Groupware Task Analysis

MARTIJN VAN WELIE, GERRIT C. VAN DER VEER

Department of Software Engineering,, Vrije Universiteit, The Netherlands

Abstract

Introduction

Groupware Task Analysis (GTA, van der Veer, Lenting & Bergevoet, 1996) is a recent method that combines aspects from several methods. Systems being built nowadays are often not (only) used by single users. 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. When starting the design process of a complex interactive system it is often useful and needed to understand the current work situation. Description of this task world we will label "task model 1". In order to design for actual use, it is mostly needed to also consider the changes in the task world that will be the effect of developing and introducing the envisioned technology. Consequently, GTA stresses the need for designing and, hence, modeling the envisioned future task world, which we will label "task model 2". Task model 2 may in fact be considered a type of global design that includes not only the artifacts that are being developed but additionally their relation to the users and stakeholders, their goals and tasks, and the context of use.

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 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. In our design practice we have used so far a combination of classical HCI techniques such as structured interviews (Sebillotte, 1988), document analysis and observation, and CSCW techniques such as ethnographic studies and interaction analysis (Jordan, 1996). The combined knowledge of the current situation is described by task model 1, also referred to as the "current task model".

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 designed 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 bas been developed and is used.

Detail 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 here the details of technology and the basic tasks that involve interaction with the system need to be worked out. This activity consists of 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 focussed on a detailed description of the system as far as it is of direct relevance to the end-users. 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 the syntax (how task delegation to the system has to be expressed by the user). When making the transition from task model 2 to the UVM, the users' tasks and the relevant objects in the task world determine the first sketch of the application. The task and 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 design patterns. This part of design is not covered in detail in this chapter.

The framework that we discuss in this chapter is intended to structure task models 1 and 2, and, hence, may feature as 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.

Views on task worlds

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 b 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.

Modeling task worlds

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.

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 to states or particular configurations in the task world. Additionally, in complex task trees based on real life situations, asks 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 Detail. While in most cases a task hierarchy shows 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.

Modeling the work flow.

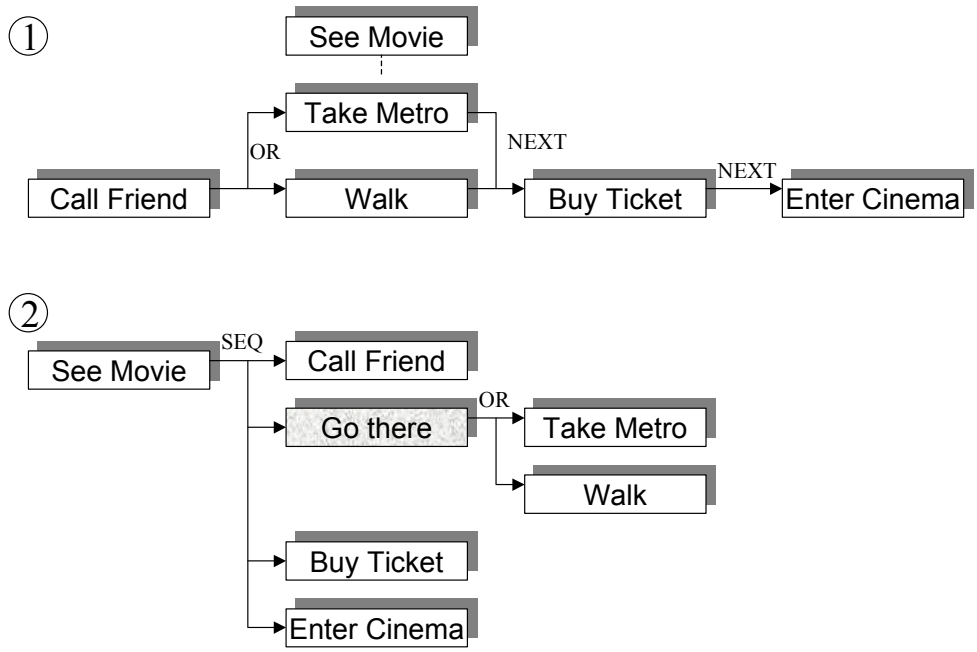

Figure 1: Problems with representation of time in trees

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 1, 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.

Modeling work artifacts

Artifact/object modeling is an addition to task analysis that resembles data structure modeling of the final design 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 basis for automatic UI generation have different requirements than models used for evaluation. For example, in GTA only the structure 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.

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 that 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 kinds 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).

Representations for Task Modeling

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 world. 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.

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. 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:

- To document and to communicate knowledge between designers.
- To analyze work and to find bottlenecks and opportunities.
- To structure thinking for individual designers.
- To discuss aspects of the task world within the design team.
- To propose changes or additions within the design team.
- 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.

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 2. 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 3. This issue becomes relevant when depicting large task models of more than about 25 tasks.

