s::%s(X)",parent->m_elt->m_ID,rel);
while(e->nextSolution()) {
    newNode = new Node(e->varToString("X"),parent->m_elt-m_etype,this);
    newNode->m_parentRelationship = rel;
    tree->insert(newNode,parent);
}
e->closeQuery();

// Process all children
current = parent->m_firstChild;
while(current!=NULL) {
    buildTree(tree,current,rel);
    current = current->m_next;
}
```

Using a *printf* like way (using a parameterized string with several arguments), a query can be specified with the *openQuery()* call. The *while* loop iterates as long as there are solutions. For each solution the result of variable *X* is used to create nodes in a tree data structure. Using a breadth-first search the whole tree is built up.

8.7.3 Mapping the ontology to Object Prolog

Object Prolog is an object-oriented Prolog that supports encapsulation and inheritance. Polymorphism is not supported since Prolog is an untyped language. The ontology has been specified in Object Prolog by creating a class object for each concept. Using dynamic predicates the relationships between concepts have been defined. An instantiation of an object can then assert a new fact using the dynamic relationship. One advantage of Object Prolog is that there is no distinction between classes and instantiations, they simply form one hierarchy. So as soon as an object is declared it can be used (as an instance) but it can also be used as a class to create new instances. This allows for high flexibility when building a system.

For each concept a class is defined and a base class is used to capture common attributes. The *gta_task* class contains all the task clauses. The following code fragment shows the main declaration with the attributes and dynamic predicates for the possible relationships with a task object. Using the *add_sub_task* predicate the task hierarchy can be created, for example, using the term "test_task_45::add_sub_task(test_task_48)"⁵.

```
% Definition for a task structure
gta_task::{
    super(gta_base) &
    attributes([goal([],duration([],frequency([],start([],stop([],
```

⁵Where test_task_45 is an internal object identifier that has been created before

```

        constr([],initial([],final([],task_rel([],motor([]),
        mental([],task_type("other")))) &

dynamic sub_task/1&
dynamic used_object/1&
dynamic performed_by/1&
dynamic triggers/2&

...
% establish subtask hierarchy
add_sub_task(X) :-
    assert(sub_task(X)) &
remove_sub_task(X) :-
    retract(sub_task(X)) &
get_subtasks(L) :-
    :setof(X,sub_task(X),L) &

...
% IO routines
write_data :-
    self(Me),
    :write(gta_task::new(Me)),:write(' '),:nl,
    write_base,write_attr,!,:nl,
    ((:setof(X,Me::sub_task(X),L),write_subtasks(L));true),!,
    ((:setof(Y,Me::used_object(Y),M),write_objects(M));true),!,
    ((:setof(Z,Me::performed_by(Z),N),write_agent(N));true),!,
    ((:setof(Q/O,Me::triggers(Q,O),P),write_tasks(P));true) &

```

Another big advantage of Prolog is that it is untyped. When writing a specification to a file all objects are dumped using the *write_data* predicate. When the file is read the terms can be read in any order because the object identifiers can be used even when the object does not exist yet. Relationships between non-existing objects can be defined without problems in Prolog. Hence, saving a specification consists of writing all the terms needed to create the same specification to a file. This way there is no fixed file format and new terms can be added without any effect on the reading routines.

For implementing the analysis features the concepts and relationships could be directly inspected using meta-predicates such as *is_instance*. Object Prolog contains some meta level predicates that allow for inspection of the whole inheritance hierarchy. An evaluation constraint in the user interface leads to a call of one term with the selected variables. The following example is the implementation of the heuristic "objects used by role X":

```

object_used_by_role(Session, ID, Role) :-
    gta_object::is_instance(Session, ID),
    gta_role::is_instance(Session, Role),
    gta_agent::is_instance(Session, Agent),
    Agent::plays_role(Role),
    ID::user(Agent, _).

```

Object Prolog allows an easy mapping of the task world ontology. Relationships can be easily added and all the reasoning features of Prolog can be used. This proved very flexible in practice.

8.7.4 Prolog and application functions

Since the tool is a mix of Prolog and C++ code, there was concern for the separation between code that dealt with Prolog and code that did not. Ideally, only the document class would use the Prolog engine and could form an abstraction layer. In practice this could not be achieved. The use of the Prolog engine was limited to mainly the *document* object, the *command* objects and some other objects for efficiency reasons. In this way, the GUI can be programmed in the "normal" way and the connection is made transparent. For example when a node is inserted in a tree the InsertCommand object handles all the Prolog queries that are necessary while the C++ tree class performs a normal tree insert. When the node parent pointer is changed a command object changes the appropriate Prolog terms. The use of command objects is conform the **Command** pattern (Gamma et al. 1995). It is the usual way to implement a multilevel undo in GUI applications. It consists of an abstract class for commands and a queue in which each command is stored before it is executed. Undo and redo then come down to navigating through the queue of commands. In EUTERPE, the abstract class Command and the CommandProcessor deal with the administrative aspects. The individual command classes implement the *Do()* and *Undo()* methods and must be prepared for multiple undos and redos.

One of the consequences of having multiple views is that certain data structures are often regenerated. It is therefore dangerous to store pointers. For example, command objects cannot store object pointers because they can become invalid when objects are regenerated. Therefore, a database of symbolic object names is maintained and used in the command object. This way a valid object pointer can always be retrieved. The symbolic name is the unique Prolog object identifier.

8.8 Lessons Learned

During the whole development we learned several things about testing our research ideas in such a prototype tool. Some concern implementation but most of them are issues that may be relevant for others that plan to develop tool support for design activities:

- **Documentation is communication.** During the project, printing and export functionality turned out to be more important than expected. Designers need to see things on paper or in other structured documents and share them with colleagues. On screen models are not sufficient to consolidate and communicate design knowledge. Unfortunately, during printing only the un-collapsed tree could be printed which was not always what designers wanted.
- **Prolog is in its infancy.** Choosing Prolog was the best choice but the Prolog system itself had many problems. In general, Prolog systems are not built to function as a subsystem in an application. The interfacing is cumbersome, multiple engines are hardly supported and debugging is difficult. Some of the latest

Prolog implementations are addressing these problems but there is still more development needed.

- **Interfacing with Office packages.** Task modeling tools are used in isolation. There are always other tools that a designer uses and designers expect to be able to transfer results from one tool to another. Especially, interfacing with word processing or drawing packages is important. Image quality is crucial and being able to make some last minute changes is considered important.
- **A multiple view approach.** HCI is a multi-disciplinary field and using multiple views/representations can bridge the gaps between the disciplines.
- **Creating better models.** The task world ontology is implicitly shown in the various representations and provides users with a broad perspective. If there is a template with standard fields, designers think about those aspects while they would not have thought about them otherwise. For some designers, certain concepts were entirely new but most of them reacted positively to them. Hence, designers made richer models partly because the tool gave them some "hints". This made designers more effective, especially novices.
- **Users evaluate products, not ideas.** If users are asked for their opinions about tools they evaluate the whole product. They can not comment well on ideas on which the tool is based. This is true even when the users are designers doing task analysis. As soon as usability problems arise or important functionality is missing, the judgement is negative. For example, some users who could perfectly use the tool to model gave a negative opinion just because they "couldn't use the diagrams in Word". Apparently, some task analysts consider such functionality crucial. It becomes hard to evaluate the ideas behind the tool when this kind of functionality is missing. Using an incomplete prototype to validate research ideas about modeling is problematic. Users can hardly see through all the usability problems of the prototype and can only comment on those aspects. "*It's merely a prototype.*" is not a valid argument to users.

In general, it can be concluded that tool support is really needed and can have a positive effect on the design process. EUTERPE is a fairly successful attempt to provide such support and has shown the direction for further development.

8.9 Summary

This chapter discusses our tool EUTERPE. It has been an exploration into tool support for task-based design. EUTERPE has been a fairly successful prototype and is now used in industry and education. The tool is directly based on the task world ontology i.e. the ontology is internally used in the logical subsystem and the representations are truly views on the available data. EUTERPE offers several representations that each focus on a specific aspect of the task model. Besides task modeling, an editor for NUAN tables

has also been implemented. This way a link between basic tasks and user actions can be made.

Although it has already proven useful in practice, the tool should be improved on many aspects. New representations should be added and the tool must mature more so that it can become a serious product. EUTERPE has shown that there is a clear need for such tools and that it can improve the quality of the task models as well as reduce the time needed to construct them. EUTERPE has set a direction for tool development which needs to be explored further.

Chapter 9

Task-based Design in Practice

9.1 Introduction

This chapter is about how task-based design actually works in practice. During the last few years, we have applied and refined our design method in various case studies. Some were done as part of an educational program and in other cases parts of our method were used in industry. Evaluating an entire method is very difficult to do, especially in a realistic setting. This chapter will therefore not discuss hard research findings but will instead report on our experiences with our design method.

When we used the method in education we had the opportunity to supervise the correct application of the method and to correct when necessary. Even in such situations, task-based design is not without problems. Some activities of the method are more difficult to perform than others. When trying to use the method in industry, such a controlled context is not feasible.

Our method is intended to be used in practice by industry. However, companies have their own methods and will not easily change their established and "proven" practices. It is therefore important that their situation is understood well, if any new method is to be adopted. Nonetheless, we have had some experiences using parts of the method in industry as an addition to existing methods. While this may not tell much about whether our method works well or not, it shows the situation companies are in and how they approach new methods. This chapter first discusses some case studies where our method has been applied. Then several issues in applying task-based design in practice are discussed.

9.2 Applications in Industry

In this section, we discuss two cases where task-based design was applied in an industrial context. It shows what kind of issues need to be faced when performing a

task analysis and the kind of results that can be obtained using a task-based design approach. The first case discusses a project for the Dutch social security system and the second discusses a redesign of a security system.

9.2.1 Dutch social security

An early case study was the case of the Dutch Social Security (van der Veer, Hoeve & Lenting 1996). It concerned only the first phase of task analysis, from initial inventori- sation and collection of information to the formal representation of the resulting task model of the current situation, task model 1. The Dutch social security system provides a large number of support facilities for citizens in diverse situations of financial and other needs. A special type of task that is frequently performed by the social secu- rity system in Dutch municipalities is the settlement of requests for support of primary subsistence. This task requires several levels of information collection and subsequent decisions.

At the time, the Dutch social security system as a whole was a traditional office orga- nization supported mainly by some database facilities. In order to work more efficient, integrated implementation of workflow management tools and other forms of coopera- tion technology are being considered. Some prototyping systems are being constructed and installed in certain offices. Our involvement was concerned with performing a task analysis with the aim to describe the current situation, as a start for a systematic design effort.

At the start of our task analysis of the handling of requests for primary support of sub- sistence in the social security case we profited from the fact that a group of colleagues already had been working in the same offices. A detailed analysis was available of the characteristics of actors and regular roles and, on the other hand, the office work- ers were aware of the status and possible obtrusive presence of ergonomic analysts. Our main data collection started with some interviews of the type proposed by Se- billotte (Sebillotte 1995). During the phase of knowledge elicitation, we found out that an enormous variety of strategies existed and that workers evidently were unable to provide a generic account of the details of their work. Additionally, an extensive ethnographic study (Jordan 1996) was done over a period of 2 months. The data was then systematically analyzed using interaction analysis (Jordan 1996). Hot spots were recorded on video and transcribed. They were then interpreted and the interpretations were combined into more general descriptions.

Although the interpretation of the records using our conceptual framework was rela- tively straight-forward, serious problems arose when trying to construct models from different accounts of the same event, both when combining observations from different actors in the same role, and from combining observations from a single actor for dif- ferent occurrences of the event. This is in contrast with the situation that is normally reported from task modeling approaches of the knowledge type (MAD, GTA), where professionals are found to be consistent and in agreement between each other where it concerns the task structure. The behavior in the case of our complex system turned out to be strongly influenced by situational factors as well as by attributes of the partners

in the interaction. The first addition we had to make concerned the way to represent the structure of a certain task. In the past, we used constructors in the sense of Scapin (Scapin & Pierret-Golbreich 1989), where notations like LOOP, PAR, OPT, indicated that certain subtasks are performed in a loop, in parallel, or optional, though we normally adjusted the actual set to the actors' perception of the task domain. Constructors are used both in the representation of task object structures, and in representations of relational graphs that show task decompositions. In the present case, however, we needed several much more elaborate constructors, especially in cases where an employee was interacting with the applicant for support. In such cases, many constructors turned out to be dependent on *situational* characteristics and therefore needed to be able describe conditions.

