An Ontology for Task World Models

Martijn van Welie, Gerrit C. van der Veer, Anton Eliëns

Vrije Universiteit, Department of Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, Holland
+31 20 4447788, {martijn,gerrit,eliens}@cs.vu.nl

Abstract. Many different task modeling methods exist. In this paper, we discuss 1) ingredients common to most task models, 2) how task modeling relates to the design of user interfaces, and 3) our proposed ontology for task analysis. We then show our task analysis tool that is based on the ontology. It is our belief that task models should be based on an ontology that describes the relevant concepts and the relationships between them, independently of any used graphical representations. Such an ontology helps to understand the different task modeling methods and it can also be operationalized for use in tools.

Keywords. Groupware Task Analysis (GTA), task models, task analysis, ontologies, user interface design (UID), CSCW.

1 Introduction

It has been generally accepted that task analysis may substantially contribute to the design of usable products because it focuses on the end user. Task analysis investigates users' characteristics and the task world of which they are a part. The information should be recorded in a task model that captures relevant aspects of the users and their task world. The resulting model should help designers in their process of designing a new product.

What is and is not relevant for the design process is determined by two factors: by what the analyst wants to record and by the task modeling method that is used. Each modeling method has certain defined concepts and relationships that are used to record information about the users and their task world. The concepts (e.g. tasks, roles, etc.) often come from the field of psychology and are clearly defined, but the relationships between concepts are usually only roughly sketched, leading to confusion when performing the actual task analysis.

Although specific task models differ, they have many things in common (e.g. task decomposition and task flow). Almost all task modeling methods also use graphical representations to show the information of the model. It is our belief that task models should be based on an ontology that describes the relevant concepts and the relationships between them, independently of any used graphical representations. An ontology can be defined as a view on a "world" that describes the nature and relations of it. In this case the task world of users is being described. In order to be useful in task analysis such an ontology should be rich enough to accommodate the extraction of the information needed to generate all the commonly used visual representations, e.g. task trees and flow diagrams. Different graphical representations would then show certain aspects of the ontology depending on the purpose of the view.

In this paper we explore the common ingredients of task models, and then introduce our ontology that makes all concepts and their relationships explicit. The ontology is the result of our attempt to clarify the concepts and relationships of our conceptual framework *Groupware Task Analysis* (GTA)[14]. We also provide an example of a tool based on this ontology. We believe that the use of this ontology could improve the results of task analysis firstly because it can be used as a reference model and secondly because it helps to structure the activity of task analysis when the used tools are based on the ontology.

2 Ingredients of task models

Task models are usually filled with several ingredients that are then related in some way. Some ingredients are less common than others, as the ideas about them have not yet stabilized; for instance, with the recent developments in CSCW systems and agent technology, modeling group processes has become important, although it is not yet clear how this should be done. This section will discuss common ingredients .

2.1 Task Decomposition

Task Decomposition is the most common ingredient of task models. It results in the classical task tree usually enhanced with constructors that indicate time relationships between tasks. This is probably the oldest ingredient and has a strong psychological basis; humans think in a structured way about their activities, which can be captured in a task decomposition[11].

Tasks and Goals. A common definition for a task is "an activity performed to reach a certain goal." However, in practice the relationship between tasks and goals is not always so clear. Some methods presume a 1-to-1 mapping between tasks and goals; for instance a *Task Knowledge Structure* (TKS[3,5]) contains a goal substructure (which would be called a task substructure by others' methods). Other models, such as GTA[14] and *Méthode Analytique de Description* (MAD[10]), allow a goal to be reached in several ways. In these models, the goal of a task is a specific state that is reached after successful execution of the task. In complex task trees based on real life situations, tasks near the leaves in a tree are usually connected with individual goals, and tasks represented by high level nodes are often closely tied with organizational goals[16]. When analyzing the tasks of an individual this is less apparent than when analyzing tasks on an organizational level. 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.

2.2 Task Flow Specification

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 (although mixing time and task decomposition).

