Ontologies and Methods in Interdisciplinary Design

Martijn van Welie[1] and Gerrit van der Veer [2]

[1] Division Wiskunde en Informatica, FEW, Vrije Universiteit Amsterdam
De Boelelaan 1081, 1081 HV Amsterdam
Tel. +31-20-4447788
martijn@acm.org
http://www.cs.vu.nl/~martijn

[2] Division Wiskunde en Informatica, FEW, Vrije Universiteit Amsterdam
De Boelelaan 1081, 1081 HV Amsterdam
Tel. +31-20-4447764
gerrit@acm.org
http://www.cs.vu.nl/~gerrit

ABSTRACT *The goal of good User Interface Design (UID) is to develop usable interfaces. The need for more usable systems is becoming more and more important as people are becoming increasingly critical towards systems while the complexity of the systems increases. The User Interfaces of the next millennium will need to be designed better, easier to use, safer and above all they should effectively support users in their work. In order to design these systems, the input of multiple disciplines is needed since Computer Science does not have all the expertise needed for the design of truly usable systems. If all aspects of usable design are to be covered, the different disciplines need to integrate in order to create a synergistic effect. Integration can be achieved using methods that recognize this need and address the integration by using ontologies.*

Keywords *Usability, User Interface Design, Ontology, Interdisciplinary Design*.

1. INTRODUCTION

User Interface Design (UID) has never really been integrated into software engineering methods. In a way, it is the black sheep of the family. The nowadays very popular method UML [13] does not mention user interface design anywhere, as if that is not important for the design of good software. However, over the years, substantial proof has been given that the user interface is an important aspect of a system. A 1992 study found that an average of 48% of the code of applications is devoted to the user interface, and that about 50% of the implementation time is devoted to implementing the user interface portion [11], and the numbers are probably much higher today. For systems that are being developed to aid humans in their work, bad user interfaces can lead to deployment problems and the developed system may never reach its intended goals. In fact, in some cases a good user interface may be of life saving importance as is being demonstrated often in the aircraft industry. Now that computers and software have such a big impact on our daily life, for computer scientists as well as for most other people, user interfaces are becoming more important for other reasons as well. Many people have to use computers in their daily work and the systems they use are usually intended to aid them in their work. Unfortunately, it is often questionable if these systems *really* aid the users.

Recently User Centered Design [9] (UCD) methods have emerged to deal with the design of user interfaces. Those methods try to focus on the users' needs in a systematic way, ensuring a system that will be both usable and useful for its users. These methods usually have a trial-and-error character because of the many iterative user tests that are needed to ensure a high level of usability. What is lacking is a way to structurally developing a usable user interface without relying solely on user testing. In order to develop methods that can do that, it has to be clear what it means to design usable systems and what makes them usable or not. The term usability is used to indicate those aspects of a user interface that determine how usable a system is. An essential factor in designing usable systems is the recognition that it is not simply a pure engineering effort. UID is a domain that requires knowledge from engineering as well as other non-engineering disciplines, e.g. Psychology, art and Ergonomics. Without these disciplines, it is simply not possible to make right decisions concerning usability issues in design. In the next sections, we will elaborate on what a good design is and what makes it usable. Then we will use a framework of usability to see what knowledge is needed and how to effectively use it. Since knowledge outside of computer science is needed we will elaborate how the different disciplines can be brought together using methods and ontologies.

2. DESIGNING USABLE SYSTEMS

In the early days of computer science, computers were used for their computational power to do complex things that were impossible to do by humans within a certain amount of time. Nowadays, computers are more and more used in a "service" context where they are not used specifically for computing purposes and these systems are mostly operated by humans. If the reason for their usage is not computing power, what are the reasons? Usually some kind of gain is intended such as an increased productivity, a decrease in costs or efficiency gains for the organization. However, verifying such claims after a system has been built often turns out to be a disappointing exercise. Thomas Landauer discusses this issue in his book "The trouble with computers" [9]. He shows that the acclaimed gains are in fact often losses or that gains are to be found in other areas (if there are gains!). Designing usable systems is about designing the *right* system in the *right* way for the *right* people.