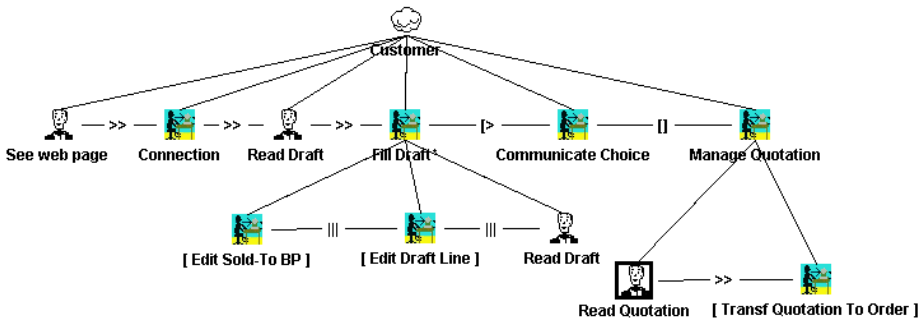

Figure 2: ConcurTaskTree example

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 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.

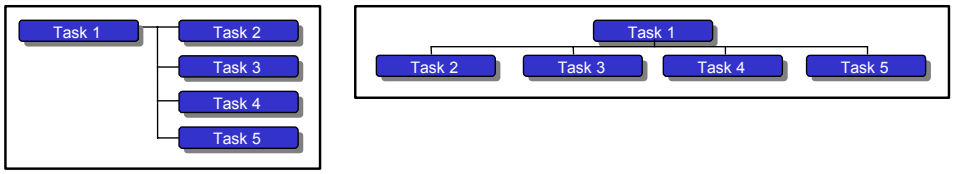

Figure 3: Depicting left-to-right versus top-to-bottom

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 focuses 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. 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 roles. UML has four representations that are directly relevant for task modeling:

- 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'.

- 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.
- 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.
- 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 community 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:

- The **Flow Model** represents the communication and coordination necessary to do the work.
- The **Sequence Model** shows the detailed work steps necessary to achieve an intent.
- The **Artifact Model** shows the physical things created to support the work along with their structure, usage, and intent.
- The **Cultural Model** represents constraints on the work caused by policy, culture, or values.
- The **Physical Model** shows 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 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 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.

Choosing a set of Representations for GTA

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 an improved collection of representations that cover the views of GTA. This collection of coherent representations is an attempt to provide a more useful collection of representations for practitioners.

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. 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. 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.

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 that 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.

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.

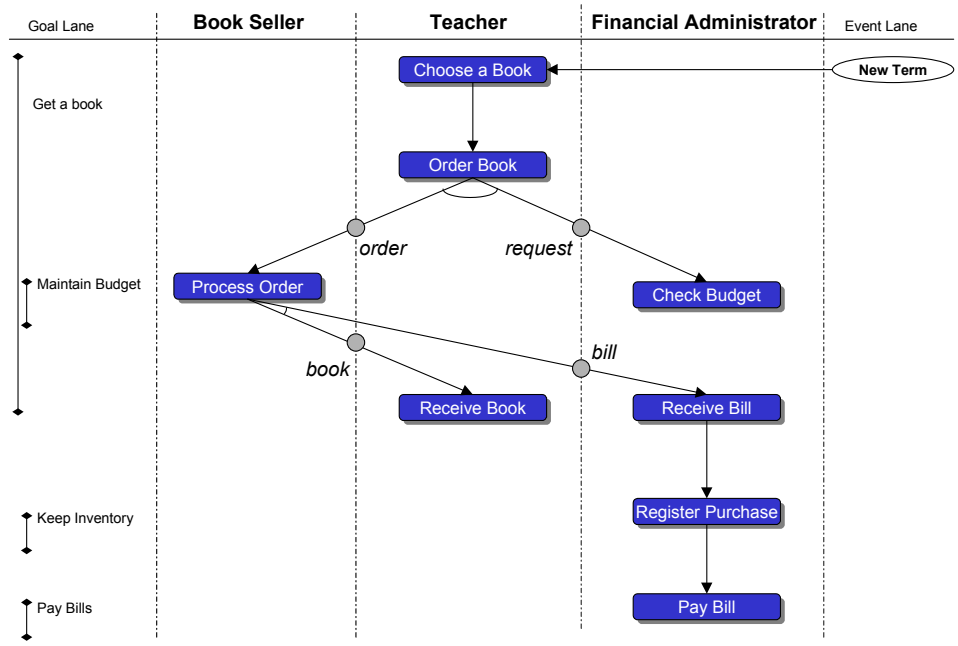

Figure 4: The WorkFlow Model

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 that 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.

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 shows that once the "teacher" has received the book his

goal is achieved but the scenario is not finished yet. Figure 5 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 loose 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.