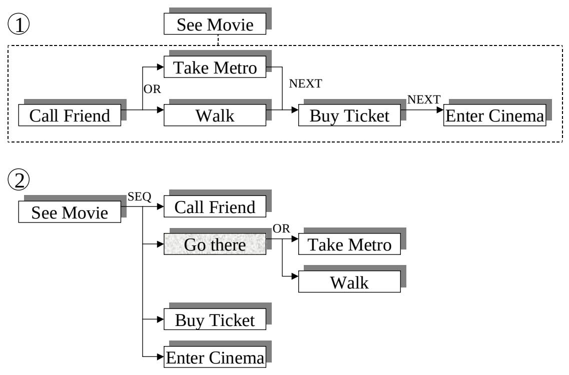


Fig. 1. Going to the cinema

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 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 Paterno's "ConcurTaskTrees"[9] 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.

2.3 Object Modeling

Object Modeling is an addition to task analysis that comes close to the data structure modeling of the final design and implementation. The purpose is to say something about the objects, as they are physically present in the task world or mentally present in the user's mind. Not every object may be directly included in the new design but in the case of models for automated user interface (UI) generation there is usually a very strong link between objects and UI widgets, such as buttons or menus.

The question remains how much object modeling should be in task models. Extensive data modeling does not appear to directly help in improving the usability of the product. It also depends heavily on the purpose of the task model; models used as a basis for automatic UI generation have different requirements than models used for evaluation. For example, in GTA only the structure of the objects and the tasks they are used in are recorded. Other models such as ConcurTaskTrees[9] and TKS[3,5] also include actions that are performed on the object.

Information passing. 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[9] describe this with task input and task output objects. The output of one task is given as input to the next task, thus passing information.

Another way to describe changes is to specify the initial and final states in terms of object attribute values. 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.

2.4 Recording of Concept Characteristics

Another common part of most task analysis models is the recording of relevant characteristic attributes of the concepts. Often mentioned attributes of tasks are pre- and post conditions, state changes, duration, etc. However, what specifically is recorded can be very different depending on the level of detail or purpose of the model. While there is no standard for concept recording, there are guidelines like the ISO 9241-11 standard[1] that suggest what could be recorded for tasks, actors and parts of the task world.

2.5 Task World Modeling

Task World modeling is for investigating the users and the world in which they act. In the past most methods have focused on modeling one user and that users' tasks. However, in current applications, group aspects are becoming very important. Every major software supplier now develops its products for use in multi-user distributed environments. 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 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 add perspective 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 actors' characteristics gives relevant information that can be used in design. The roles then need to be attributed to actors. In TKS[3,5] 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 actor 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 regarding 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 op the movie projector. Mayhew[7] 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). Therefor in GTA the concept of actor was renamed *agent* to avoid any confusion.

Events. When designing it is useful to know what factors influence the actor when he is performing the task and what impact they have. *Events* model dynamic aspects of the task world: things that happen in the task world, over which the actor does not always have direct control (e.g. the film breaks, or the cinema has a power failure). Sometimes there may be no need to explicitly incorporate the event in the new design but in other cases incorporation is more important. For example, it may prove very useful to model the projectionist's reaction to the event of the film breaking.

3 Task modeling for user interface design

After looking at several aspects of task models, some questions remain unanswered. What distinguishes UI task models from other task models (such as multi agent models in the field of *Artificial Intelligence*)? More importantly, what exactly should a task model include in order for its use to improve the usability of systems in the task world? We will look at the last question more precisely.

Markopoulos and Gikas[8] take a formal approach in using task models in system design. They argue that although a designer has a task model, it is still unclear how it helps him in designing. A task model needs to be formalized in order to conform to the task world *and* be useful in UI generation system.

In general UI task models should be able to represent psychological, social, environmental and situational aspects of the actors and their tasks. Hierarchical structures such as task decompositions and object structures can give a hint to a good system structure but for other aspects, such as user characteristics or timing properties, the relationship with the design is less obvious. Task analysis literature shows that each task model is designed for a certain purpose. For example, there are task models that can be used to:

- **Validate**. Some models are only meant as informal validating tools. They validate if the designers know enough about the task world for which they are designing. Methods such as HTA[6] are in this category. These methods result in quite informal models and do not have a very strong direct relationship with the design.