Moreover, in the formal representation we still found the need to frequently add notes to certain constructors where a single verbal label was insufficient to make the formal representation interpretable for members of the design team. Another addition for the task relational graphs indicated the (frequently occurring) interruption of a sequence of related tasks by periods of events that were unrelated to the current process. A last addition to the task graph concerns the frequent situation where a task is decomposed into a set of subtasks, each of which is performed based on a criterion that is valid for that subtask only. In order to provide a readable and valid representation, we had to explicitly introduce this combined set of criteria and subtasks as part of the task hierarchy.

Task-based design, in particular the task analysis part, showed to be valid for the highly complex and highly situated type of process that we analyzed. The methods of data collection in this case were mainly of ethnographic nature, where a long informal introduction and subsequent interaction analysis turned out to be feasible. Some formalisms needed to be adjusted to allow a representation of situated and person-dependent substructures of tasks and events. The task analysis as a whole was in fact checked for validity by the subsequent phases of design. Based on the model of the current task situation (task model 1), an analysis of the detected conflicts and problems, a new task structure of the process was specified. This specification was formalized with the same extended formalism, which allowed the development of scenarios for the process and especially for the sub processes that were proposed to be redesigned. The scenarios were implemented by developing video fragments via role-playing (acting out the high level task scenarios). Both the original task model 1 and the scenarios that represented the proposed re-design were judged by the workers as realistic descriptions, resp., sensible proposals to solve the conflicts and problems. The method seems to abstract the rich task insight that the analyst acquired via ethnography in such a way that the important aspects were saved, and thus were available for systematic redesign.

9.2.2 Seibersdorf

Another case where our task-based design was applied was at the Austrian Research Centre Seibersdorf (ARCS) mainly for the redesign of a security system produced and marketed by Philips Industry, Austria (van Loo et al. 1999). The system is used at many

sites such as banks, railways and electric power plants. The main problem in this case is related to the confidentiality of the knowledge of the task domain. It is the actual security systems in use in these companies that are the basis for our knowledge of the task domain. Obviously, none of these companies is eager to have details of its security management situation and security procedures being made available to outsiders, even if they are employed by a company that designs their system.

Securing large objects like factories, museums, banks or airfields is no small feat. Monitoring and controlling areas in these objects is done with the help of movement-detection systems, video camera systems, access control systems, fire detector systems, elevator control systems etc. In practice, all the information from these (sub-)systems is led to a control room where human operators have to respond appropriately to (alarm)signals. Keeping an overview on the building status becomes almost unmanageable in complex combinations of sub-systems. To support the operator in monitoring the state of the secured object and to integrate the different subsystems into one system, the sCalable Security System (CSS) has been developed. The CSS integrates the information flow from -and to- all subsystems in one central computer system with a generic (graphical) user interface available on several workstations.

Little knowledge was available about how the current system is used and what kind of problems the users have with the system and/or User Interface in performing their work. To gain more insight on this topic, the first phase of task-based design was performed. The analysis focused on the use of the system by the end-users (i.e. operators, porters and system-managers) in their actual use-environments (factory, chemical factory and office buildings were visited). Goal of the analysis was to gain insight in the current use of the system and to propose directions of change/extensions to improve the (practical) usability of the system. In this case study, the physical layout of the control rooms was essential in detecting problems, see Figure 9.1. Many systems assumed that an operator would always be in viewing distance of the screen, which was not the case.

Participating observations and semi-directed interviews were first carried out at two sites that seemed relevant. Getting the necessary co-operation of the managers and employees of the visited sites took some effort. Explaining the goal of the visits by telling some characteristic cases and ensuring that the resulting information remained confidential proved to be helpful in this process. At the end of the visits most managers and employees were enthusiastic about the fact that finally someone took the time to listen to them, and took their grievances seriously. Due to the sensitive nature of the work observed (security control rooms inside the objects to be secured) camera recordings (as usual in observations) and tape recordings of interviews were (almost) impossible.

Crucial to the acceptance of task-based design in the project organization were the following factors:

1. External funding. The analysis of the CSS was carried out in the context of the European OLOS Project (EC Human Capital Programme CHRX-CT94-0577). This made it possible for ARCS to get to know task-based design with low financial risks. The results were the main ground for extending the appointment

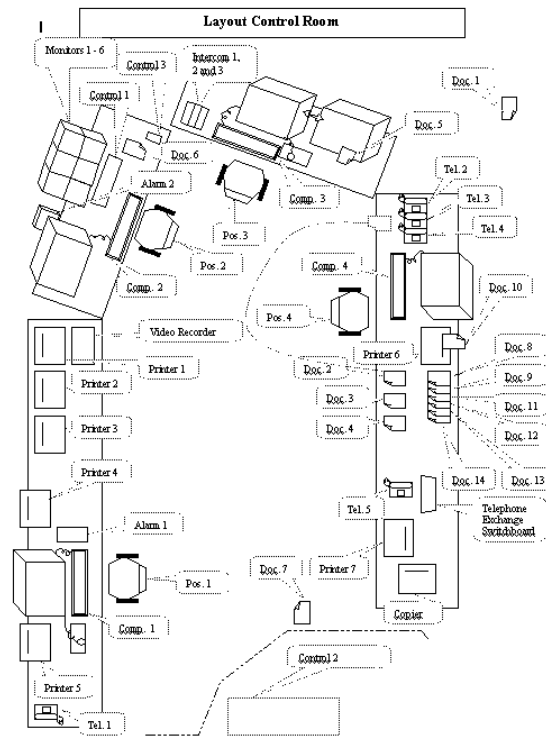


Figure 9.1: Layout of a control room

of the researcher at ARCS, outside European funding, to apply the method also on other projects.

2. Complementary expertise. The results of the analysis made visible a gap in the acquired expertise in the project organization: The direct translation of context-of-use characteristics in design information.
3. Task-based design being a method. It offers a more solid base to work and communicate from. One does not start from scratch. Also, in getting support for gathering your data it is also very handy to mention that you are using a method. In a way, one abuses connotations of the method-concept like 'objectivity' and 'being systematic'; notions that make a technology oriented organization more receptive.

During analysis the conceptual framework and the ontology proved to be the greatest help. It worked as a kind of 'check-list' to focus attention on things that matter in performing tasks. During analysis it was felt that methodical support was lacking, although this was not a big problem. What is needed more is an openness on the side of the analyst in learning to understand why people do something. One has to

be able to 'just enter a room' and start observing something, which might turn out useless or useful for the analysis. One has to be able to deal with the openended-ness associated with a more 'ethnographic' style of performing analysis. That the method leaves enough room for this without losing the focus for system design as witnessed in the framework and later stages, is regarded as a strong point, not as a weak one.

In this project, not the complete task-based design approach was used since we were not involved from the beginning of the project. This is, however, not considered a problem: as part of a larger design team and project culture the results of the analysis stated in the conceptual framework (work, people, environment) and the translation of these findings into system design are incorporated in the project. Design recommendations were done by iteratively developing sketches of design alternatives. Especially the 'translation' was crucial, and this turned out to be part of the core benefit of the method in our case. The outcomes of the analysis were translated in concrete recommendations for interface and system design. This way the results were regarded very beneficial to the overall project. One acts as an intermediary between users and system designers and one has to be able to speak both languages. If one stays too far away from either the conceptual world of the users or of the designers the method won't work. Reactions to the method are very positive and money is spent to carry on the work within the framework. However, what is still lacking is more 'awareness' in the company of what it means to perform task analysis's at the customer and how it should be integrated in the already present system design culture. This design case showed that task-based design indeed delivers the kind of results that are expected from such a method.

9.3 Application Issues of Task-based Design

During the application of task-based design (or a subset thereof) in both educational and industrial contexts, several application issues were encountered. In the next sections, we discuss these issues. These issues all reoccurred to a certain extent and together give a critical view on the practical value of task-based design.

9.3.1 Performing task analysis

Task analysis and task modeling are at the heart of task-based design. Even though the tools and techniques we developed have improved task analysis, there are still several issues that occur in practice. For most designers task analysis is a new technique which needs to be learned. Some problems are related to how designers learn such a technique. Other issues are related to difficulties with the techniques themselves. The issues are:

- **The need for a methodology for practitioners.** Most of our documentation on task analysis is in the form of research publications. They are consequently written using a certain style required for scientific publications. However, this

is often not the best way to reach practitioners. Practitioners are often not interested in the scientific argumentation that is common for such publications and are more practice oriented. Additionally, topics such as a task world ontology are difficult to understand and the attitude of designers is focussed towards direct application, i.e. "Ok, but what do I do now?". If the way practitioners work is really to be changed, our "story" needs to be adjusted so that it appeals more to practitioners. Practitioners need a lot more practical help. It is probably better to give them diagramming techniques and methodological support that tells them which techniques to use and how to model in the proper way. Techniques need to be outlined with many examples and should discuss the detailed steps of creating models.

- **Task or Goal?** Distinguishing between tasks and goals is crucial for a good task analysis but it is very difficult to understand for practitioners. During the development it is not always directly clear what the goals are. Some may initially be modeled as tasks while it later becomes clear that they are actually goals. It takes time to realize such aspects and it is one of the most valuable outcomes of the modeling exercise. Practitioners may sometimes be impatient when facing such issues and their background clutters the issue even more. For engineers that develop the technology, tasks or goals are easily mistaken for system functions. The internals of the system are so familiar to them that it may become difficult to make a shift of thought and 'forget' about the system internals for a moment.
- **Role or Agent?** The distinction between roles and agents is also difficult to understand. If only one existed in the technique, most people are comfortable with roles. However, when both are present designers are easily confused. The most important one to model is probably a role. It is often not necessary or useful to model agents.
- **Modeling is not easy.** Making and understanding hierarchical models is not always easy. In (Cooper 1995) it is reported that end-users also often have difficulties dealing with hierarchical structures in user interfaces. For some designers, it is more natural to use flow diagrams because they are more used to it. Additionally, translating data from observational studies or other sources into models is not a trivial task. Designers need to learn to recognize the "concepts", the important parts in the raw data, and model them.
- **Task Analysis may not be possible.** In some domains, it may turn out to be impossible to do a task analysis simply because it is not allowed. For example, the Dutch company Holland Signaal develops radar systems for naval ships but because of military security strategies, is not allowed to enter the ships and see how people work and how the work environment is. In other cases such as discussed in the previous sections, it may not be possible to record interviews or film on location.
- **Modeling Task Interruptions.** Often tasks are being interrupted and continued later. In the current formalism it is not possible to model this. Extensions are needed to adequately describe task interruptions and their possible continuation.

- **Modeling Conditional Task Flow.** In some cases the task flow is determined by conditions. While these can be modeled using start conditions of tasks, they are not visible enough in the task flow representations. Usually, conditional task flow is related to decision making tasks.
- **Modeling Task Strategies.** Task experts often use strategies to deal with their tasks. It allows them to be more efficient and effective. A strategy could be modeled as a specific separate task flow/tree but this is not entirely satisfactory. Strategies are often based on implicit knowledge that is very difficult to make explicit.

9.3.2 Integration with current design practice

Currently, most companies are not doing task-based design. The design method we developed would ideally be widely used in practice but making companies switch to a new method is easier said than done. In constructing our method, we are designing from an idealistic point of view. In practice, there are many other factors to consider. Business goals influence the importance of designing for usability and most companies already have a design method that may be considered satisfactory by them. There is a legacy problem of people and current techniques. Integration issues include:

- **Task-based design and other software design methodologies.** Most companies already have their own design methods. It is unlikely to expect them to easily adopt a new design method. For some industries, object-oriented software design is still new and switching to a task oriented view for the user interface is difficult. Some wonder whether it is realistic to focus so much on the GUI while the software construction part in itself is already complicated enough for practitioners! If task-based design is to be integrated in current software design practice, the position of the TB-UID activities need to be "positioned" in relation to software design activities. The task modeling and analysis activities can be done previous to or as part of a requirements phase and should not be very problematic. The later activities concerning detailed design and iterative evaluation should at least be done in parallel with internal software design. However, this requires that both parallel "tracks" are consistent with each other which is not a trivial activity.
- **Throw-away prototyping.** Many people nowadays use a Rapid Application Development (RAD) (Martin 1991) methodology where the prototype evolves into the final product. This makes it very difficult for designers to throw away prototypes because they want to re-use what they have. However, in some cases it may be very desirable to throw away a prototype because the "main idea" was wrong. From another point of view, prototyping in the sense of task-based design is just developing a special representation of detailed design specifications for early evaluation. For the sake of task-based design, it is not acceptable if the development of this representation is influenced or constrained by issues like code reuse.