2.1 What is "Good" Design?

In order to produce a "good" design or system, it is important to define what a "good" design is. There can be many reasons why a design or a system is good. E.g. it was cheap to build, it looks good, it works, it meets the functional requirements, is fully documented, it came through a stress test etc. If we speak from a UID point of view, a system is "good" when it is *usable* by the end-users and it is *useful* for them. Usage by end-users is the *only* criterion for determining if a system is "good" because the end-users are the people using the system in real life. Determining how usable a system is remains a topic of much controversy but factors such as the performance-speed and learnability are regarded as key measures. The intention is that a system should fit the user as a glove! The user should not have to adapt his work to the software but the software should support the user in his work, although in many cases the work-processes need to be redesigned as well.

2.2 Why Design "Good"?

Designing "good" is simply a necessity in order to reach the intended gains of the system. The intended gains such as reduced costs, better service or greater efficiency exist in the real world and not on paper. They will therefor only occur when humans are *using* the system. If the intended gains cannot be perceived or proven, it is very unlikely that the system achieves its intended goals. No matter how bug free, modular, reusable or configurable the software is, in the end the actual usage is what counts. Of course, if the system is usable enough, the software architecture is also important for other aspects such as maintenance and integration purposes. The important issue is that for the user the system is seen as a black box and the user does not really care about what is inside.

3. USABILITY IN CONTEXT

In the past there were several terms used to denote that a system was "good" or usable. First it was "user friendly" which did not point out the real problem. A system does not need to be *friendly* to its users. Then there was "ease of use". This term was better but still not good enough. The issue whether a system is easy to use is too subjective and vague, capturing only a small part of the whole thing leaving out questions like "is the system *useful* for the user?" Nowadays we speak of *usability* to indicate how usable a system is. The term is much better because it covers much more than the previous terms. Usability covers the fact that it may depend on the person, the work and context whether a system is usable or not. The main problem with usability, both as a concept and a term, is that there is not (yet) an agreement on the exact meaning of usability. In the next sections, we will give a framework that gives structure to the concept of usability.

3.1 Defining Usability

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 and they mostly differ on more detailed levels. In the ISO 9241-11 [3] standard a rather abstract definition is given in terms of efficiency, effectiveness and satisfaction. "Efficiency" is defined as the *resources expended in relation to the accuracy and completeness with which users achieve goals* and "effectiveness" as the *accuracy and completeness with which users achieve specified tasks*. "Satisfaction" is a subjective measure and concerns the *comfort and acceptability of use by end users*. This definition approaches usability from a theoretical viewpoint and may not be very practical. Nielsen [12] has a slightly different definition that is specified in elements that are more specific. Nielsen only regards expert users when talking about efficiency although learnability is also directly related to efficiency. Memorability mainly relates to casual users and errors deal with those errors not covered by efficiency, which have more catastrophic results. A similar definition is given by Shneiderman [16]. Sheiderman 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 1, Shneiderman's definition is essentially identical to Nielsen's definition and only differs in terminology.

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

Table 1 Usability as in ISO 9241-11, B. Shneiderman and J. Nielsen

Table 2 shows the usability factors as described by Dix [5]. This categorization looks rather different from the ISO 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.

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

Table 2 Usability categorization by Dix et al.

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 measures for improving the usability of a design. On the other

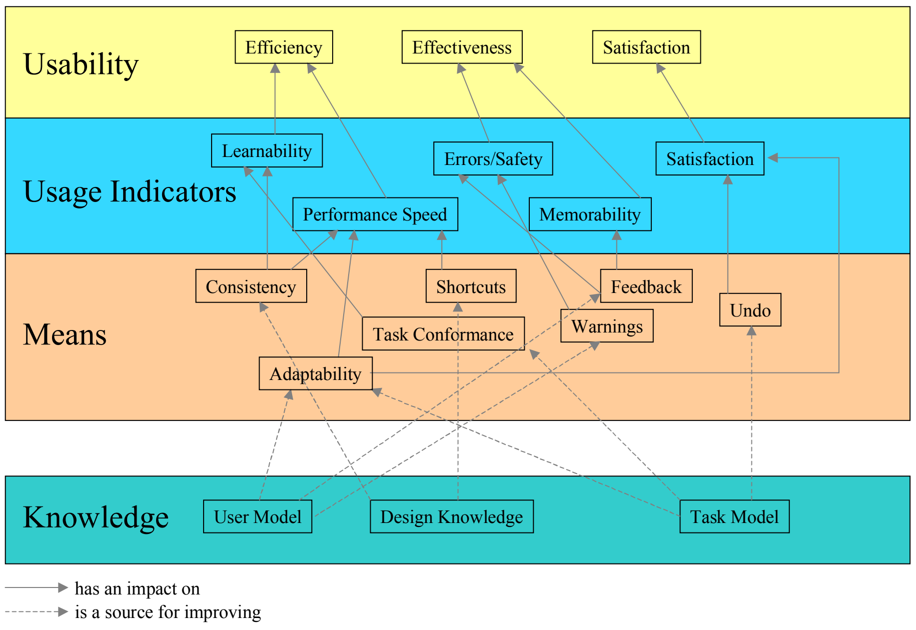


Figure 1 Layered model of usability

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.