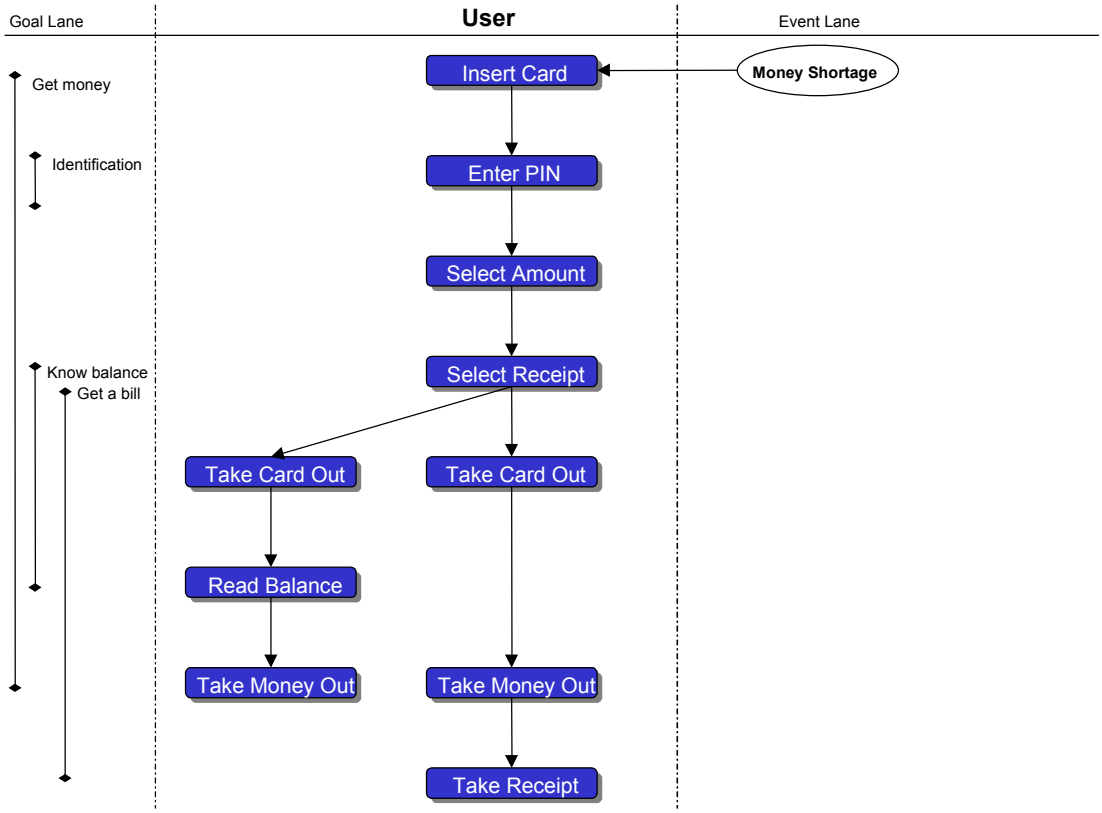

Figure 5: A Flow Model of a Dutch ATM

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 only are used in one task are not shown in this representation. For example, the PIN card itself is not shown in Figure 5. 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 it 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.

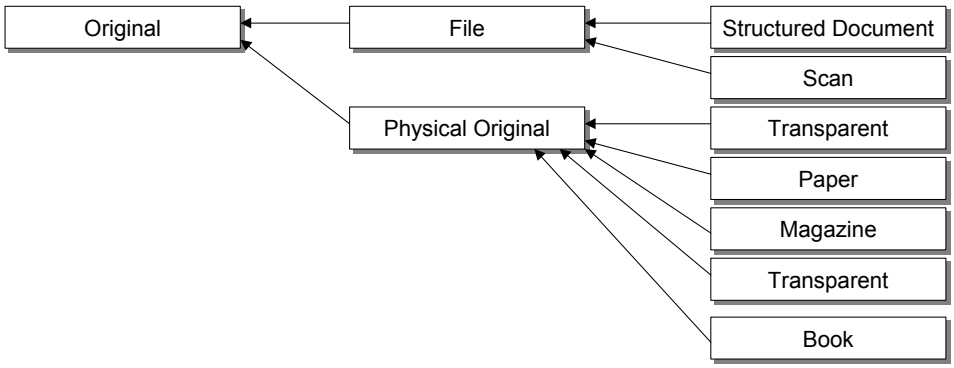

Figure 6: Example of a UML class diagram

Modeling the Work Artifacts. To model work artifacts two relationships between objects need to be represented: the containment and type relationship. In order to express containment and type, the UML notation can be used, see Figure 6. The objects themselves can be annotated with their attributes or their visual appearance. However, it is important to remember that we are only modeling objects and relations that are relevant to the user and not any irrelevant internal system objects. To some this may suggest that the task models describe an object oriented system model, which is not the case.