- **Arrogance or ignorance?** *"We know how our products are used or what the user wants to do"* is an attitude typically found in industry. However, there is often no data to support such claims. Sales figures seem to be the most important indicator for usability to many companies. Some really *think* that the usability of their product is good, mainly because they *feel* they design usable systems. Often no testing whatsoever has been done to confirm this.

9.3.3 Designing for usability

Designing for usability and doing task-based GUI design also raises issues related to this new way of working. In contrast to what HCI researchers might hope, usability is not a first class citizen among the design goals. Although the mentality of practitioners towards usability may be changing with the growing developments of interactive media, some fundamental issues will not easily change:

- **Usability may not influence sales.** For many products, usability is not a selling point. People rarely choose a product because of usability. Who ever asks for the remote control when they buy a TV set? The user interface is important but seems the least important aspect when deciding to buy a product or not. After all, if the product is not functional enough, usability does not even matter. A product seems to be ready as soon as it is functional and stable enough provided the usability is not too bad. Only for special contexts, usability is important enough to get the attention. Such systems are often used by highly skilled professionals where productivity and effectiveness is utmost important. The list of potential advantages as listed in chapter 2 is perhaps motivating but in practice hard to quantify. In (Mayhew & Mantei 1994), an attempt is made to discuss justification of the costs of designing for usability but the calculations are tentative. We are not far enough in proving that it is actually worth it. Doing a task analysis may often result in just a few design changes. Humans are excellent in overcoming usability problems, as long as the products are functional enough. Research on measuring usability needs to be more precise. Task modeling needs to become more common and design needs to be learned from successful design (and less in artistic design that evolves around vision and art.) If the product sells even though the usability is not great, for the company there is no problem.
- **Reuse in HCI?** If designers are to systematically build better systems it is crucial that some form of reuse occurs in user interface design. Nowadays, there is a high dependency on local experts, which is a risk for constant high quality in use. Guidelines have not been very successful in ensuring design quality. Patterns may turn out to become a better means for reuse in design. After all, UID is mostly engineering and not art, although many seem to disagree. Many tasks are found over and over again in applications, and they form patterns of solutions. Any application that uses those tasks should reuse those solutions; it is simply engineering. With the current techniques, for many designers it feels like they need to start from scratch for each design case.

9.3.4 Developing design alternatives

After the task modeling and analysis has been done, design alternatives need to be developed. This is both an engineering and a creative task. In practice, this is a difficult step where a lot of iteration is needed. We have applied techniques such as sketching, scenarios and formal notations for this purpose. Although each of these techniques has its pros and cons, some issues remain unsolved. The design patterns discussed in chapter 6 were developed to deal with some of the issues described here.

- **Applying design knowledge.** Guidelines and other techniques such as Mayhew's design strategies (Mayhew 1992) to choose the right dialog styles, are not easy to use, especially for novices. Expert designers rely on their own experiences with user interfaces and mainly design by copying.
- **Creativity versus formal methods.** In the detailed design phase, the design is worked out in such detail that a prototype can be made out of the specifications and finally a full-blown implementation. Representations for this phase include sketches, screen shots, interactive prototypes, hardware mock-ups and NUAN. Especially in this phase the combination of formal and informal representations is working out well. The creative ideas are born in the informal representations and then become more detailed at each iteration. At the end of the process a detailed specification is handed to the implementers and this specification needs to be unambiguous. If not, the final product may not have the intended look and behavior.

Formal methods allow precision and informal methods allow gradual refinement and creative interpretation. At some point in the design process both are needed. In our experience we found that formal and informal methods can work complementary (van Welie & van der Veer 2000). At some point details need to be worked out and by using formal methods hidden faults and unnoticed issues may be found. Another issue is communication. When a design group needs another group's results as input, the work needs to be 100% clear and unambiguous. Using a formal method is never a guarantee for unambiguous descriptions but it makes it easier to understand a description and find imperfections. This is facilitated because formal methods have explicitly defined syntax and semantics.

- **Design Culture.** Many focus on the product and not the use of the product. In some disciplines there is still a tendency to let the artistic values prevail. Such designers let their artistic vision prevail over usability. For some it is a mind shift to think about the user, for example certain engineers may think "*as long as it works it is ok*". Additionally, within a company, the marketing department gives an important push to the design process. Features sell even if these are not always what the user needs in his task, "*we can do this feature so let's put it in!*"
- **Designing in Groups.** In task-based design, team design is a requirement. Teams need to work together in order to make the right decisions and obtain a high quality end product. For example, in a typical project we found the following teams:

1. Task Analysis Group
2. Detailed Design Group
3. Evaluation Group
4. Scenario and Prototyping Group
5. Management Group

We found that it is important to have one independent group that manages all other groups. The manager or management group guards the time schedules, deals with delays and group conflicts and the overall project documentation. Management negotiates with the client and other stakeholders such as marketing and production. Management's main purpose is to guard the dialog between disciplines as well as between design phases. Management is monitoring the use of representation to communicate between design sub-disciplines, as well as between the design team and the users, stakeholders and the client.

Designing in teams can work well and potentially has several advantages. First of all, working in groups on separate issues creates a competition effect. Each group depends on other groups and is being judged by other groups. If one group delivers poor work the other groups immediately remind them. On the other hand if a group has developed really new ideas, it is very disappointing if the other groups just discard their creativity or fail to see its merits. Not surprisingly, it is not unusual that certain groups get really upset by the work of others. However, this even leads to qualitative improvements of the final work as it forces the management as well as the whole team to reconsider and argue misunderstandings and rejected proposals. Another advantage is that in this way the expertise people have is used optimally.

Teaching design requires training the students in the dynamics of design teams. In our educational experiences, an adequate model is when the management team forms the groups and designers have to apply for a position. This way everyone gets to determine what he or she does, within some limits of course. In industry, a comparable "marketing" situation and resource management is in fact common practice.

9.4 Limitations and Critical Success factors

Task-based design is certainly not the best or only solution for all types of interactive systems. There are some limitations that make it less suitable in certain contexts. Typical limitations are:

- **It is sometimes impossible to do task analysis.** There may not be a current task or sometimes access to the current task domain is impossible. For instance, for security reasons it may not be allowed to go to the work environment.

- **Task-based design is applicable for complex interactive systems only.** Our method was developed for complex interactive systems where many actors work together. It is therefore not well suited for systems with high real-time demands or systems where humans are only marginally involved.

For the successful application of task-based design, several factors have turned out to be critical. These success factors include:

- **Management Support.** Task-based design is not commonly accepted. Getting management support is crucial. The direct managers need to be convinced that task-based design will bring the project the necessary benefits. This also depends on the role of usability in the product being developed. For certain products the effort can be justified more easily than for other products. Long term projects that last for more than one year are very suitable for applying task based design all the way. For shorter projects, the effort spent on each design activity needs to be balanced appropriately.
- **A client demand for usability.** If the product that is being developed has a high demand for usability, task-based design is more likely to be accepted and to be successful. For example, in Air Traffic Control or other control room contexts, usability is the key to a successful and safe system. For other products usability is less important which makes it hard to justify the efforts. Products where sales are largely influenced by artistic design (for example many consumer electronics), require a short product life cycle and usability is often not important enough to spend the effort.
- **Skilled Designers.** Task-based design is not common practice and designers really need to get used to that. Only after the method has been applied several times, the method can be used to its full potential. Novice designers make many mistakes and need to learn new skills which naturally takes some time.
- **Using a method.** Even if a method is not perfect it is advisory to stick to one method anyway. It will structure the activities and make it more manageable and acceptable for management.

The success factors are in fact quite general and similar to those found in (Hall & Fenton 1997). One hypothesis is that they apply to any new technology that is to be introduced.

9.5 Summary

This chapter discusses our experiences with applying task-based design in practice. In theory, task-based design solves some of the common problems in user interface design. Validating such a claim in practice is extremely difficult and depends on many uncontrollable factors. Our experiences with TB-UID are very promising, but much

work is needed before industry will adopt it in its current practice. Research needs to refine the methods and techniques so that it becomes easier to apply them. On the other hand, practitioners also need time to get used to a new technique.

Additionally, the methodology needs to be tuned for industry practice. Research needs to validate the techniques and provide better insight into the potential gains of task based design. Task-based design must be able to quantify the gains and show how the activities allow designers to develop usable interfaces in a structured way. Once the client's demand for usability increases the relevance and importance of methods such as task-based design will also increase.

Chapter 10

Conclusions and Future Research

This chapter concludes this thesis. It summarizes the previous chapters that discussed various aspects of *Task-based User Interface Design*. On the one hand this thesis shows the benefits of task-based design but on the other hand it shows that many improvements can still be made. We first summarize the work and state the main contributions of this thesis. Since the field of user interface design is still developing rapidly, we discuss some ideas for future research on task-based design.

10.1 Summary of the Thesis

Task-based design is a method for designing complex interactive systems. Such systems are characterized by having several kinds of users and other stakeholders that are all involved with the system. Designing such a system is a difficult task and our design method is intended to handle such cases. In task-based design, quality is defined as a high level of usability. Usability is a complex concept and has many often confusing interpretations. Therefore we developed a new framework for usability. It gives structure to the various aspects of usability such as what is measurable and how usability is related to tasks. In task-based design, it is used as a reference throughout the process when designing for usability.

Having defined our design goal and quality concept, the design activities need to be defined. As usual in structured methods, we start the design process by doing a thorough analysis of the problem situation. In the case of user interface design this means an analysis of the users, their work and their environment. This process is traditionally called *task analysis*. Performing a task analysis and using the gained knowledge effectively in design is not trivial. It is important to know what aspects of the task world are relevant and hence need to be described. Those aspects are captured by the task world ontology that we developed. Such a clear conceptualization helps designers "see" the

important things when they are collecting data and when they are modeling the task world.

For the actual modeling of the task world, effective representations are needed. Representations should allow designers to describe all the relevant aspects of the task world. We have defined a set of representations that covers all aspects of the task world as defined in the ontology. When the task world has been modeled, designers must look for problems or chances for improvements. This can partially be done by looking at particular aspects of a task model. We defined useful aspects to look at, again using the ontology as a reference, and we operationalized them in our tool EUTERPE.

Since modeling usually takes a lot of effort we developed and tested a tool for task analysis called EUTERPE. It allows designers to build task models and to produce various kinds of documentation such as small web sites including multimedia data or paper documentation. EUTERPE has been used extensively in both industry and education over the last two years.

After the analysis phase, the detailed technology needs to be designed. This involves many aspects such as determining the functionality to be offered, the interaction structure and the presentational aspects. Most importantly the design needs to be based on the *task model*. This is crucial for building systems that support users in their tasks. The goal is to design a system that supports the users in their tasks which is why a task model is necessary. On the other hand, design knowledge is used as well. Design knowledge is usually the expertise of the designers themselves but explicit design knowledge such as guidelines is also frequently used. Using design knowledge prevents designers from making the same mistakes repeatedly and allows them to reuse proven solutions in a new design. Guidelines have turned out to be problematic despite their popularity. We discuss *patterns* as a possibly better way to capture this design knowledge. Patterns describe proven solutions to user problems in a specific context. This incorporation of the context allows patterns to overcome some of the shortcomings guidelines suffer from. To validate this idea we developed a collection of 30 patterns that demonstrate the essence of interaction design patterns. When the collection evolves into a pattern *language*, the design process can rely on both a thorough task analysis *and* design knowledge in the form of a catalog of proven solutions.

We also argue that detailed design needs a mix of formal and informal techniques. Sometimes there is a need for precision while creativity often asks for informal techniques. One of such precise techniques is NUAN (New User Action Notation) which extends UAN. This technique allows bridging the gap between tasks modeling and detailed design. The notation is suitable for task-based design and the link has been described in, again, an ontology.

When the detailed design has evolved into some form of prototype, usability testing usually starts. However, usability evaluation can also be done much earlier when even only task models exist. This thesis does not discuss evaluation techniques in detail since we currently use existing techniques such as walkthroughs, observations and questionnaires. Instead we discuss the possibilities and limitations of various evaluation techniques using our usability framework. When only paper mockups are available it is already possible to ask users to perform their tasks. This usually leads to high level

comments about the interface. When a running prototype has been developed, detailed usability testing can be done. In all of the testing activities we use the usability framework from chapter 2 to measure the correct aspects and to look for problems or possible improvements.