- **Generate User Interfaces**. Models like HUMANOID[12] and ADEPT[4] are designed to automatically generate a prototype of a user interface. To make the generation possible they need to be very precise and do not really produce human readable specifications. They are, not surprisingly, focussed on specifying the user interface behavior in detail, which makes them much more formal than the previous category.
- **Aid Design**. A more advanced use of a task model as a validating tool is to use it as a design aid. Methods like GTA[14], TKS[3,5] or GOMS[2] give a designer some "handles" for the design. Extensive task modeling gives concrete information about which objects or structures should be reflected in the design. Modeling the roles and agents can give information about the need for user identification or adaptability of the user interface for users. These methods are more formal than methods such as HTA[6] but less formal than UI generation methods such as ADEPT.

In principle a task model should not be used for specifying the user interactions with the system. Notations like User Action Notation[17] or ETAG[13] allow user actions to be specified in detail. What is needed is a mechanism to link a task analysis model to a user interaction model. That way the structure and task flow of the interaction model can be constructed out of a task model. A prerequisite for this is a formalized task model. This is however still the missing link in user interface design. Task models are often too informal which makes them hard to connect with interaction models. A relatively new approach called model-based interface design can be seen as an attempt to connect task models and interface models in a more fluent way. In this approach the design process consists of refining several models that each represent a different aspect of the design. A formal definition language that is used in every model relates all models with each other. Currently the task models used in such approaches are rather basic which limits the power of this approach. In the next sections we will present an ontology that may fill in this gap in user interface design because it gives a formal definition of the concepts and relationships that can be used to describe a task world.

4 An ontology

So far we have discussed common ingredients of task models, and some ways task models are used in the design of user interfaces. Now we propose an ontology that incorporates the mentioned ingredients, each to a certain extent. Although the ontology is mostly based on GTA it can serve as a reference for comparing other models such as TKS and MAD as well.

It is called an ontology because it describes logical relationships between concepts, something only informally done (if at all) by the various task analysis models that are currently being used. The ontology does not imply any graphical representation because it is defined in concepts and relationships and it therefor makes an extra abstraction.

4.1 Concepts and Attributes

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

Object. An object refers to a physical or non-physical entity. A non-physical entity could be anything ranging from messages, passwords or addresses to gestures and stories. Objects have attributes consisting of attribute-name and value pairs. What can be done with an object is specified by actions, for instance *move*, *change*, *turn off* etc. Furthermore, objects may be in a type hierarchy and can also be contained in other objects, for example a form may contain an address field, and a cinema can contain a snack bar. Objects are typically used in tasks but they can also influence the task execution sequence when they cause events to occur.

Agent. An agent is an entity that is considered active. Usually agents are humans but groups of humans or software components may also be considered agents. Agents are not specific individuals (like Chris) but always indicate classes of individuals with certain characteristics. Attributes of the agent can include *skills*, *attitude* and other *miscellaneous*. Agents perform tasks and always play certain *roles* within the task world.

Role. A role is a meaningful collection of tasks performed by one or more agents. The role is meaningful when it has a clear goal or when it distinguishes between groups of agents. A role is consequently *responsible* for the tasks that it encompasses. Roles can be hierarchically composed and are assigned to an agent in a certain way. The role can be obtained by assignment, delegation, mandate or because of a situational context.

Task. A task is an activity performed by agents to reach a certain goal. A task typically changes something in the task world and requires some period of time to complete. Complex tasks can be decomposed into smaller subtasks. Tasks are executed in a certain order and the completion of one task can *trigger* the execution of one or more other tasks. A task could also be started because of an event that has occurred in the task world.

Important for the task concept is the distinction between unit tasks and basic tasks, where (ideally) a unit task should only be executed by performing one or more basic tasks. The relationship between the unit task and basic task is interesting because it can indicate the problems that an agent may have in reaching his goals.

A unit task is defined by Card, Moran and Newell[2] as the simplest task that a user really wants to perform. A basic task[13] is a task for which a system provides a single function. Usually basic tasks are further decomposed into user actions and system operations. A user action is an action done by the human users that is only meaningful in the context of its basic task (e.g. a key press). A system operation is an action done by a system; it is not a typical task because it, as such, serves no goal for the user. The *type* attribute captures the task type, which can be either *unit*, *basic* or *composite* (consisting of unit, basic and/or composite tasks). In case of a basic task the *user_actions* and *system_operations* attributes are valid.

Tasks are started because of the completion of other tasks or because of events. For successful execution of a task a number of start-conditions have to be fulfilled. The stop-condition specifies when the task has reached completion. The changes in the task world that have taken place because of this task are described by the difference in the initial and final state.