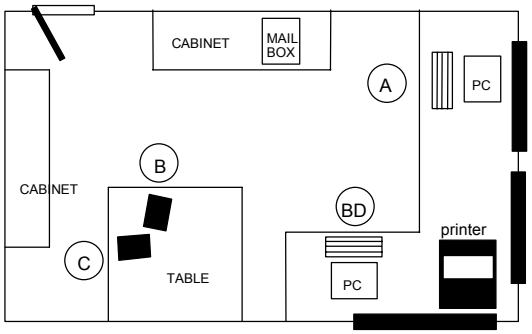

Figure 7: The Physical Layout Model

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 7 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).

The other model is the **Culture Model** (Figure 8**)**. 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.

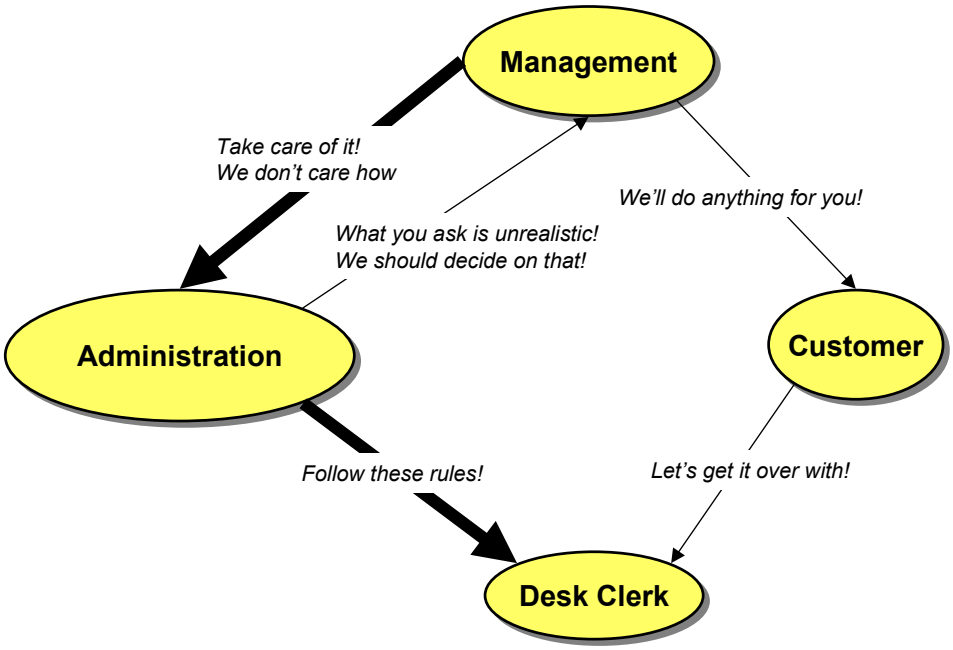

Figure 8: The Culture Model

We define the culture model as follows.

- Roles are represented as ovals.
- 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.

- 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".

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". 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. Animation can be used in simulations of scenarios or task models (Bomsdorf & Szwillus, 1999). 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.

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. We found that many problems fall in the same categories and have a 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.

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. 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 sub roles 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.

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. 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 were 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.
- Each agent should have at least one role.
- Each role should have at least one responsible task.
- Each object should be used in at least one task.
- Each task should be performed by at least one role.
- Each task should have at least one role that is responsible for it.
- Each object should have an owner (someone with the owner right).

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 itself.
- An object can not be of its own type.
- A task cannot have itself as a subtask.
- A task cannot trigger itself.
- A role cannot have itself as a sub role.
- For 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 the duration 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).

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:

- Comparing two current task models.
- Comparing two envisioned task models.
- 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 tasks 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.

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, 1999) 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 tasks 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.

Supporting Task-based Design

With the development of GTA, we developed a design environment / tool called EUTERPE (van Welie, 2001), that is currently available as public domain software from www.cs.vu.nl/~gerrit/gta. 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 (See Figure 9 for an example of a screenshot of the tool in use).