10.2 Contributions

This thesis discusses *Task-based User Interface Design*. Starting with a general outline of the design process, each of the main activities is discussed in detail. Not only the theoretical aspects of the method are discussed but also practical aspects such as representation techniques and tool support. In chapter 1 we discuss the main research questions and indicate problems in two areas: improving the design *process* and improving the final *product*. A total of five main problems is stated together with ideas on how to improve them. We discuss these problems together with our contributions:

- *Improving the available techniques for both task analysis and design.* Developing better techniques has been done by taking an ontological approach. This means that we first investigate the question *what* needs to be modeled. The task world ontology is a meta-level model that describes what we want to model of the task world. The ontology is based on common concepts in task analysis techniques and has been developed further over the duration of the research project. Based on our experiences in industry and education, we are confident that it is sufficiently complete. Using the ontology we have looked at representations for designers to answer the question *how* the task world can be modeled. The *why* question is answered by our investigation into the usability concept where we show that task knowledge is crucial for developing usable systems.
- *Reducing the effort of performing a detailed task analysis.* The effort required is reduced in several ways. First of all, by offering improved techniques and models designers have a much better idea of what and how to model. It gives them conceptual tools to start doing task analysis. Second, the manual activities such as creating and editing the models is supported by our tool EUTERPE. It operationalizes conceptual ideas about task analysis into a practical tool for designers. In practice both in industry and education, EUTERPE has proven to be a useful tool, although many improvements are still possible. Our tool is not the only tool that can be used in task-based design. Therefore we present an overview of tools that could be useful in task-based design.
- *Improving effective use of task analysis results in the design of the actual product.* This is perhaps one of the most difficult problems we tried to address. One aspect we looked at was what kind of questions exist in detailed design that should ideally be answered by a task model. This gives us information about what to model and it is one of the ways to improve the task world ontology. On the other hand, we investigate patterns as a more direct link between user tasks and design solutions. Considering that interaction design patterns are still a new field, this

link has not yet been fully established. The patterns address user problems which are task related but those tasks are still very general. The next step would be to go from a specific task model to patterns via more generic task categories. For describing detailed designs we have extended one of the existing methods, i.e. User Action Notation. Our variant called NUAN allows event driven systems to be modeled easier. It also allows relevant mental actions to be modeled. It is a small step towards more effective techniques for detailed design specifications.

- *Providing a better understanding of what usability is.* Chapter 2 of this thesis is entirely devoted to this issue. We have developed a framework of usability that gives structure to the concept. Using this framework we analyzed existing views on usability and showed how they “fit” in the framework. We believe we have “deepened” the understanding of usability by showing what is measurable, what can be changed and how it is all related to understanding the users tasks. This framework gives guidance when making design decisions and when evaluations are done.
- *Allowing re-use of successful solutions in design.* Our work on interaction patterns is the main contribution to address this issue. The patterns offer very concrete solutions that can be applied directly. Each pattern has been evaluated by several experts and contain a rationale that provides the justification for the solution. This way product quality not only depends on a good analysis but also on the use of explicit and proven design knowledge. Designers can select patterns based on the design context and apply them appropriately. We developed a set of 30 patterns to establish a start for a future pattern language for interaction design.

10.3 Future Research

Developing a design method almost never ends. There are still many areas that need to be improved both on the theoretical as the practical side. Theories are important to link together many aspects in the process of design. With the help of sound theories we can develop practical techniques that make improvements of the process and product possible. On the other hand, many techniques are also developed ad hoc when designers face problems. Studying them can also contribute much to theoretical understanding.

- *Change-effect relationships between means and usage indicators.* In chapter 2, the concept of usability is discussed and put into a structural framework. The usage indicators of the model are measurable but the really interesting thing is how we can use the means to influence the usage indicators. There is a complex relationship between means and usage indicators which we hardly understand at present. The appropriate use of means depends on contextual information that is part of the task model and design context. Design principles give us some rough ideas about how to use the means but a much deeper understanding is needed if we want to make real progress. If we are able to formulate metrics

that incorporate the contextual information, such metrics could allow usability evaluation even when only detail design specifications exist.

- *Bringing task analysis to the work practice.* In the area of task analysis and task modeling this thesis has made advances both on theory and on practical techniques. Nonetheless, those techniques probably need to be refined through many applications before they are established techniques that every designer uses. Both the task world ontology and the representation techniques need to be refined further through practical applications. Extensive application experiences can show missing aspects, tune techniques and improve insights. Development of the most important design knowledge, formulated in patterns, also needs to be grounded in practice. In addition, the methodological side needs to be covered as well so that practitioners understand what to do and how to do it.
- *Patterns and task-based design.* This thesis discusses interaction design patterns as a promising technique to incorporate explicit design knowledge into the design process. We feel that this is just the start of a promising research area. User interfaces are composed of many elements that are put into a specific structure. Patterns are a means to try to understand why some arrangements of elements are better than others and under which circumstances. This is exactly the kind of knowledge which gives designers a better understanding of their tools of the trade so they will get better at using them. In order to make user interface design more of an engineering discipline, it needs to excel in analysing the problem well and creating solutions using valuable design knowledge.

Besides using patterns for interaction design, patterns could also be used in the task analysis phase. Certain activities can be described more generally in patterns. Such "work" or "ethnographic" patterns could help seeing how the work is done. When redesigning work, patterns can be used to propose proven task structures. Going even further, task patterns could then be linked to interaction designs via an intermediate "categorized" task model. For example, specific tasks could be generalized as "comparison tasks" or "entry tasks" which then lead to the selection of patterns related to such tasks.

- *Integrated tools supporting several design activities.* Our tool EUTERPE proves that there is indeed a need for tool support in the early activities of task-based design. The tool is not finished and more development could lead to a tool of commercial quality. Tools are nowadays essential elements in most design processes. Any method for design could greatly benefit from tool support both for the process itself but also for acceptance in the industry.

The issues mentioned above are not only relevant from a task-based design perspective. Such issues have also been raised by people that are not directly working on task-based design. The need for better processes, a better understanding of quality and how to reach it is central in the field of Human Computer Interaction. The most important thing in task-based design is the central position of the users and their work. The research described in this thesis uses that viewpoint to create better tools and techniques. We

feel that we made a small step forwards but still a lot more progress is needed to fully achieve our goals.

Appendix A

NUAN symbol definitions

User Actions	Functions	Description
Move the cursor	POINTERTO(x) CURSORTO(x)	Move the mouse pointer to x. Move the cursor to x.
Clicking Pressing	CLICK(x) CLICK(x,y) PRESS(x) HOLD(x) RELEASE(x)	Press and depress x Press and depress x y times Press and release x Hold x Release x (a click is just a short function for press and release)
Enter a string Looking Talking Insert something	ENTER(x) LOOKAT(x) SAY(x) INSERT(x)	Enter something. Look at x. Say x to the system. Insert x in the system.
Highlighting Unhighlighting Flash Removing Moving Changing Showing Waiting Output Force Feedback Ejecting Warning Recalling Memorizing Forgetting Finding Choosing Comparing Compute Transform	HIGHLIGHT(x) UNHIGHLIGHT(x) FLASH(x) REMOVE(x) MOVETO(x,y) CHANGE(x,y) SHOW(x) WAIT(x) OUT(x,y) FORCEF(x,y) EJECT(x) WARNING(x,y) RECALL(x) MEMORIZE(x) FORGET(x) FIND(x) CHOOSE(x,y) COMPARE(x,y,locX,locY) COMPUTE(Formula) TRANSFORM(Input, Output)	x is highlighted x is not highlighted Highlight and immediately un-highlight x is removed. x is moved to y. x is removed and y is shown. x is shown to the users. Wait x seconds. Output x using y (audio, video etc) movement x is done by device y. x is ejected from the system Display a warning with options y Retrieve x from working memory Memorize x Remove x from working memory Find x on the display Choose x from the set y Compare x with y in their locations Compute Formula Transform Input into Output

Constant	Description	
button1	(Mouse)button 1	
any_key	Any key from a keyboard	
any_button	Any button from some device	
'a'	The key 'a'	
"abc"	The string abc	
@(x,y)	The location (x,y)	
[x]	Some object x, for example [text.file]	
<x>	The object x, for example <test.txt>	
x,y	A set containing the elements x and y	
left, right, up, down	The directions left, right, up and down	
audio	The audio device	
video	The video display terminal	

Operation	Operator	Description
Repetition	(X)*	X is performed zero or more times
	n(X)	X is performed n times
	#(X)	X is performed zero or one time.
Choice	X Y	X or Y is performed (but not both)
Order independence	X && Y	X and Y are performed in no particular order
Interruptability	X ? Y	X can be interrupted by Y
Sequence	X Y	X is followed by Y
Parallelism	X Y	X and Y are performed simultaneously
Interleavability	X \ Y	Y can be started before X is completed and vice versa
Condition	X : Y	If X is true, Y is performed
Start/stop	⋈ X	X is started
	> X	X is stopped
	(X)~	The inverse of X is performed
Addition	x + y	y is added to x
Substraction	x - y	y is subtracted from x
Events	!(X)	The interface action X isn't directly caused by a user action

Appendix B

Interaction Design Patterns

Wizard

Problem The user wants to achieve a single goal but several decisions need to be made before the goal can be achieved completely, which may not be known to the user.

Usability Principle User Guidance (Visibility)

Context A non-expert user needs to perform an infrequent complex task consisting of several subtasks where decisions need to be made in each one. The number of subtasks must be small e.g. typically between 3 and 10.

Forces

- The user is highly interested in reaching the overall goal but may not be familiar or interested in the steps that need to be performed.
- The task can be ordered but are not always independent of each other i.e. a certain task may need to be finished before the next task can be done.
- To reach the goal, several steps need to be taken but the exact steps required may vary because of decisions made in previous steps.

Solution Take the user through the entire task one step at the time. Let the user step through the tasks and show which steps exist and which have been completed.

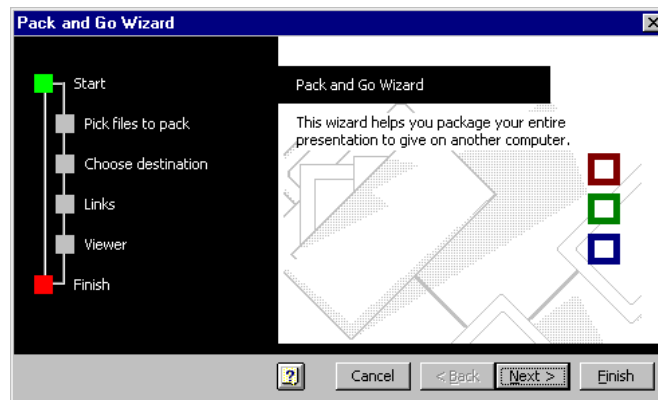
When the complex task is started, the user is informed about the goal that will be achieved and the fact that several decisions are needed. The user can go to the next task by using a navigation widget (for example a button). If the user cannot start the next task before completing the current one, feedback is provided indicating the user cannot proceed before completion (for example by disabling a navigation widget). The user should also be able to revise a decision by navigating back to a previous task.

The user is given feedback about the purpose of each task and the user can see at all times where (s)he is in the sequence and which steps are part of the sequence. When the complex task is completed, feedback is provided to show the user that the tasks have been completed and optionally results have been processed.

Users that know the default options can immediately use a shortcut that allows all the steps to be done in one action. At any point in the sequence it is possible to abort the task by choosing the visible exit.

Rationale The navigation buttons suggest the users that they are navigating a path with steps. Each task is presented in a consistent fashion enforcing the idea that several steps are taken. The task sequence informs the user at once which steps will need to be taken and where the user currently is. The learnability and memorability of the task are improved but it may have a negative effect of the performance time of the task. When users are forced to follow the order of tasks, users are less likely to miss important things and will hence make fewer errors.

Example This is the 'Pack 'n Go Wizard' from PowerPoint. The user wants to package a presentation so that the presentation can be given on another computer.



Several relevant decisions need to be taken and the wizard helps the user take these decisions. The green box shows the current position in the sequence of tasks.

Known Uses Microsoft PowerPoint Pack and Go wizard; Installshield installation programs

Related Patterns Consider NAVIGATING SPACES and LIST BROWSER to provide the navigation controls.

Hinting

Problem The user needs to know how to select functions.

Usability Principle Incremental Revealing (Visibility)

Context Applications where the functionality is accessible in more than one way, e.g. through menus, using keyboard shortcuts, or through toolbars. This pattern can be used to make the user aware of the other possibilities in a subtle and non-obtrusive way.

Forces

- The available screen space may be limited so there is no space for extra visual hints.