3.2 A Layered Model of Usability

All the different definitions and principles make usability a confusing concept when actually designing a new system. Usually authors spent a lot 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 1 shows a layered model of usability 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 [3]. 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.
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 and are consequently not *goals* by themselves. For instance, consistency may have a positive effect on learnability and warnings may reduce errors. On the other hand, there may be good reasons for not complying completely with a consistent platform style. 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". It is up to the designer to find those optimal levels for each means. In order to do that the designer has to use the three knowledge domains (humans, design, and task) 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 1 are examples of means and usage indicators 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.

3.3 Dealing with Usability Issues

In practice, usability testing is done late in the design process. Using a prototype the user interface is evaluated by letting users use the prototype. The resulting data informs designers about the *usage indicators* which will need to be (subjectively) evaluated. Some aspects may be on a satisfactory level and others may not. Overall it needs to be determined of the usability level is good enough in relation to the intended goal of the system. The *usage indicators* may say that something is wrong, they do not say anything about the cause. Since the *means* affect the usage indicators, the designer needs to reflect of the usage of the means. The means are the “tools” you can use to fix any usage problems. However, they are complex tools since in most cases their usage depends heavily on their context. This context is described in the *knowledge domains*. The knowledge domains can give a designer the information needed when design decisions need to be made. If decisions are taking without consulting the knowledge domains, the decision is not well founded which leads to a “trial and error” process. In short, the knowledge domains are the basis for usable design!

4. USING THE KNOWLEDGE DOMAINS

4.1 Using User Knowledge

The systems we design are being used by humans so we need to know the abilities and limitations of humans. Especially cognitive and perceptual abilities are relevant to design. Humans have serious limitations when it comes to information processing and other tasks such as decision making, searching and scanning [16]. 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 short and long term memory can directly be used to improve learnability of systems. In the past, methods such as GOMS [4] and CCT [8] have tried to incorporate cognitive aspects to predict the influence of changes to dialogue aspects of a design. 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. Contextual aspects about users have a less direct impact on the design process and are strongly related to the position of a new system in the organization where it is going to be used. Generally speaking, usability problems are caused by a mismatch between the users' abilities and the required abilities that the system enforces on the users.

4.2 Using Design Knowledge

Every designer acquires skills and experiences during projects and that knowledge helps the designer in later projects. This *design knowledge* comes from both practical experience and from literature. Currently the amount of design knowledge available in literature is rather limited which makes the personal experience of the designer an important factor for the usability of the design. Basically the only concrete design knowledge that can be used during design is embedded into *guidelines*. Several guidelines exist but there is no agreement on the form guidelines should have. Some guidelines such as the Macintosh [1] or MS Windows [10] guidelines mainly describe a platform *style* and hardly contain concrete guidelines. The underlying assumption that applications that have been designed according to the guidelines have good usability remains unjustified. Other guidelines such as Mosier's [17] focus on narrow scoped list of guidelines dealing with detailed design choices and consequently they are quickly outdated by new developments of technology. 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 [5] 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 [14]. Some older guidelines were designed for designing character based applications and it is not clear in how far they apply to e.g.

WIMP interfaces or Virtual Reality interfaces. Another way of capturing design knowledge is in *design patterns* [2]. Such patterns describe generalized problems and proven solutions that can be immediately used in practice. Research on design patterns has just started and no concrete results are available yet.

Guidelines deal with both structural (the dialogue) and presentational aspects of a design. For example, guidelines on color use and button sizes refer to the presentation and guidelines on feedback and menu structure deal with dialogue. Usually no explicit distinction between dialogue 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 structural aspects. Because design patterns work from a problem to a solution it is more likely to find guidance on structural aspects emphasized in design patterns.