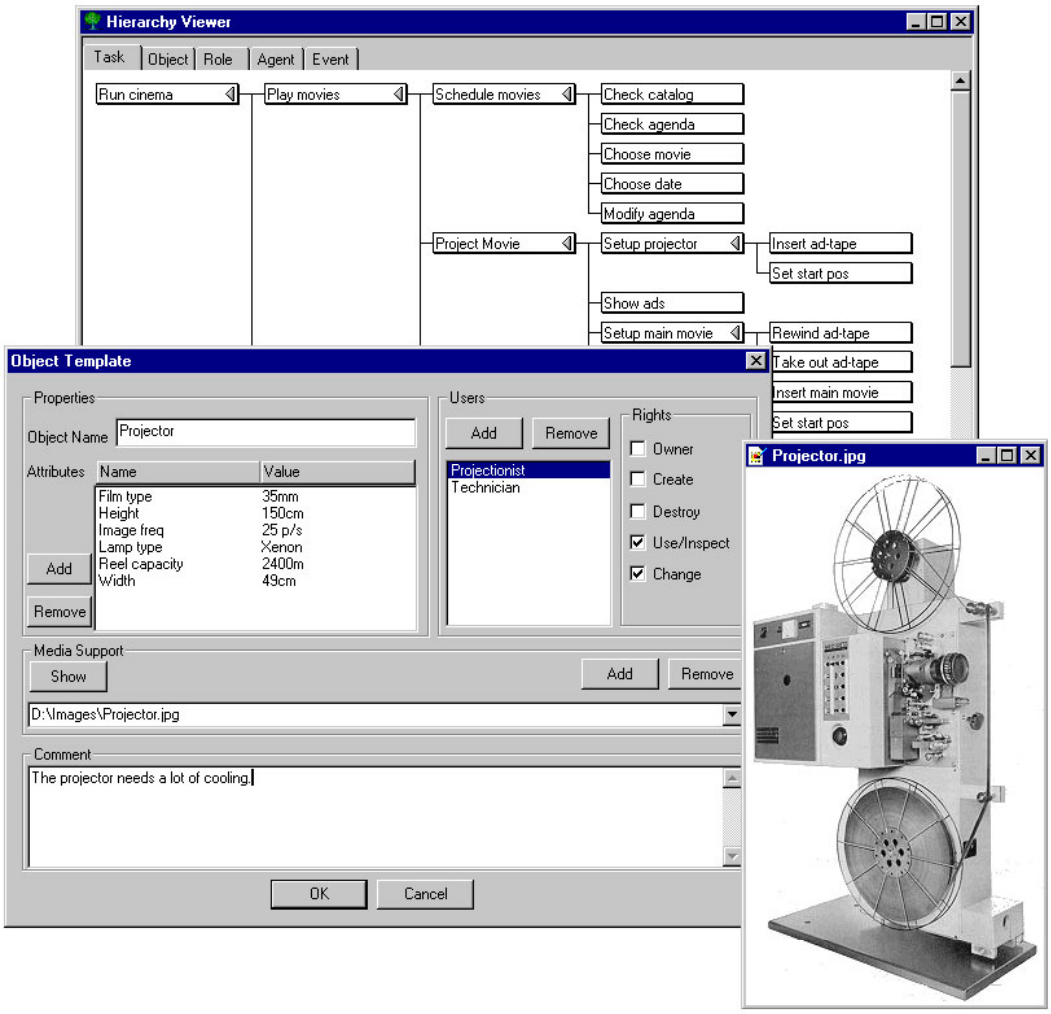

Figure 9: The Hierarchy Viewer, an Object Template and a Picture.

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 main GTA concepts (Tasks, Roles, Agents, Objects, Events), except for the Goal concept. 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.

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 16 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. Therefore, 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 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.

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 earlier.