- The user needs some way of discovering and learning these alternatives and possibly more efficient ways, in a non-obtrusive way.
- The user may or may not already know the other ways to access the function.
- The number of ways to activate the function determines the number of possible hints.

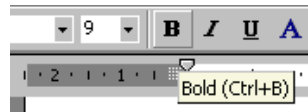
Solution Give the user hints for other ways to access the same function.

When accessing a function in one way, provide hints for other ways to access the same function. One possibility is to use multiple labels; one label for each way the function can be accessed. For example, if the function has a keyboard shortcut, show the key combination. If there is an icon shortcut for the function, show the icon. Always show the main label and show other labels directly if possible within the constraints.

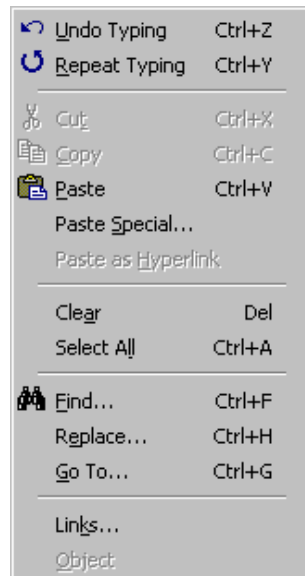
Other possibilities are to use helper agents or delayed messages that react on user actions. For example, a tool tip is displayed when the user holds the mouse over a widget for 2 seconds.

Rationale Chances are high that the user is familiar with at least one way to access a specific function. By showing other labels such as the key shortcut or an icon, the user will learn more associations for the function access. At some point the user may "see" the icon in a toolbar and use it instead of the menu. In the same way, the user may prefer keyboard access over mouse access. The solution increases learnability and memorability. When the user actually starts using other ways of selecting functions the performance speed may also increase.

Example These screenshots are taken from Word2000. They shown to possible instances of this pattern, one using tool tips and the other using menus with icons.



In the menu there is space to include the icon and shortcut but the toolbar icon does not allow this. In that case the information is displayed in a tool tip that pops up after a short delay. That way advanced users are not bothered with windows that pop up all the time.



Known Uses Tool tips, Office2000 menus.

Related Patterns Consider the COMMAND AREA pattern for the placement of the functionality.

Continuous Filter

Problem The user needs to find an item in an ordered set.

Usability Principle Immediate feedback (Feedback)

Context This pattern allows the user to dynamically narrow the search depending on the immediate feedback given by the continuous filter.

Forces

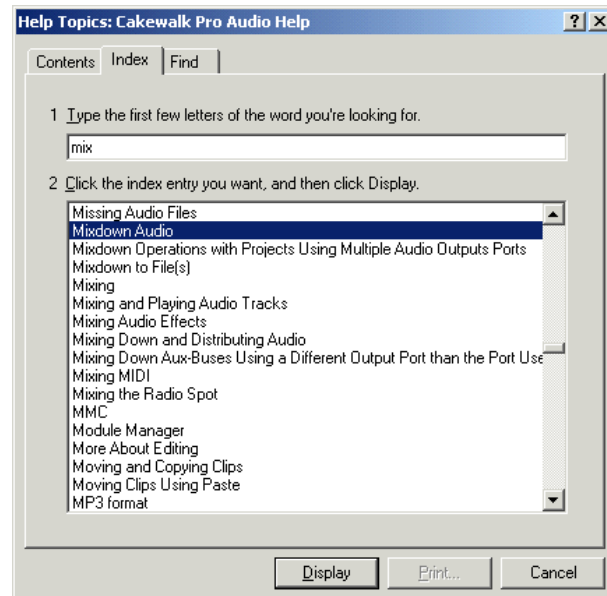
- The user is searching for an item in a large ordered set and may not be familiar with the exact item, nor is the user sure the item exists.
- The user searches for an item but the search term may lead to multiple results.

Solution Provide a filter component with which the user can in real time filter only the items in the data that are of his interest.

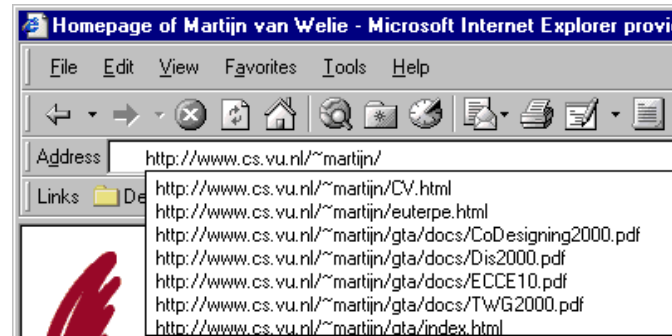
The filtered set is shown concurrently with the search term from the moment the user starts entering the search term. If relevant, the closest matches are highlighted while some previous and successive items might be shown as well.

Rationale Because the user gets immediate feedback on the search term, the user searches very efficient and may even see other relevant items. Because the filtered items are shown the user can adjust the search term in real time or even bypass completing the search term and go directly to the preferred item. The solution improves the performance time and satisfaction.

Example This screenshot taken from Cakewalk 9 uses the common help functionality. In the index function the user is guided towards the item while typing.



This screenshot shows the URL history of Internet Explorer 5. As you type in a URL it shows the list of possible URLs that match the unfinished URL.



Known Uses Help systems (Cakewalk, MS Word 2000, Visual Studio 6), Internet Explorer 5, IntelliSense.

Related Patterns Consider the FAVOURITES pattern if the users often repeat the same query.

Unambiguous Format

Problem The user needs to supply the application with data but may be unfamiliar with which data is required or what syntax to use.

Usability Principle User Guidance (Constraints)

Context Any system where structured data must be entered. Data such as dates, room numbers, social security numbers or serial numbers are usually structured. The exact syntax used for such data may vary per country or product.

Forces

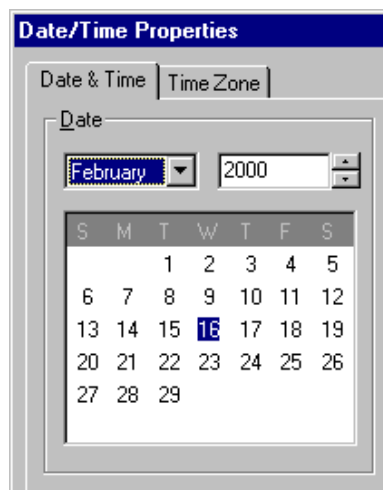
- When the data is entered using an unexpected syntax, the data cannot be used by the application.
- The user may be familiar with the data but may not know the exact required syntax.
- The user strives for entry speed but also wants it to be entered correctly.
- Cultural conventions determine what the user expects the syntax to be. For example, dd/mm/yy is usual in Europe while mm/dd/yy is used in the United States.

Solution Only allow the user to enter data in the correct syntax.

Present the user with fields for each data element of the structure. Label each field with the name of the data unit if there can be doubt about the semantics of the field. The field does not allow incorrect data to be entered. Avoid fields where users can type free text. Additionally, explain the syntax with an example or a description of the format. Provide sound defaults for required fields, fields that are not required should be avoided or otherwise marked as optional. When optional fields are used, the consequences for the user must be explained.

Rationale The main idea is avoid entering incorrect data by not making it possible to enter wrong data. By showing the required format the chances of errors are reduced because the user is given complete knowledge. However, because the user now has to give multiple data inputs instead of one, more time is needed to enter the data. The solution reduces the number of errors and increases satisfaction but the performance time may go down.

Example This snapshot is from the date and time control panel in MS Windows. Entering the date is spit up in three input areas. Each of the input fields allows only valid entries. Entering an invalid date becomes impossible.



Known Uses MS Windows Date/Time control panel, Windows 9x serial box

Related Patterns Use the GRID LAYOUT pattern to layout the data units.

Helping Hands

Problem Users need to enter many different types of objects.

Usability Principle Separation (Task Conformance)

Context Applications with modes and for expert users. Typical use is in drawing and modelling applications where one hand is used to control an input device for the actual drawing.

Forces

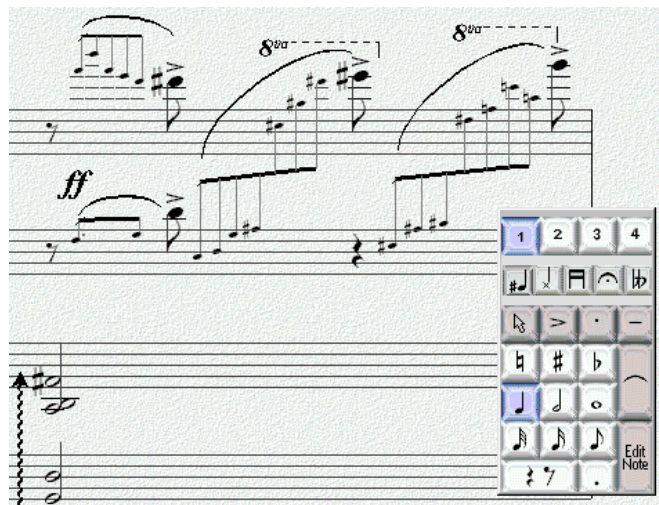
- Although the number of modes may be high, the user may only use a particular subset regularly.
- Most people can use two hands while they mostly use one.

Solution Use one hand to enter the data while the other hand is used to switch modes.

The data-entry hand can use any available input devices such as mouse, trackball, pen etc. The data-entry hand can keep the cursor at the working area while the other hand is used to switch modes using the keyboard or any other special purpose device. Allow mode switching for the most often used modes, in particular during data entry functions.

Rationale Using one hand to switch modes allows the other hand to keep the cursor on the working area. This eliminates many mouse movements otherwise needed for the mode switching. It speeds up the entry of data.

Example This example is taken from Sibelius, a music score editor. Just like in many drawing programs it contains many modes, for instance for entering notes. In Sibelius the keys of the numeric keypad are mapped to entry modes. This way, it is not necessary to use the mouse to switch modes and it is only used for entering notes. Using the other hand, users can rapidly switch modes.



Known Uses Sibelius, Blender (3D modelling tool).

Related Patterns To indicate the current mode, consider the MODE CURSOR pattern.

Grid Layout

Problem The user needs to quickly understand information and take action depending on that information.

Usability Principle Consistency, Predictability (Conceptual Models)

Context Any circumstance where several information objects are presented and arranged spatially on a limited area. Typically in the design of dialog screens, forms and web pages.

Forces

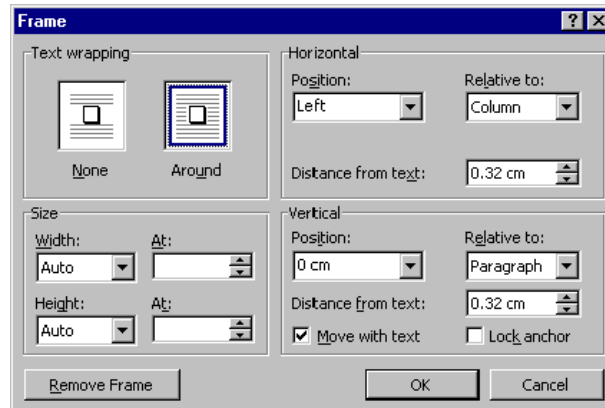
- The user needs to see many objects but wants to see them in a clear organized way.
- The user want to minimize the time it takes to scan/read/view objects on the screen.
- The objects are often related and can be grouped conceptually.
- The presentation needs to be compact, but still clear, pleasant and readable.

Solution Arrange all objects in a grid using the minimal number of rows and columns, making the cells as large as possible.

The objects are arranged in a matrix using the minimal number of rows and columns. Objects that are of the same type must be aligned and displayed in the same way. If several objects can be grouped, the grid applies on the level of groups as well. Short elements can be stretched, beginning and ending on grid boundaries. Long elements can span multiple grid cells. Certain objects may have a fixed size that increases the number of rows and columns in which case they should keep their fixed size. Standard response buttons may have predefined positions and can be regarded as being outside the grid.