Event. An event is a change in the state of the task world at a point in time. The change may reflect changes of attribute values of internal concepts such as Object, Task, Agent or Role or could reflect changes of external concepts such as the weather or electricity supply. Events influence the task execution sequence by *triggering* tasks. This model does not specify how the event is created or by whom.

4.2 Relationships

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

Uses. The uses(Task,Object,Action) relationship specifies which object is used in executing the task and how it is used. The Action specifies what is being done with the object. It typically changes the state of the object.

Triggers. The triggers(Task/Event, triggeredTask, triggerType) relationship is the basis for specifying task flow. It specifies that a task is triggered (started) by an event or a task and the type of the trigger. If the task is part of a choice the triggertype is OR. Other possible triggers are: AND for specifying parallel executed tasks, and NEXT for indicating linear succession of tasks. The triggers relationship is very similar to triggers used in workflow representations and allow for specifying concurrency in various ways. We will explain later how this relationship is used to generate a visual representation of a flow diagram.

Plays. Every agent should play one or more roles. The plays(Agent, Role, Appointment) relationship also indicates how this role was obtained. Currently, the Appointment parameter can be ASSIGNED, DELEGATED, MANDATED or SOCIAL. In the future, we want to look more closely at role appointing, so this relationship may undergo changes in subsequent versions of this ontology.

Performed_by. The relationship performed_by(Task, Agent/Role) specifies that a task is performed by an agent. This does not mean that agent is also the one who is responsible for the task because this depends on his role and the way it was obtained. When it is not relevant to specify the agent that performs the task, a *role* can also be specified as the performing entity.

Subtask. The subtask(Task, SubTask) relationship describes the task decomposition.

Subrole. The subrole(Role, SubRole) relationship brings roles into a hierarchical structure. The subrole relationship states that a role includes other roles including the responsibility for the task that encompass the role. When a role has subroles the task responsibilities are added up for the role. For example, the role of *snack bar worker* may have the subroles of *popcorn maker, snack bar cashier*, and *daytime janitor*.

Responsible. The responsible(Role, Task) relationship specifies a task for which the role is responsible. Continuing the example above, the snack bar worker role is *responsible* for all the tasks of the subroles popcorn maker, snack bar cashier, and daytime janitor.

Used_by. The used_by(Object, Agent/Role, Right) relationship indicates who used which object and what the agent or role can do with it. The agents' rights regarding objects can be of existential nature (CREATE and DESTROY), indicate ownership (OWNER), or indicate daily handling of objects (USE, CHANGE).

The relationships of this model form a minimal set of relationships that exist. However, when using this model there are also other relationships that can be of interest. Consider for instance a relationship *involved_role* that indicates which roles