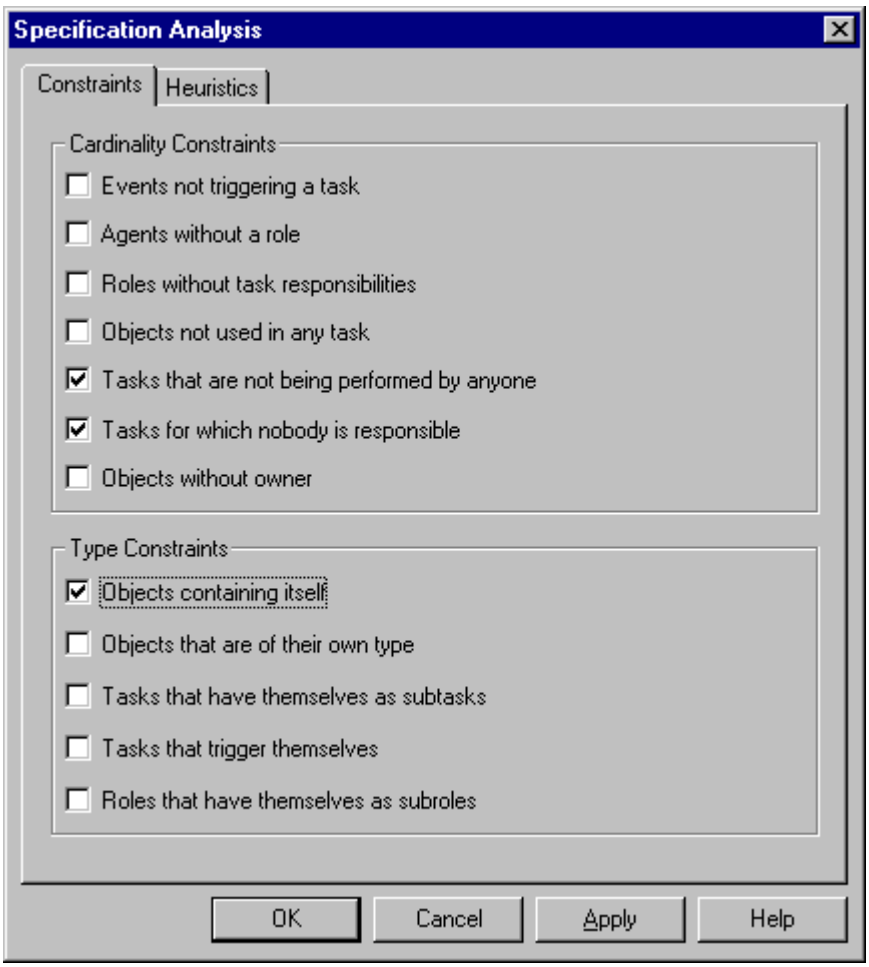

Figure 10: Evaluation constraints

Figures 10 and 11 show the dialog screens 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. The designer questions have been split into constraints and heuristics. Constraints apply to every specification and should ideally have zero results, see Figure 10. Heuristics can be used to analyze a specification, to find inconsistencies or problems, see Figure 11.

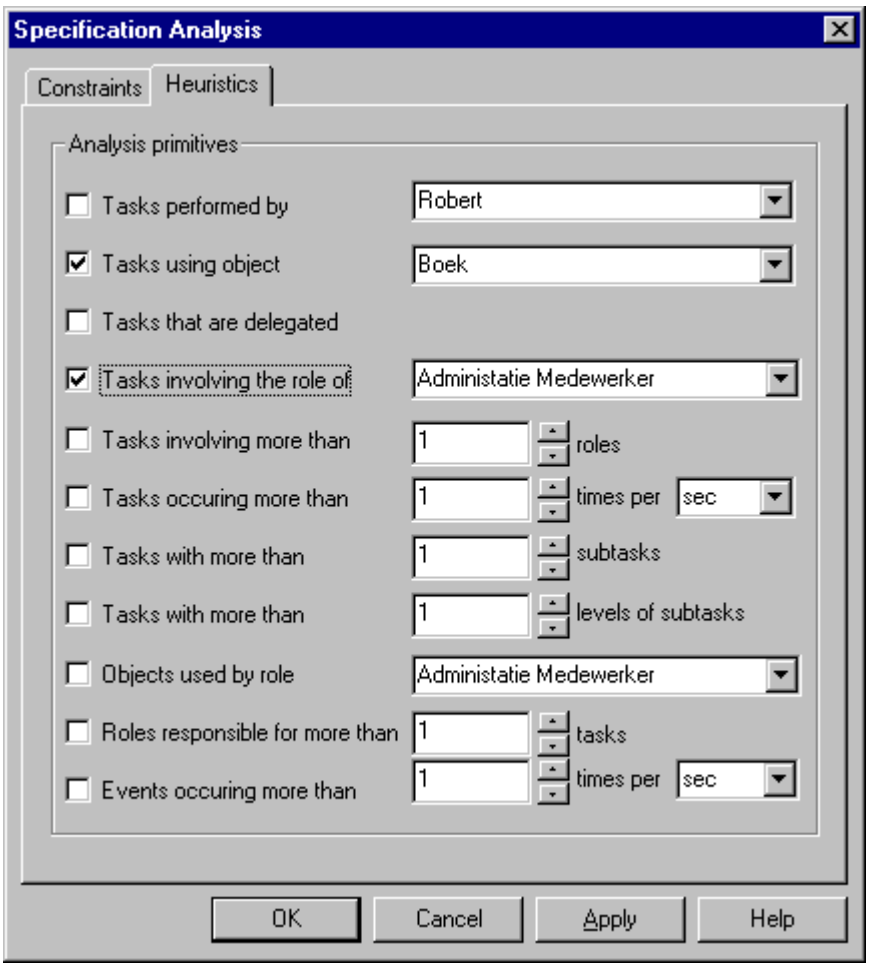

Figure 11: Evaluation using Heuristics

EUTERPE can process several queries in parallel. Each node found by a query is colored in the HierarchyViewer. If a node is selected in more than one query, it gets the color belonging to the last query. While this may not always seem a satisfactory solution, in practice this was never considered a problem.

From Task Analysis to Task 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.

Guidelines for Bridging the Gap

Holzblatt describes a 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 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:

- Develop an essential conceptual model of the new task world. This model describes the task world without any reference to tools and systems being used.
- Identify the major tasks and objects that need to be part of the system. These will become the high-level interface structure.

These steps result in the design of the future task model, task model 2 (see the introduction).

- Depending on the type of application structure the application based on a process or product metaphor.
- Create navigational paths in the interface structure depending on the task structure.
- Design the presentation using a platform style.