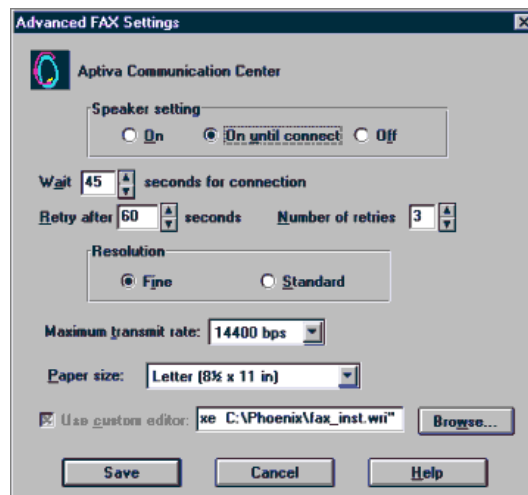
Rationale Minimising the number of rows and columns improves the time needed to scan the information and to take the appropriate action (Fitts Law). Additionally, it causes a very consistent layout with minimal visual clutter and is perceived to be non-obtrusive to the user. The time needed to read the information is reduced which can increase the task performance time. The resulting layout is pleasant to see and increases the satisfaction.

Example This screenshot is taken from Word 97. Several objects are placed in a dialog box. The elements have been arranged in a grid and objects have been aligned and sized evenly to reduce the number of rows and columns.



Known Uses Microsoft Word Frame Options, many other applications.

Counterexample This image is taken from IBM's Aptiva Communication Center, and demonstrates that the developers simply wanted to get the settings on the screen, rather than make it easy for people to adjust the settings. There is no flow to the screen; your eyes just jump around from place to place as your brain tries to elicit some sort of order.



Bibliography

- Abowd, G. D., Wan, H. M. & Monk, A. F. (1995), A formal technique for automated dialogue development, in 'Proceedings of Designing Interactive Systems, DIS '95', ACM Press, Ann Arbor MI.
- Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. & Angel, S. (1977), *A Pattern Language*, Oxford University Press, New York.
- Annett, J. & Duncan, K. (1967), 'Task analysis and training in design', *Occupational Psychology* **41**, 211–221.
- Apple Computer Inc. (1992), *Macintosh Human Interface Guidelines*, Addison-Wesley Publishing Company.
- Arnold, A. (1998), Action Facilitation and Interface Evaluation, PhD thesis, Technische Universiteit Delft.
- Balbo, S. (1994), Ema: Automatic analysis mechanism for the ergonomic evaluation of user interfaces, Technical Report 96/44, CSIRO.
- Barnard, P. (1987), Cognitive resources and the learning of human-computer dialogues, in 'Interfacing thought: Cognitive aspects of human-computer interaction', MIT Press.
- Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., Gross, B., Lehder, D., Marmolin, H., Moore, B., Potts, C., Skousen, G. & Thomas, J. (1998), 'Putting it all together: Towards a pattern language for interaction design', *SIGCHI Bulletin* **30**(1), 17–24.
- Beard, D., Smith, D. & Denelsbeck, K. (1996), QGOMS: A direct-manipulation tool for simple GOMS models, in 'Proceedings of CHI '96', ACM Press, pp. 25–26.
- Bevan, N. (1999), 'Quality in use: Meeting user needs for quality', *Journal of Systems and Software* **49**(1), 89,96.
- Beyer, H. & Holtzblatt, K. (1998), *Contextual Design*, Morgan Kaufmann Publishers.
- Bias, R. & Mayhew, D., eds (1994), *Cost-Justifying Usability*, Academic Press.

- Biere, M., Bomsdorf, B. & Szwillus, G. (1999), The visual task model builder, in 'Proceedings of CADUI '99', Louvain-la-Neuve, Belgium.
- Bomsdorf, B. & Szwillus, G. (1999a), Tool support for task-based user interface design, in 'Proceedings of CHI 99, Extended Abstracts', Pittsburgh PA, United States, pp. 169–170.
- Bomsdorf, B. & Szwillus, G. (1999b), 'Tool support for task-based user interface design - a CHI99 workshop', *ACM SGCHI Bulletin* **31**(4), 27–30.
- Britton, C. & Jones, S. (1999), 'The untrained eye: How languages for software specification support understanding in untrained users', *Human-Computer Interaction* **14**, 191–244.
- Brown, W., Malveau, R., McCormick, H. & Mowbray, T. (1998), *Anti Patterns, Refactoring Software, Architectures and Projects in Crisis*, John Wiley, New York.
- Butterworth, R., Blandford, A. & Duke, D. (1998), The role of formal proof in modeling interactive behavior, in '5th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS98', Abingdon, UK.
- Card, S., Mackinlay, J. & Shneiderman, B. (1999), *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann Publishers.
- Card, S., Moran, T. & Newell, A. (1983), *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates.
- Carroll, J. M. (1995), *Scenario-Based Design*, John Wiley & Sons Inc., Toronto.
- Cooper, A. (1995), *About Face, the essentials of user interface design*, IDG Books Worldwide.
- Cooper, A. (1999), *The Inmates are Running the Asylum*, SAMS publishing.
- Coutaz, J., Faconti, G., Paterno, F., Nigay, L. & Salber, D. (1993), MATIS: a UAN Description and Lessons Learned, Technical Report SM/WP14, The AMODEUS Project.
- Dayton, T., McFarland, A. & Kramer, J. (1998), Bridging user needs to object oriented gui prototype via task object design, in L. Wood, ed., 'User Interface Design: Bridging the Gap from User Requirements to Design', CRC Press.
- de Haan, G. (2000), ETAG, A formal model of competence knowledge for user interface design, PhD thesis, Vrije Universiteit Amsterdam.
- de Haan, G., van der Veer, G. & van Vliet, J. (1991), 'Formal modelling techniques in human-computer interaction', *Acta Psychologica* **78**, 27–67.
- Dilli, I. & Hoffmann, H. J. (1994), DIADES-II, a multi-agent user interface design approach with an integrated assesment component, in 'CHI'94 HCI Bibliography, SIG on Tools for Working with Guidelines', ACM Press.

- Dix, A., Abowd, G., Beale, R. & Finlay, J. (1998), *Human-Computer Interaction*, 2nd edn, Prentice Hall Europe.
- Earthy, J. (1999), Usability maturity model: Attitude scale, Technical Report D5.1.4s, INUSE Project.
- Farenc, C., Palanque, P. & Vanderdonckt, J. (1995), User interface evaluation: is it still usable ?, in 'Proceedings of 6th International Conference on Human-Computer Interaction HCI International'95', Elsevier Science Publishers, Yokohama.
- Fitts, P. (1954), 'The information capacity of the human motor system in controlling the amplitude of movement', *Journal of Motor Behavior* **47**, 381–391.
- Gamboa Rodriguez, F. & Scapin, D. (1997), Editing MAD* task description for specifying user interfaces, at both semantic and presentation levels, in '4th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS97', Granada, Spain.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass.
- Gorny, P. (1995), Expose, hci-counseling for user interface design, in 'Proceedings of INTERACT '95', Chapman & Hall, Lillehammer, Norway.
- Graesser, A., Robertson, S. & Anderson, P. (1981), 'Incorporating inferences in narrative representations: A study of how and why', *Cognitive Psychology* **13**, 343–370.
- Hall, T. & Fenton, N. (1997), 'Implementing Effective Software Metrics Programs', *IEEE Software* **14**(2), 55–65.
- Hartson, H. (1998), 'Human-computer interaction: Interdisciplinary roots and trends', *Journal of Systems and Software* **43**(2), 103–118.
- Hassenzahl, M., Platz, A., Burmester, M. & Lehner, K. (2000), Hedonic and ergonomic quality aspects determine a software's appeal, in 'Proceedings of CHI 2000, Extended Abstracts', Den Haag, The Netherlands, pp. 201–208.
- Herczeg, M. (1999), A Task Analysis Framework for Management Systems and Decision Support Systems, in 'Proceedings of the AoM/IAoM 17th International Conference on Computer Science', San Diego, California, pp. 29–34.
- Hix, D. & Hartson, H. (1994), Ideal: An environment to support usability engineering, in 'Proceedings of the 1994 East West International Conference on HCI', St. Petersburg, Russia.
- Hix, D. & Hartson, H. (1998), *Developing User Interfaces: Ensuring Usability Through Product & Process*, John Wiley & Sons Inc.

- ISO (1988), *ISO/IS 8807 Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on Temporal Ordering of Observational Behaviour*.
- ISO (1991a), *ISO 9126 Software product evaluation - Quality characteristics and guidelines for their use*.
- ISO (1991b), *ISO 9214-10 Ergonomic Requirements for office Work with VDT's - Dialogue Principles*.
- ISO (1991c), *ISO 9214-11 Ergonomic Requirements for office Work with VDT's - Guidance on Usability*.
- John, B. E. & Kieras, D. E. (1996), 'The GOMS family of user interface analysis techniques: Comparison and contrast', *ACM Transactions on Computer-Human Interaction* (3), 320–351.
- Johnson, P., Diaper, D. & Long, J. (1984), Tasks, skills and knowledge: Task Analysis for Knowledge-based Descriptions, in B. Shackel, ed., 'Proceedings of INTERACT 84', North Holland, pp. 499–503.
- Johnson, P., Johnson, H., Waddington, R. & Shouls, A. (1988), 'Task-Related Knowledge Structures: Analysis, Modeling and Application', *People and Computers IV* pp. 35–62.
- Johnson, P., Wilson, S., Markopoulos, P. & Pycock, J. (1983), Adept - advanced design environment for prototyping with task models, in 'Proceedings of INTERACT 83', Amsterdam, The Netherlands, pp. 56–64.
- Jordan, B. (1996), Ethnographic Workplace Studies and CSCW, in D. Shapiro, M. Tauber & R. Traunmüller, eds, 'The Design of Computer Supported Cooperative Work and Groupware Systems', Elsevier North Holland, Amsterdam, pp. 17–42.
- Kahn, P. & Krysztof, L. (1998), 'Principles of Typography for User Interface Design', *Interactions* 5(6), 15–29.
- Karat, C. (1994), A business case approach to usability cost justification, in R. Bias & D. Mayhew, eds, 'Cost-Justifying Usability', Academic Press.
- Kieras, D. (1994), A Guide to GOMS Task Analysis, Technical Report Unpublished, University of Michigan.
- Kieras, D. (1996), Guide to GOMS model usability evaluation using NGOMSL, in M. Helander & T. Landauer, eds, 'The Handbook of Human-Computer Interaction', ?, Amsterdam.
- Kieras, D. & Polson, P. (1985), 'An approach to the formal analysis of user complexity', *International Journal of Man-Machine Studies* 22(4), 365–394.
- Kirwan, B. & Ainsworth, L. (1992), *A Guide to Task Analysis*, Taylor and Francis.

- Lecerof, A. & Paternò, F. (1998), 'Automatic support for usability evaluation', *IEEE Transactions on Software Engineering* **24**(10), 863–888.
- Lim, K. & Long, J. (1994), *The Muse Method for Usability Engineering*, Cambridge University Press.
- Löwgren, J. (1995), Perspectives on Usability, Technical Report LiTH-IDA-R-95-23, ISSN-0281-4250, Department of Computer and Information Science, Linköping University, Sweden.
- Macinlay, J. (1986), 'Automating the Design of Graphical Presentations of Relational Information', *ACM Transactions on Graphics* **5**(2), 110–141.
- MacLean, A., Young, R. & Bellotti, V. (1991), 'Questions, options, and criteria: Elements of design space analysis', *Human-Computer Interaction* **6**, 201–250.
- Mahajan, R. & Shneiderman, B. (1995), A Family of User Interface Consistency Checking Tools, Technical Report ISR-TR-95-52, Institute for Systems Research, University of Maryland, USA.
- Mahemoff, M. & Johnston, L. (1998a), Pattern languages for usability: An investigation of alternative approaches, in 'Proceedings of the Asia-Pacific Conference on Human Computer Interaction APCHI'98', IEEE Computer Society, Shonan Village, Japan, pp. 25–31.
- Mahemoff, M. & Johnston, L. (1998b), Principles for a usability-oriented pattern language, in 'Proceedings of the Australian Computer Human Interaction Conference OZCHI'98', IEEE Computer Society, Adelaide, Australia, pp. 132–139.
- Martin, J. (1991), *Rapid Application Development*, MacMillan.
- May, J. & Barnard, P. (1995), Cinematography and interface design, in K. Nordby, P. Helmersen, D. Gilmore & S. Arnesen, eds, 'Proceedings of Interact '95', pp. 26–31.
- Mayhew, D. (1992), *Principles and Guidelines in Software User Interface Design*, Prentice Hall PTR, New Jersey.
- Mayhew, D. (1999), *The Usability Engineering Lifecycle: a practitioner's handbook for user interface design*, Morgan Kaufmann Publishers.
- Mayhew, D. & Mantei, M. (1994), A basic framework for cost-justifying usability, in R. Bias & D. Mayhew, eds, 'Cost-Justifying Usability', Academic Press.