4.3 Using Task World Knowledge

Besides the design knowledge needed for a good design, every project also needs the right information about the specific design case for basing the design on. Both knowledge about humans and knowledge about design is domain-independent but the task world knowledge is different for every design project. Task analysis should provide the information for the requirements of the system both in the functional sense but also in the ergonomic and cognitive sense. The functional side of a task analysis can be transferred quite directly to the design but the ergonomic and cognitive side is very hard to transfer into design. First of all because it is not clear what the relevant information is; what needs to be known in the task model in order to contribute to a more usable design? Secondly, because cognitive aspects are difficult to translate into concrete design decisions. One of the weak points in task analysis research is that it is difficult to justify how task analysis helps to design more *usable* systems as far as this is not directly based on functional requirements of the systems. An answer to this question may be given if it can be defined which properties of a design make the design usable. It can then be seen which information the task model ideally needs to contain. One way to approach this problem is to use an ontology. The ontology does not describe one particular representation but it describes the semantic structure of the task world we describe when using representations.

4.3.1 A Task World Ontology

The task world ontology describes the way we look at the task world during task analysis. This ontology is derived from the conceptual framework of GTA [19]. It defines the relevant *concepts* and *relationships* between them that we regard relevant for the purpose of a task analysis. The ontology is of great importance because it is the conceptual basis of all information that is recorded and the way it is structured. Unfortunately, most task analysis methods do not define an ontology. Our ontology is derived from the three viewpoints from GTA and incorporates aspects of several other task analysis methods.

4.3.2 Concepts and Attributes

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 the next section will define their relationships in detail.

- **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.
- **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.
- **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.
- **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.

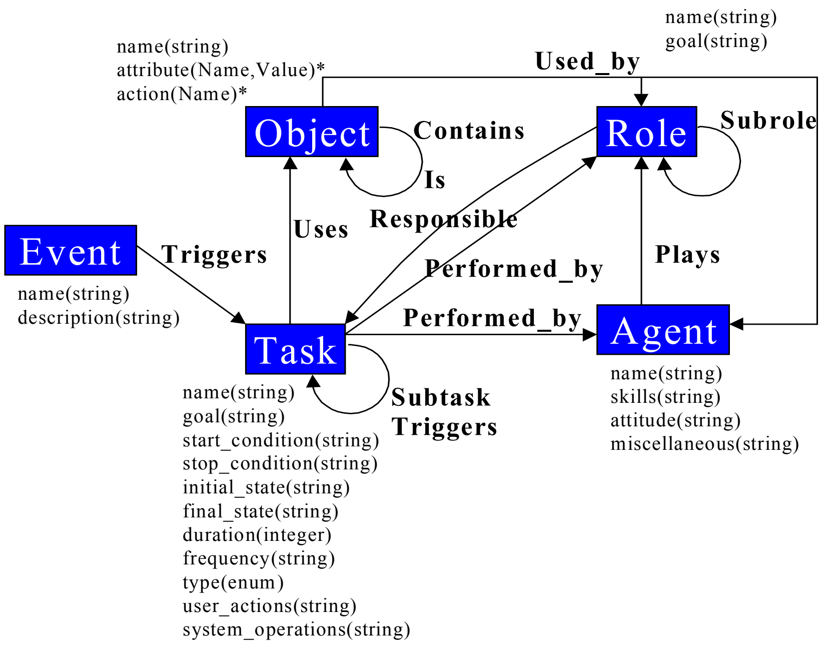


Figure 2 Concepts and relationships

4.3.3 Relationships

The concepts defined in the previous section are related in specific ways. In this section we sketch the relationships that we are using now. For each relationship the first-order predicate definition is given and explained. Figure 2 shows all the concepts and relationships together in a diagram.

- **Uses.** The uses(Task,Object,Action) relationship specifies which object is used in executing the task and how it is used. The Action specifies what is being done with the object. It typically changes the state of the object.
- **Triggers.** The triggers(Task/Event, triggeredTask, triggerType) relationship is the basis for specifying task flow. It specifies that a task is triggered (started) by an event or a task and the type of the trigger. Several triggertypes 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(Agent, Role, Appointment) relationship also indicates how this role was obtained. Currently, the Appointment parameter can be ASSIGNED, DELEGATED, MANDATED or SOCIAL.
- **Performed_by.** The relationship performed_by(Task, Agent/Role) 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.