These steps result in the specification of the UVM.

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.

Specifying 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"*.

Task model and design solutions

The relationship between a task model and an interface design is a complex one. Task models are task/goal focused, almost by definition, while most user interfaces are object oriented. This means that some form of transformation *must* occur and both Dayton and Holtzblatt have given guidelines to do so. Although the task structure is central for workflow type of applications such applications are still very object oriented. The tasks may reappear in menu functions but there must still be objects to manipulate. It

is there very important to establish a link between the task structure and an object model for use in the interface. Since objects and the operations on them do directly dictate a task flow, the designer must make sure that the original task flow is optimally supported in the detailed design. For example, parallel task may require several functions to be accessible at all times and none of them can be model. Another example is that certain sequential tasks may actually be presented sequentially while in other cases the same result is achieved implicit by a sequence of operations on objects.

The final detailed design is good when it supports the task model or even better, improves on it. When design principles are used effectively the number steps in the detailed design may be reduced in comparison to the task model. This is where creativity comes in and the designer's expertise is of high importance. However, when the task model is used as a basis for design it must be able to answer the design questions designers have, such as:

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

We believe that the viewpoints of GTA and the modeling techniques we use provide sufficient information to answer such design questions. In the process of refining GTA such design questions have led to additions or changes of our techniques. Task modeling for design has different requirements from cognitive task modeling in general. The goal is to model not to model too much and not too less. In practice, resources are always limited so it is of utmost importance that a task analysis can be done efficiently while at the same time it should be optimally effective for design.

Application of GTA in Industry

In this section, we discuss a case where task-based design was applied in an industrial context. It shows what kind of issues needs to be faced when performing a task analysis and the kind of results that can be obtained using

a task-based design approach. The case discusses a redesign of an Austrian industrial security system. Other examples of practical applications collected so far are several design cases for the Dutch tax office, and design cases at Philips Design.

Seibersdorf 's re-design of security systems

Our task-based design has been 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 its original version has been 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 status of the building or premises 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 12.