- McKay, E. (1999), *Developing User Interfaces for Microsoft Windows*, Microsoft Press.
- Microsoft (1995), *The Windows Interface Guidelines for Software Design*, Microsoft Press.

- Mullet, K. & Sano, D. (1995), *Designing Visual Interfaces: Communication Oriented Techniques*, SunSoft Press, Prentice Hall.
- Myers, B. (1995), 'User Interface Software Tools', *ACM Transactions on Computer-Human Interaction* 2(1), 64–103.
- Nardi, B. (1996), *Context and consciousness: activity theory and human-computer interaction*, MIT Press.
- Neches, R., Foley, J., Szekely, P., Sukaviriya, P., Luo, P., Kovacevic, S. & Hudson, S. (1993), Knowledgeable development environments using shared design models, in 'Proceedings of the 1993 International Workshop on Intelligent User Interfaces', pp. 63–70.
- Nielsen, J. (1993), *Usability Engineering*, Academic Press.
- Noldus Software (1999), *The Observer*.
URL: <http://www.noldus.com>
- Norman, D. (1988), *The Design of Everyday Things*, Basic Books.
- Norman, D. (1999), *The Invisible Computer*, Basic Books.
- Palanque, P. & Paternò, F. (1997), *Formal Methods in Human Computer Interaction*, Springer Verlag.
- Parnassus Software (1999), *Scriptwerx*.
URL: <http://www.scriptwerx.com/online.html>
- Paternò, F. (1999), *Model-Based Design and Evaluation of Interactive Applications*, Springer Verlag.
- Paternò, F., Mancini, C. & Meniconi, S. (1997), ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, in S. Howard, J. Hammond & G. Lindgaard, eds, 'Proceedings of INTERACT '97', Chapman & Hall, Sydney, pp. 362–369.
- Payne, S. & Green, T. (1989), Task-Action Grammar: The Model and its Developments, in D. Diaper, ed., 'Task Analysis for Human-Computer Interaction', Ellis Horwood, Cambridge MA.
- Perzel, K. & Kane, D. (1999), Usability Patterns for Applications on the World Wide Web, in 'Proceedings of the Pattern Languages of Programming PLoP'99'.
- Petre, M., Blackwell, A. & Green, T. (1997), Cognitive Questions in Software Visualisation, in 'Software Visualization: Programming as a Multi-Media Experience', MIT Press.
- Pfaff, G. & ten Hagen, P. (1985), *Seeheim Workshop on User Interface Management Systems*, Springer Verlag, Berlin.

- Polson, P., Lewis, C., Rieman, J. & Wharton, C. (1992), 'Cognitive walkthroughs: a method for theory-based evaluation of user interfaces', *International Journal of Man-Machine Studies* **36**, 741–773.
- PowerProductions Software (1999), *Storyboard Quick*.
URL: <http://www.powerproduction.com/quick.html>
- Preece, J., Benyon, D., Davies, G., Keller, L. & Rogers, Y. (1990), *A Guide to Usability*, The Open University.
- Puerta, A. (1997), 'A Model-based Interface Development Environment', *IEEE Software* **14**(4).
- Puerta, A. & Eisenstein, J. (1999), Towards a general computational framework for model-based interface development systems, in 'Proceedings of International Conference on Intelligent User Interfaces (IUI99)', Los Angeles, United States, pp. 171–178.
- Rauterberg, M. (1995), Four different measures to quantify three usability attributes: Feedback, interface directness and flexibility, in '2th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS95', Toulouse, France.
- Reisner, P. (1983), Analytic tools for human factors of software, in A. Blaser & M. Zoepfritz, eds, 'Enduser Systems and Their Human Factors', Lecture Notes in Computer Science, Springer Verlag, Berlin, pp. 94–121.
- Rijken, D. (1994), 'The timeless way... the design of meaning', *SIGCHI Bulletin* **6**(3), 70–79.
- Rosson, M. & Carroll, J. (1995), Integrating task and software development for object-oriented applications papers, in 'Proceedings of CHI '95', ACM Press, pp. 377–384.
- Rumbaugh, J., Jacobson, I. & Booch, G. (1997), *Unified Modeling Language Reference Manual*, Addison Wesley.
- Sage, M. & Johnson, C. (1998), Pragmatic formal design: A case study in integrating formal methods into the hci development cycle, in '5th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS98', Abingdon, UK, pp. 134–154.
- Scapin, D. & Bastien, J. (1997), 'Ergonomic Criteria for Evaluating the Ergonomic Quality of Interactive Systems', *Behaviour & Information Technology* **16**(4/5), 220–231.
- Scapin, D. & Pierret-Golbreich, C. (1989), 'Towards a method for Task Description: MAD', *Work With Display Units* (89), 371–380.

- Sears, A. (1995), AIDE: A step toward metric-based interface development tools, in 'Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'95)', ACM Press, New York, pp. 101–110.
- Sebillotte, S. (1988), 'Hierarchical planning as method for task analysis: The example of office task analysis.', *Behaviour and Information Technology* 7(3), 275–293.
- Sebillotte, S. (1995), 'Methodology guide to task analysis with the goal of extracting relevant characteristics for human-computer interfaces', *International Journal of Human-Computer Interaction* 7(4), 341–363.
- Shackel, D. (1991), Usability - Context, Framework, Definition, Design and Evaluation, in D. Shackel & S. Richardson, eds, 'Human Factors for Informatics Usability', Cambridge University Press.
- Sharratt, B. (1990), Mental-Cognition-Action Tables: A Pragmatic Approach to Analytical Modelling, in 'Proceedings of INTERACT '90', Elsevier Science Publishers, Amsterdam, The Netherlands, pp. 271–275.
- Shneiderman, B. (1998), *Designing the User Interface: strategies for effective Human-Computer Interaction*, 3rd edn, Addison-Wesley Publishing Company.
- Smith, S. & Mosier, J. (1986), Guidelines for Designing User Interface Software, Technical Report MTR-010090, EDS-TR-86-278, The Mitre Corporation.
- Stry, C. (1999), *Task Analysis Design End User Systems*.
URL: <http://www.ce.uni-linz.ac.at/research/TADEUS/TADEUS.html>
- Szwilius, G. (1997), Object-oriented dialogue specification with odns, in 'Proceedings of Software Ergonomie '97', Dresden, Germany.
- Tam, R., Maulsby, D. & Puerta, A. (1998), U-tel: A tool for eliciting user task models from domain experts, in 'Proceedings of International Conference on Intelligent User Interfaces, IUI '98', San Francisco.
- Tauber, M. (1988), On Mental Models and the User Interface, in G. C. van der Veer, T. R. G. Green, J.-M. Hoc & D. Murray, eds, 'Working With Computers: Theory Versus Outcome', Academic Press, London.
- Tauber, M. (1990), ETAG: Extended Task Action Grammar - A Language for the Description of the User's Task Language, in D. Diaper, D. Gilmore, G. Cockton & B. Shackel, eds, 'Proceedings of INTERACT '90', Elsevier, Amsterdam.
- Tidwell, J. (1998), Interaction Design Patterns, in 'Proceedings of the Pattern Languages of Programming PLoP'98'.
- Tufte, E. (1983), *The Visual Display of Quantitative Information*, Connecticut, Graphics Press.
- Tufte, E. (1990), *Envisioning Information*, Connecticut, Graphics Press.

- Tullis, T. (1988), Screen Design, in M. Helander, ed., 'The Handbook of Human-Computer Interaction', Elsevier Science Publishers, Amsterdam.
- van der Veer, G. (1989), Users' representations of system-variety as function of user interface, culture, and individual style, in 'Man-Computer Interaction Research (MACINTER-II)', Human Factors in Information Technology 2, Elsevier Science Publishers.
- van der Veer, G. (1990), Human-Computer Interaction - Learning, Individual Differences, and Design Recommendations, PhD thesis, Vrije Universiteit Amsterdam.
- van der Veer, G. C., van Vliet, J. C. & Lenting, B. F. (1995), Designing complex systems - a structured activity, in 'Proceedings of Designing Interactive Systems '95', ACM Press, New York, Michigan.
- van der Veer, G., Hoeve, M. & Lenting, B. (1996), Modeling complex work systems - method meets reality, in 'Eighth European Conference on Cognitive Ergonomics', Granada, Spain, pp. 115-120.
- van der Veer, G., Lenting, B. & Bergevoet, B. (1996), 'GTA: Groupware Task Analysis - Modeling Complexity', *Acta Psychologica* **91**(3), 297-322.
- van der Veer, G. & van Welie, M. (2000), Task Based Groupware Design: putting theory into practice, in 'Proceedings of DIS 2000', New York, United States, pp. 326-337.
- van der Veer, G., van Welie, M. & Thorborg, D. (1997), Modeling Complex Processes in GTA, in S. Bagnara, E. Hollnagel, M. Mariani & L. Norros, eds, 'Sixth European Conference on Cognitive Science Approaches to Process Control (CSAPC)', CNR Rome, Italy, pp. 87-91.
- van Loo, R., van der Veer, G. & van Welie, M. (1999), Groupware task analysis in practice: a scientific approach meets security problems, in 'Seventh European Conference on Cognitive Science Approaches to Process Control (CSAPC)', Villeneuve d'Ascq, France, pp. 105-110.
- van Welie, M. & Trættemberg, H. (2000), Interaction Patterns in User Interfaces, in '7th Pattern Languages of Programs Conference (PLoP 2000)', Allerton Park, Monticello, Illinois, USA.
- van Welie, M. & van der Veer, G. (2000), Structured Methods and Creativity - a Happy Dutch Marriage, in 'Proceedings of Co-Designing 2000', Coventry, England, pp. 111-118.
- van Welie, M., van der Veer, G. & Eliëns, A. (1998a), An Ontology for Task World Models, in '5th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS98', Abingdon, UK, pp. 57-70.
- van Welie, M., van der Veer, G. & Eliëns, A. (1998b), Euterpe - Tool support for analyzing cooperative environments, in 'Ninth European Conference on Cognitive Ergonomics', Limerick, Ireland, pp. 25-30.

- van Welie, M., van der Veer, G. & Eliëns, A. (1999a), Breaking down Usability, in M. Sasse & C. Johnson, eds, 'Proceedings of INTERACT 99', Edinburgh, Scotland, pp. 613–620.
- van Welie, M., van der Veer, G. & Eliëns, A. (1999b), Usability Properties for Dialog Models, in '6th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS98', Braga, Portugal, pp. 238–253.
- van Welie, M., van der Veer, G. & Eliëns, A. (2000), Patterns as Tools for User Interface Design, in 'International Workshop on Tools for Working with Guidelines', Biarritz, France, pp. 313–324.
- van Welie, M., van der Veer, G. & Koster, A. (2000), Integrated representations for task modeling, in 'Tenth European Conference on Cognitive Ergonomics', Linköping, Sweden, pp. 129–138.
- Vanderdonckt, J. (1999), 'Development Milestones toward a Tool for Working with Guidelines', *Interacting with Computers* **12**(2), 81–118.
- Visio Software (1999), *Visio*.
URL: <http://www.visio.com>
- Wandmacher, J. (1997), 'Ein werkzeug für GOMS-analysen zur simulation und bewertung von prototypen beim entwurf', *Tagungsband PB97: Prototypen fr Benutzungsschnittstellen* **19**, 35–42.
- Whiteside, J., Bennett, J. & Holtzblatt, K. (1988), Usability Engineering: Our Experience and Evolution, in M. Helander, ed., 'The Handbook of Human-Computer Interaction', Elsevier Science Publishers, Amsterdam.
- Wood, L., ed. (1997), *User Interface Design: Bridging the Gap from User Requirements to Design*, CRC Press.
- Young, R. M., Green, T. R. G. & Simon, T. (1989), Programmable user models for predictive evaluation of interface designs, in 'Proceedings of CHI '89: Human Factors in Computings Systems', ACM Press.

Samenvatting

Dit proefschrift getiteld “Task-based User Interface Design” ofwel “Taakgebaseerd Ontwerpen van Gebruikersinterfaces” behandelt een ontwerpmethodologie voor user interfaces. Deze methodologie is taakgebaseerd wat wil zeggen dat de mensen en hun werk centraal staan in het ontwerpproces. De assumptie is dat wanneer een systeem ontworpen wordt om uitdrukkelijk de gebruikers te ondersteunen in hun taken, het systeem bruikbaar zal zijn.