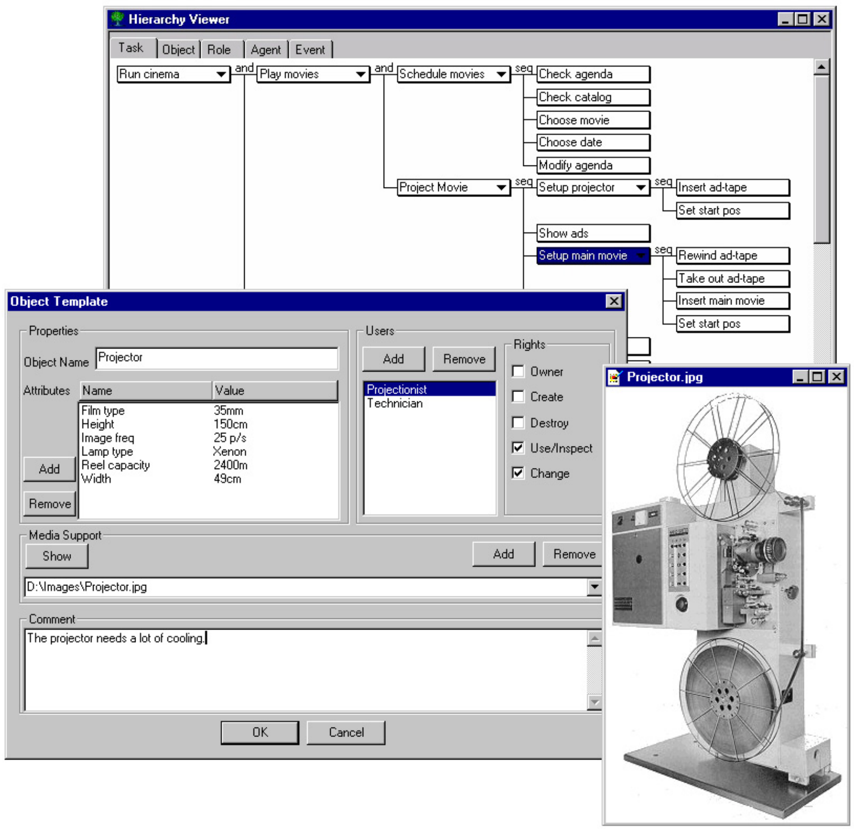
- **Subtask.** The subtask(Task, SubTask) relationship describes the task decomposition.
- **Subrole.** The subrole(Role, SubRole) relationship brings roles into a hierarchical structure. The subrole relationship states that a role includes other roles including the responsibility for the task that encompass the role. When a role has subroles the task responsibilities are added up for the role.
- **Responsible.** The responsible(Role, Task) relationship specifies a task for which the role is responsible.
- **Used_by.** The used_by(Object, Agent/Role, Right) 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 (CREATE and DESTROY), indicate ownership (OWNER), or indicate daily handling of objects (USE, CHANGE).

Besides these relationships, other relationships can be useful as well and some of them can be expressed using the above relationships. These relationships provide information that is shown in certain representations such as templates. For instance for each concept a related(Concept) predicate has been defined which differs for each concept.

4.4 Deriving representations

The ontology only defines a structure for the task model data and does not limit or dictate any representation. We developed a tool EUTERPE [20] that is based on a repository that contains the project data and all the representations are views on the repository. The task world ontology is specified in a logic programming language (Prolog) and is the main data structure for the repository. EUTERPE offers several different representations and all the representations are coherent because each representation is build up on the fly out of the same 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 relationship can be inspected. Naturally 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 a new fact *subtask(X,Y)* is asserted.

This way the users of EUTERPE can work with the representations without having to deal with the logic representation underneath. Besides task trees EUTERPE also offers templates that show detailed task information and some context information such as the objects used in this task or the roles that are involved in this task or any of the subtasks. A somewhat different kind of view is the web browser view. At the moment we are working on a process flow view based on workflow representations. Figure 3 shows some representations of EUTERPE.



Object Name: Projector

Name	Value
Film type	35mm
Height	150cm
Image freq	25 p/s
Lamp type	Xenon
Reel capacity	2400m
Width	49cm

Users: Projectionist, Technician

Rights: Owner [ ] Create [ ] Destroy [ ] Use/Inspect [✓] Change [✓]

D:\Images\Projector.jpg

The projector needs a lot of cooling.

Figure 3 Some representations.