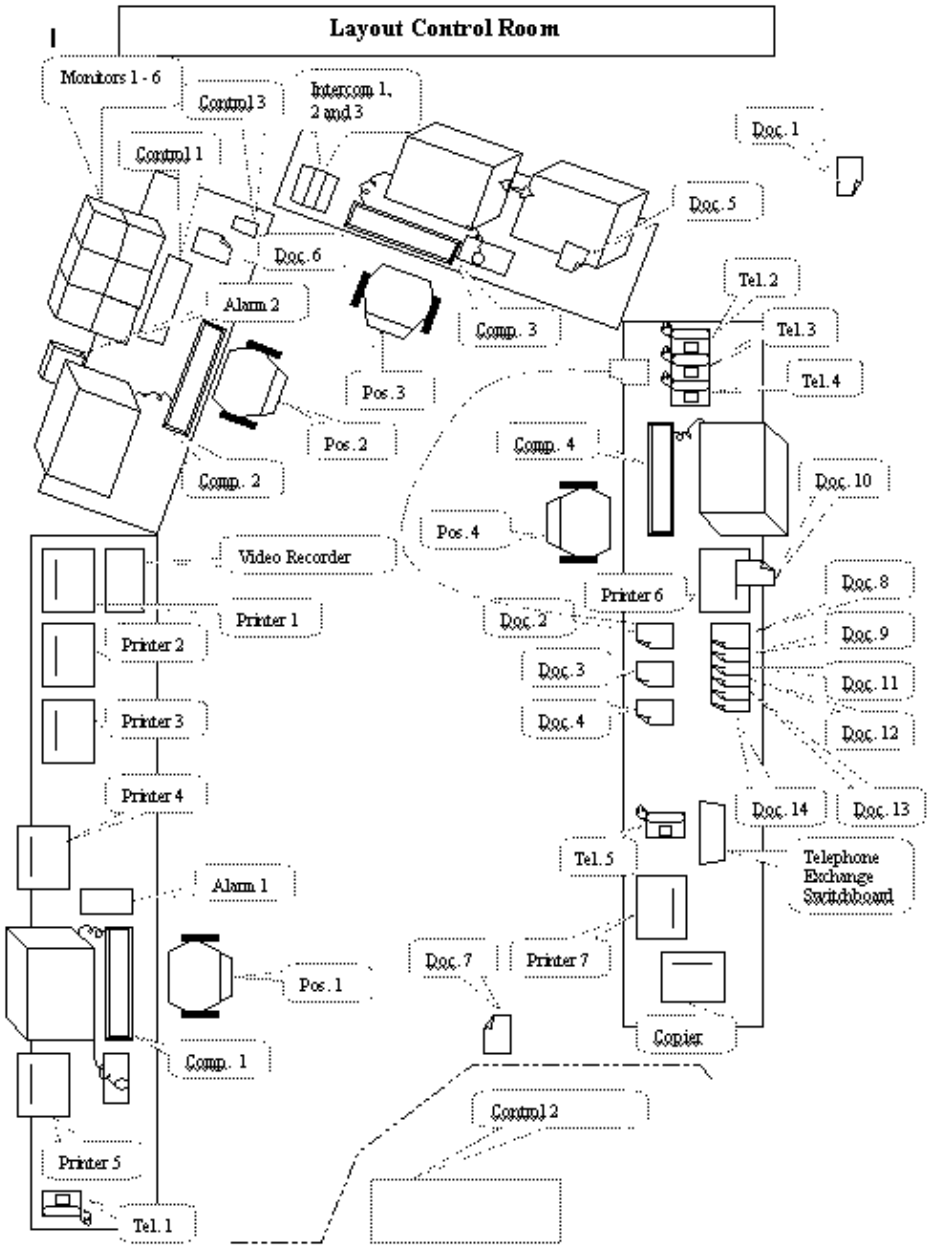

Figure 12: Layout of a control room

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:

- 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 of the researcher at ARCS, outside European funding, to apply the method also on other projects.
- 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.
- Task-based design (in this case: GTA) as an "official" method. This offers a more solid base to work and communicate from. One does not start from scratch. In getting support for gathering your data it is turned out to be crucial 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 proved to be the greatest help. It worked as a kind of 'check-list' to focus attention on things that matter in performing tasks. On the other hand, during information collection the most important attitude of the analyst is openness 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 open ended-ness associated with a more 'ethnographic' style of performing analysis. That the method leaves enough room for this without loosing 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 detail design (the UVM) are incorporated in the project. Technical design recommendations were done by iteratively developing sketches of design alternatives and providing these as mock-ups in the real work situation. 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 situated 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.

Application Issues of Task-based Design

During the application of GTA in industrial contexts, several application issues were encountered. In the next sections, we discuss these issues. These issues all reoccurred to a certain extent in all cases we collected so far, and together they provide a critical view on the practical value of task-based design.

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 regarding practice. For most designers task analysis is a new technique that 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. 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 (things to do) while it later becomes clear that they are actually goals (states to be reached). 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 often have difficulties dealing with hierarchical structures in user interfaces. For some designers, it is 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 to collect information on the context of use. 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 currently used formalisms 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.

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 task based user-interface design 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.

Conclusions

References

Annett, J. & Duncan, K. (1967), 'Task analysis and training in design', *Occupational Psychology* **41**, 211–221.

Beyer, H. & Holtzblatt, K. (1998), *Contextual Design*, Morgan Kaufmann Publishers.

Bomsdorf, B. & Szwillus, G. (1999), Tool support for task-based user interface design, *in* 'Proceedings of CHI 99, Extended Abstracts', Pittsburgh PA, United States, pp. 169–170.

Britton, C. & Jones, S. (1999), 'The untrained eye: How languages for software specification support understanding in untrained users', *Human-Computer Interaction* **14** , 191–244.

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.

Cooper, A. (1995), *About Face, the essentials of user interface design*, IDG Books Worldwide.

Fitts, P. (1954), 'The information capacity of the human motor system in controlling the amplitude of movement', *Journal of Motor Behavior* **47**, 381–391.

ISO (1988), *ISO/IS 8807 Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on Temporal Ordering of Observational Behaviour*.

Johnson, P., Johnson, H., Waddington, R. & Shouls, A. (1988), 'Task-Related Knowledge Structures: Analysis, Modeling and Application', *People and Computers IV* pp. 35–62.

Jordan, B. (1996), Ethnographic Workplace Studies and CSCW, *in* D. Shapiro, M. Tauber & R. Traunmuller, 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.

Lim, K. & Long, J. (1994), *The Muse Method for Usability Engineering*, Cambridge University Press.

Macinlay, J. (1986), 'Automating the Design of Graphical Presentations of Relational Information', *ACM Transactions on Graphics* **5**(2), 110–141.

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.

Mullet, K. & Sano, D. (1995), *Designing Visual Interfaces: Communication Oriented Techniques*, SunSoft Press, Prentice Hall.

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. Lindegaard, eds, 'Proceedings of INTERACT '97', Chapman & Hall, Sydney, pp. 362–369.

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.

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

Scapin, D. & Pierret-Golbreich, C. (1989), 'Towards a method for Task Description: MAD', *Work With Display Units* (89), 371–380.

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.

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.

Tufte, E. (1983), *The Visual Display of Quantative 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. 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* 'Eigth 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. & 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. (2001), Task-based User Interface Design. PhD thesis, Vrije Universiteit Amsterdam.

Visio Software (1999), *Visio*.

URL: *http://www.visio.com*

Wood, L., ed. (1997), *User Interface Design: Bridging the Gap from User Requirements to Design*, CRC Press.