Het begrip “bruikbaarheid”, usability in het Engels, staat centraal binnen het veld van user interface ontwerp. Bruikbaarheid is het voornaamste kwaliteitsbegrip en vormt het voornaamste doel tijdens het ontwerpen. Alhoewel het een intuïtief begrip is bestaan er verscheidene uiteenlopende definities en gerelateerde begrippen zoals heuristische principes en ergonomische criteria. Om meer duidelijkheid te scheppen hebben we een nieuwe raamwerk ontwikkeld waaraan de verscheidene begrippen en definities gerelateerd kunnen worden. Daarnaast laat het zien welke aspecten meetbaar zijn en hoe ontwerpbeslissingen bruikbaarheid beïnvloeden. De gebruikers en hun taken vormen een belangrijk deel van dat raamwerk.

Een complete methodologie beschrijft de activiteiten, de gebruikte technieken en de tools die het werken met die technieken ondersteunen. In dit proefschrift beschrijven we al deze aspecten in detail. De eerste activiteit is het bestuderen van de gebruikers en hun werk, ook wel “taakanalyse” genoemd. Om de taakanalyse goed te laten verlopen is het belangrijk dat het duidelijk is welke aspecten van het werk beschreven moeten worden. Daartoe is een ontologie ontwikkeld die aangeeft welke aspecten we zouden moeten “zien” als er een taakanalyse wordt gedaan. De ontologie is een soort bril waarmee naar de gebruikers en hun taken zou moeten worden gekeken.

Wanneer een taakanalyse uitgevoerd wordt, moeten de resultaten gedocumenteerd worden, zowel voor de analisten zelf als voor anderen die later de resultaten moeten gebruiken. Hiervoor gebruiken we representaties. Aangezien de informatie erg complex is en dus niet in één representatie beschreven kan worden, hebben we een aantal complementaire representaties ontwikkeld. Tezamen dekken zij alle belangrijke aspecten van de ontologie. Een ander gerelateerd aspect is dat de informatie ook geanalyseerd moet worden. Op basis van de ontologie zijn daartoe een aantal primitieven ontwikkeld die helpen om een eerste analyse te doen.

Als de gebruikers en hun taken in kaart zijn gebracht en er een analyse heeft plaatsgevonden, wordt er doorgaans een nieuw systeem ontwikkeld. De eerste stap is het

herontwerpen van de taken van de gebruikers gevolgd door het detail ontwerp van het nieuwe systeem. Het doel is om de resultaten van de taakanalyse effectief te gebruiken zodat een bruikbaar nieuw systeem ontstaat. We beschrijven een aantal technieken en representaties die gebruikt kunnen worden om het nieuwe systeem, de user interface, te specificeren. Een van die technieken genaamd NUAN is een door ons ontwikkelde extensie van een bestaande methode. Daarnaast is besproken op welke momenten er evaluatie plaats kan vinden en welk soort bruikbaarheidsproblemen daarmee gedetecteerd kunnen worden.

Een moeilijk punt in de detail-ontwerpfase is hoe ontwerpers zo snel mogelijk een goed ontwerp kunnen maken. We hebben een nieuwe techniek voorgesteld waarmee oplossingen waarvan men weet dat ze “werken” vastgelegd kunnen worden. Met behulp van zogenaamde “patronen” beschrijven we relaties tussen gebruikersproblemen in een specifieke context en goede oplossingen. Ontwerpers kunnen een verzameling patronen gebruiken zowel tijdens de ontwerpactiviteiten als tijdens de evaluatieactiviteiten. Patronen worden in andere disciplines reeds veelvuldig gebruikt en we hebben het concept ingezet voor het ontwerpen van user interfaces. Een verzameling van zo'n 30 patronen vormt het resultaat. Patronen vormen een stuk expliciete ontwerp-kennis die naast een goede analyse bijdragen aan de uiteindelijke kwaliteit, bruikbaarheid, van het systeem.

Tools zijn tegenwoordig niet meer weg te denken in de software ontwikkeling. Ook voor taakgebaseerd ontwerpen zijn er een aantal tools die gebruikt kunnen worden. We geven in dit proefschrift een overzicht van de bestaande tools en geven op basis daarvan enkele richtlijnen waaraan tools moeten voldoen. Vervolgens beschrijven we EUTERPE, een tool die we ontwikkeld hebben om onze representaties en specificatie technieken te ondersteunen. EUTERPE is gebaseerd op de ontologie en ondersteunt ontwerpers in het maken en wijzigen van enkele representaties. Daarnaast ondersteunt het een aantal analyse primitieven. De tool is over de jaren gebruikt in de praktijk en in het onderwijs. Verscheidene universiteiten in Nederland en in het buitenland gebruiken het.

Als laatste schetsen we onze ervaringen met onze methode. Een aantal malen per jaar wordt de methode gebruikt in ontwerpprojecten. De resultaten zijn positief, alhoewel de methode op enkele punten nog zeker verbeterd moet worden.

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
Copromotor: 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)
Promotiedatum: 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
Copromotor: dr. P.D.V. van der Stok
Promotie: 29 mei 2000
- 2000-3 Carolien M.T. Metselaar (UVA)
*Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief*
Promotor: Prof. dr. B.J. Wielinga
Co-promotor: Dr. P.A.A. van den Besselaar
Promotie: 20 juni 2000
- 2000-4 Geert de Haan (VU)
*ETAG, A Formal Model of Competence Knowledge for
User Interface Design*
Promotor: Prof. dr. J.C. van Vliet
Co-promotores: Dr. G.C. van der Veer
Dr. M.J. Tauber
Promotie: 10 oktober 2000
- 2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval
Promotores: Prof.dr. H.J. van den Herik (UM/RUL)
Prof.dr.ir. J.L.G. Dietz (TUD)
Prof.dr.ir. A. Hasman (UM)
Promotie: 14 september 2000
- 2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
Promotor: prof. dr. John-Jules Ch. Meyer
Co-promotoren: Dr. Frank S. de Boer
Dr. Wiebe van der Hoek
Promotie: 18 oktober 2000
- 2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
Promotor: prof.dr. J.-J. Ch. Meyer
Co-promotor: Dr.P.J.F.Lucas
Promotie: 30 oktober 2000
- 2000-8 Veerle Coupé (EUR)
Sensitivity Analysis of Decision-Theoretic Networks
Promotores: prof.dr.J.D.F. Habbema
prof.dr.ir.L.C van der Gaag
Promotie: 27 september 2000

- 2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
Promotor: prof. dr. M.L. Kersten (CWI/UvA)
Promotie: 03 november 2000
- 2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture
Promotor: prof. dr. M.L. Kersten (CWI/UvA)
Promotie: 14 december 2000
- 2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management
Promotor: prof. dr. M.L. Kersten (CWI/UvA)
Promotie: 14 december 2000
- 2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
Promotores: prof.dr. J.-J.Ch. Meyer (UU)
prof.dr.ir. L.C. van der Gaag (UU)
Co-promotor: dr. C.L.M. Witteman (UU)
Promotie: 12 maart 2001
- 2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
Promotor: prof. dr. J.-J.Ch. Meyer (UU)
Co-Promotoren: dr. W. van der Hoek (UU)
dr. F.S. de Boer (UU)
Promotie: 5 februari 2001
- 2001-3 Maarten van Someren (UvA)
Learning as problem solving
Promotor: prof. dr. B.J. Wielinga (UvA)
Promotie: 1 maart 2001
- 2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
Promotor: Prof. dr. H.J. van den Herik (UM/RUL)
Promotie: 22 februari 2001
- 2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
Promotor: prof.dr. J.C. van Vliet (VU)
Promotie: 10 april 2001

Index

- Adoptation, 149, 156
- Affordance, 100
- Agents, 36

- Basic task, 36

- CCT, 78
- Cinematography, 77
- Claims, 87
- Cognitive abilities, 11
- Cognitive walkthrough, 87
- Communication, 158
- Conceptual framework, 124
- Conceptual models, 101
- ConcurTaskTrees, 50, 81
- Constructors, 34, 37
- Context of use, 13, 86
- Contextual Design, 53
- Contextual Modeling, 53, 56
- Constraints, 100, 133
- CTA, 31
- Culture, 158

- Data collection, 113
- Design knowledge, 71, 93, 95
- Design rationale, 125
- Design space analysis, 137
- Dutch, 2
 - process, 2

- Effectiveness, 14, 48
- Efficiency, 14
- Elicitation, 116, 117
- Ergonomic criteria, 16
- ETAG, 82
- Evaluation techniques, 71
- Events, 58

- Feedback, 85, 101
 - haptic, 77
 - tactile, 77
- Fitts' law, 11, 95
- Flexibility, 101

- Gap, 72
 - bridging the gap, 72
- Generic interactions, 86
- Goals, 59, 131
- GOMS, 30, 31, 50, 78, 120
- GTA, 35
- Guidelines, 13, 16, 87
 - problems, 94
 - purpose, 94
 - rationale, 94
 - tools for, 93

- Hermeneutics, 37
- Heuristic evaluation, 64, 87
- Heuristics, 16, 19, 133
- Hierarchical models, 155
- Hierarchical representations, 130
- Hot spots, 136, 150
- HTA, 30, 31, 50, 114
- HTML, 135

- Incremental development, 71
- Inferential analysis, 114
- Interaction
 - understanding, 13
- Interaction design, 6, 95, 98
- Interface builders, 79
- ISO Standards
 - ISO 9126, 20
 - ISO 9241-10, 16
 - ISO 9241-11, 14

- Iterations, 60
- Java, 140
- Knowledge elicitation, 111
- LOTOS, 50, 57
- MAD, 34
- Management, 159
- Mental actions, 76, 85
- Mental models, 12
- Mental workload, 32, 78
- Meta predicates, 145
- Method acceptance, 152
- Methodology, 154
- Metrics, 88
- Model-based Interface Development Environments, 111
- Models
 - constraints, 112
 - integration, 111
 - purposes, 111
- MPEG1, 136
- Multimedia, 128
- Natural mapping, 100
- Observational data, 113
- ODSN, 121, 138
- Ontology, 37, 42, 84
- Operators, 59
- PANDA, 72
- Pattern language, 95, 105, 107
- Patterns, 14, 87
 - Alexander, 93, 97, 98
 - anti-patterns, 97
 - definitions, 97
 - format, 102
 - Gang of Four, 98, 106
 - rationale, 99
 - selecting, 106
 - task patterns, 98
 - types, 97
 - understanding, 106
 - user perspective, 99
- Performance estimation, 32
- Principles, 16
- Prolog, 129, 132, 140, 142
- Prototypes
 - mockups, 80
 - paper, 80
- Prototyping, 156
- PUMS, 91
- QOC, 125, 137
- Quality in use, 21
- Quality metrics, 21
- Quantitative analysis, 88
- Representations, 124
 - complexity, 49
 - purposes, 48, 49
 - static vs. dynamic, 62
 - understandability, 49
- Reuse, 157, 166
- Safety, 101
- Satisfaction, 14
 - fun, 14
- Scenarios, 53, 87
 - tool support, 118
- Seeheim model, 74
- Simulation tools
 - TAMOSIA, 121
 - VTMB, 121
 - WinCrew, 113
- Sketching, 79, 111
- Software quality, 20
 - external attributes, 20
 - internal attributes, 20
- State Transition Diagrams, 82
- Storyboard, 115
- Swim lanes, 53, 59
- TAG, 78
- TAKD, 30
- Task allocation, 12
- Task analysis
 - cooperative task analysis, 124
 - history of, 30
 - phases, 110

-
- Task conditions, 34
 - Task flow, 39
 - Task model evaluation, 132
 - Task models
 - constraints on, 65
 - detecting problems, 63
 - envisioned, 47, 63
 - existing, 47
 - prescriptive, 82
 - validation, 65, 68
 - Task-based design
 - limitations, 159
 - Tasks
 - distinguishing, 38
 - distinguishing goals, 155
 - type, 39, 81
 - Templates, 34, 51, 130
 - TKS, 31
 - Tools
 - categories, 110
 - requirements, 123
 - Typography, 77

 - UAN, 83, 119, 136
 - UML, 49, 51, 56, 60
 - Unit task, 36
 - Usability
 - categorization, 15
 - costs, 10
 - definitions, 14
 - evaluation, 21
 - means, 19
 - measuring, 22
 - metrics, 22
 - potential benefits, 10
 - process maturity, 24
 - usage indicators, 18
 - Usability properties, 89
 - Usage logs, 88
 - Use case, 53
 - Usefulness, 14
 - User problems, 100
 - User testing, 19
 - Utility, 14
 - UVM, 74

 - Visibility, 100
 - Visual design, 48
 - Visual language, 48
 - Visualizations, 48

 - WIMP, 13
 - Work culture, 42