5. GETTING THE KNOWLEDGE

In the previous sections it was shown that several different kinds of knowledge are needed in UID. Parts of the knowledge are "stable" and do not need to be obtained or created for each project. For instance, knowledge about limitations in memory capacity or motor skills remains (almost) unchanged and can be used for most projects. Other parts are different for every project, especially the task knowledge and this knowledge needs to be obtained by the design group.

5.1 Knowledge Elicitation

Collecting task knowledge for analyzing the current situation for a complex system has to start by identifying the relevant knowledge sources. In this respect, we refer to a framework derived from [7], see Figure 4.

task world knowledge	**Individual**	**Group**
explicit	**a**. knowledge and skills	**c**. models/stories/instructions
implicit	**b**. intuition/expertise	**d**. culture/community of practice

Figure 4 Dimensions of knowledge of complex task domains

Relevant task domain information may have to be collected focusing on different phenomena, using different methods of data collection. Based on an analysis of the character of the knowledge sources in this framework, different methods are identified to collect all information needed to construct a model of the current task world.

5.2 Collecting task knowledge

Going from knowledge that is available from professionals in the task world and domain experts, via knowledge that is present in the culture and in the social environment towards artifacts and the physical environment, we encounter knowledge in the cognitive psychological sense, awareness and anecdotal material in the culture, and traces of manufacture and use as well as environmental opportunities and constraints.

Going from explicit knowledge, via skills and rule based behavior, through intuitive and instinct-like behavior in individuals and groups and culture, we meet documented knowledge and conscious representations, stories and myths, as well as unspoken and unspeakable insights that still prove to be valid for guiding or monitoring adequate behavior in the context of use.

For task knowledge in cell a, psychological methods will be used including those elaborated by [6] and [15]: interviews, questionnaires, think-aloud protocols, and (single person oriented) observations. For knowledge indicated in cell b observations of task behavior will have to be complemented by hermeneutic methods to interpret mental representations (see [18]). For the knowledge referred to in cell c the obvious methods concern the study of artifacts like documents and archives. In fact all these methods are to be found in classical HCI task analysis approaches.

The knowledge indicated in cell d is unique in that it requires ethnographic methods like interaction analysis (see [7]). Moreover, this knowledge can be in conflict with what can be learned from the other sources, as is already shown in the examples presented in the previous sections. First of all, explicit individual knowledge often turns out to be abstract in respect to observable behavior, and turns out to ignore the situatedness of task behavior. Secondly, explicit group ‘knowledge’ (e.g., expressed in official rules and time schedules) often is in conflict with actual group behavior, and for good reasons. In fact, official procedures do not always work in practice and the literal application of them is sometimes used as a political weapon in labor conflicts as a legal alternative for strike. In all cases of discrepancy between sources of task knowledge, ethnographic methods will reveal unique and relevant additional information that has to be explicitly represented in task model 1.

The allocation of methods to knowledge sources should not be taken too strictly. In fact the knowledge sources often cannot be located completely in single cells of the conceptual map. The main conclusion is that we need these different methods in a complementary sense, as far as we need information from the different knowledge sources.

It can be shown that different techniques of date collection and date analysis are needed for different types of knowledge, and these techniques seem to map systematically on to the "two-dimensional" framework of knowledge sources. Related to the different types of knowledge and the techniques is the notion of reliability of collection of information, and the validity of the resulting knowledge. We consider the validity of the knowledge in relation to the history and time aspects of the task world. E.g., experts may base their current knowledge on training they received in a different phase of equipment application, and documents may reflect a rule that is yet to be accepted by the authorities that control task performance.

6. A NEED FOR MULTIPLE DISCIPLINES

In the previous sections, we have seen that computer science as a discipline is not enough to guarantee that all relevant aspects of UID are dealt with appropriately. Computer scientists are generally not experts in collecting task knowledge and they know little about visual design. Other disciplines excel in these areas but know little about software design. Therefor, multiple disciplines are clearly needed and these disciplines need to be brought together. Since not one discipline knows about "everything", they need to complement each other and work together on a project but still with their own views on the problems. We propose two ways of achieving this synergistic effect. First of all by using ontologies to describe "what we are talking about" and secondly by developing methods that have the interdisciplinary character 'built in".

6.1 Connecting People with Ontologies

The task world ontology described in section 4.3.1 was an example of an ontology about a certain domain. Representations are the visualizations of the information captured by an ontology. Since each discipline has their own models and representations, ontologies can help to relate these models in order to take away the confusion in understanding each other. Another useful aspect is that when building these ontologies you find out which method covers what part of an area and which issues it does not take into account.