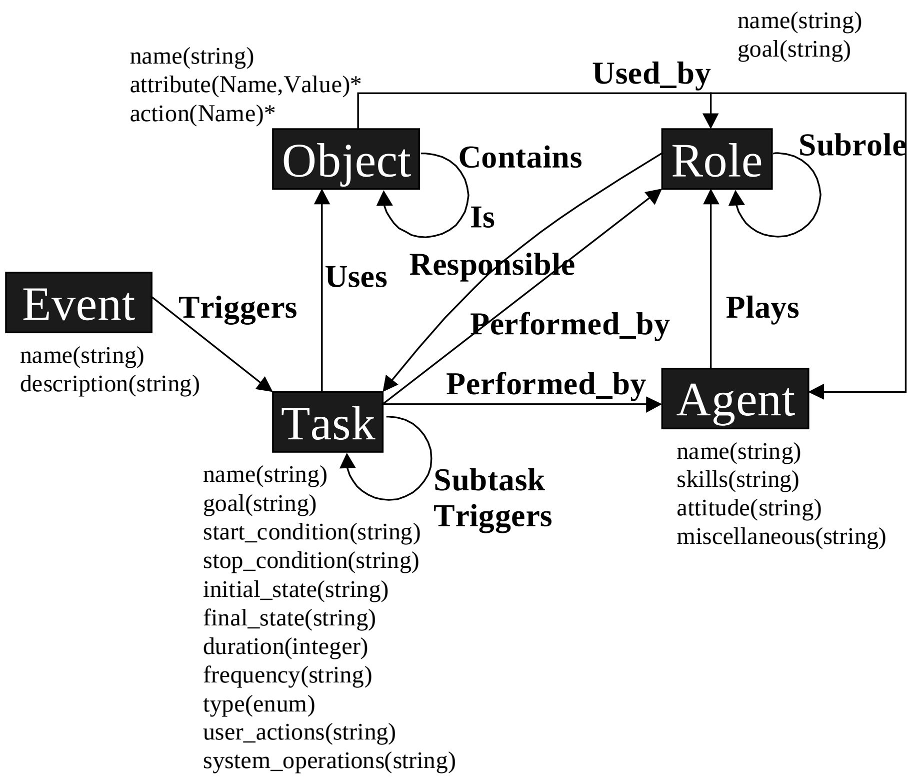


Fig. 2. The concepts and relations

are involved in a task. Such a relationship could be defined as the roles of the agents involved in the task and all the involved roles of the subtasks. The *involved_role* relationship is not part of the ontology because it can be defined using only the relationships of the ontology. The next section will show how all these relationships can be used in practice.

5 Supporting task analysis

Anyone who has ever done task analysis knows that even with the use of models it is still not an easy task. The biggest problem is often not in *applying* one technique[15], but in *communicating* results to a client or to other members of a design team. If the results of a task analysis are not communicated well their value for the design process rapidly diminishes. This section addresses this issue of how task analysis can be supported.

5.1 Formal models and tools

The ontology described in the previous section is not very formal and still leaves task details in the darkness. Many other task-modeling methods end up with quite formal models. Our experience with design situations, both in education and industry, shows that formal methods do **not** (by themselves) improve the communication between the members of a design team or with the client. Designers are very easily demotivated by pages of formal methods which they can not easily use. We think that task analysis tools (based on the models) are more appropriate for supporting communication, thereby hiding the complexities of the models and hiding parts of the models designers do not need to know about.

For task analysis to add value to the design process, its tools should be tailored to the designer. Usually a design team consists of many designers from different disciplines and backgrounds varying from psychology or computer science to anthropology or art. Each of them has specific needs and requires certain views on the task model. For instance some members may want just a quick overview of the task structures while others may need details on task execution conditions. However when multiple representations are being used they need to be consistent with each other. This consistency can be guaranteed when the representations are all based on the same model. Our ontology meets this requirement by abstracting from the visual representations and focussing on the underlying relationships between concepts, allowing multiple visual representations.

5.2 Graphical representations and the ontology

Currently we are developing a task analysis tool - EUTERPE- that uses the ontology as a basis for task analysis[1]. A Prolog representation of the ontology is used as a relational database from which several visual representations such as simple templates, lists, trees, or flow models are generated. Another possible view is a generated HTML document that serves as hypertext documentation of the task model which can only be viewed and not edited (yet). The views not only allow designers to inspect a task model but also allow modification and creation of new instances of tasks etc. which will automatically show up in other views as well. Figure 3 shows the basic structure of the tool.

One part of our research is to investigate which visual representations are useful to designers from different disciplines. One example of new visual representation is a

[1] The latest information about GTA and our tools can be found at: http://www.cs.vu.nl/~martijn/gta/index.html

kind of fish-eye view where you only see a part of all the information for instance about a certain task. Shifting the fish-eye to a related object shows all the information related to that object. By distinguishing between the underlying ontology and the graphical representations, views can be changed or added to without having to completely revise the underlying data or ontology. It is our goal to find views that are useful and to provide an environment in which these views are available to all members of the design team.

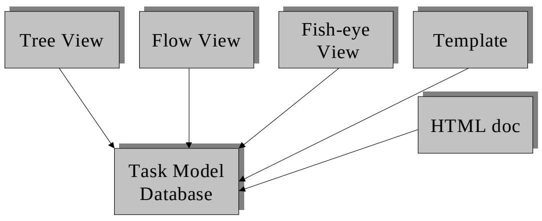


Fig. 3. Basic tool structure

5.3 Generating visual representations

Our tool EUTERPE uses the ontology directly to generate the visual representations. Consider a task tree. For editing and viewing the task tree we have program code that takes care of all the user interaction handling. To do this you need at least a tree data-structure that is created by querying the prolog engine. The algorithm looks like this:

```
procedure BuildTaskTree(startNode) {
    (1) ask the database for all subtasks of startNode (in Prolog: subtask(startNode,X))
    (2) add them as siblings of the startNode
    (3) call BuildTaskTree() for every sibling of startNode
}
```