6.2 Developing Methods

The other solution is to develop methods that structurally use the different views of each discipline. Ontologies can then be used to "connect" the views so that they can be semantically understood by each discipline. Each view is partial in the sense that is shows a particular aspect of a model and that is why multiple views are needed. They can complement each other so each discipline can use their own views which can be understood using ontologies.

UID is not a pure engineering discipline and therefor requires methods that recognize this. Each related discipline needs to be able to use their own expertise in the way they are used to, although they should be interpreted in the context of UID and not in their "usual" context. UID will be somewhat "alien" to the non-engineering disciplines such as Ergonomics and Ethnography but if their relation to UID is made clear co-operation can be very valuable.

7. CONCLUSIONS

There is an increasing need for developing systems that are more usable than the systems are nowadays. Designing truly usable systems requires knowledge about many aspects of human computer interaction. Both engineering disciplines as well as non-engineering disciplines are needed since not one discipline has expertise on all relevant areas. All of these disciplines need to integrate in order to create the necessary synergistic effect. Using ontologies and structured methods disciplines can be brought together. Ontologies allow the representations used by one discipline to be understood by others.

8. REFERENCES

1. Apple Computer Inc. (1992), *Macintosh Human Interface Guidelines*, Addison-Wesley Publishing Company.

2. Bayle, E. (1998), *Putting it All Together: Towards a Pattern Language for Interaction Design*, SIGCHI Bulletin, vol 30, no. 1, pp.17-24.

3. Bevan, N. (1994), *Guidance on Usability*, ISO 9241-11 Ergonomic Requirements for Office Work With VDTs..

4. Card, S.K., Moran, T.P. and Newell, A. (1983), *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Ass, Hillsdale.

5. Dix, A., Abowd, G., Beale, R. and Finlay, J. (1998), *Human-Computer Interaction*, Prentice Hall Europe.

6. Johnson, P. (1989), *Supporting system design by analyzing current task knowledge*, in: Diaper, D., Task Analysis for Human-Computer Interaction , Ellis Horwood, Chichester .

7. Jordan , B. (1996), *Ethnographic Workplace Studies and CSCW*, in: D. Shapiro, M. J. Tauber and R. Traunmueller, The Design of Computer Supported Cooperative Work

and Groupware Systems , North-Holland, Amsterdam.

8. Kieras, D. and Polson, P.G. (1985), *An approach to the formal analysis of user complexity*, International Journal of Man-Machine Studies, vol 22, no. 365-394.

9. Landauer, T. K. (1996), *The Trouble With Computers: Usefulness, Usability and Productivity*, MIT Press,Cambridge, Massachusetts.

10. Microsoft Press (1995), *The Windows Interface Guidelines for Software Design*, Microsoft Press,Washington.

11. Myers, B.A. (1995), *User Interface Software Tools*, ACM Transactions on Computer-Human Interaction, vol 2, no. 1, pp.64-103.

12. Nielsen, J. (1993), *Usability Engineering*, Academic Press,London.

13. Rumbaugh, J., Jacobson, I. and Booch, G. (1997), *Unified Modeling Language Reference Manual*, Addison Wesley.

14. Scapin, D.L. and Bastien, J.M.C. (1997), *Ergonomic criteria for evaluating the ergonomic quality of interactive systems*, Behaviour & Information Technology, vol 16, no. 4/5, pp.220-231.

15. Sebillotte, S. (1988), *Hierarchical planning as a method for task-analysis: the example of office task analysis*, Behaviour and Information Technology, vol 7(3), no. 275-293.

16. Shneiderman, B. (1998), *Designing the User Interface*, Addison-Wesley Publishing Company,USA.

17. Smith and Mosier (1986), *Guidelines for Designing User Interface Software*, MITRE.

18. Van der Veer, G.C. (1990) Human-Computer Interaction: Learning, Individual Differences, and Design Recommendations, Ph.D. Dissertation Vrije Universiteit, Amsterdam.

19. Van der Veer, G.C., Lenting, B.F. and Bergevoet, B.A.J. (1996), *GTA: Groupware Task Analysis - Modeling Complexity*, Acta Psychologica, 91, pp.297-322.

20. van Welie, M., van der Veer, G. C., and Eliëns, A. (1998), *Euterpe - Tool support for analyzing cooperative environments,* Ninth European Conference on Cognitive Ergonomics, Limerick, Ireland.