This procedure recursively builds up a tree datastructure that is used in the tree drawing routines. In a similar way other views can be generated using the ontology. Consider the task flow representations of Figure 1. This task flow representation can be expressed using the triggering relationships as follows:

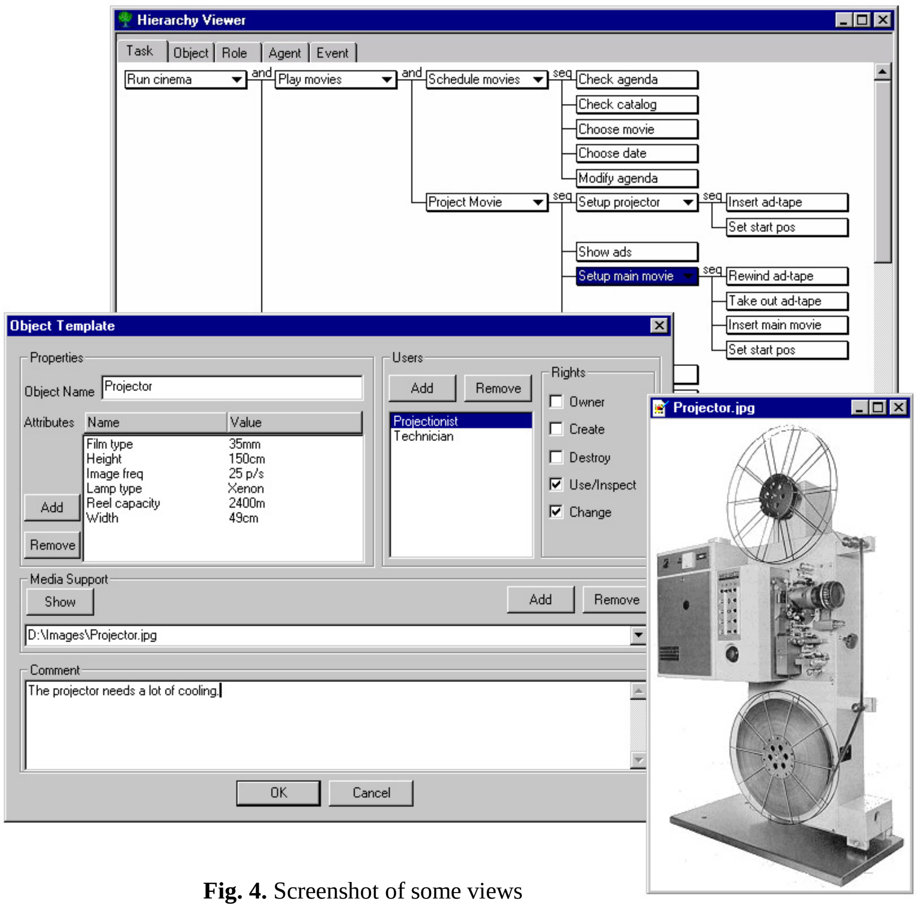
```
triggers("Call friend","Take Metro",OR),
triggers("Call friend","Walk",OR),
triggers("Take Metro","Buy Ticket",NEXT),
triggers("Walk","Buy Ticket",NEXT),
triggers("Buy Ticket","Enter Cinema",NEXT)
```

A task flow viewer can generate visual representations by asking for the specified triggering relationships. The algorithm looks like this:

```
procedure BuildFlowGraph(startNode) {
    (1) ask the database for all tasks triggered by startNode (in Prolog:
        triggers(startNode,X,Y))
    (2)  add them as nodes and add edges between them and the startNode
    (3)  call BuildFlowGraph() for every node that has an edge starting in startNode
}
```

Note that even the extra task "Go There" can be automatically detected in case of a task tree with constructors; move tasks to the next level if the trigger type is not NEXT and tasks with NEXT triggers exist in that level.

For the templates similar techniques are being used, complemented by



Object Name: Projector

Attributes

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

Users: Projectionist, Technician

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

D:\Images\Projector.jpg

Comment: The projector needs a lot of cooling.

Fig. 4. Screenshot of some views

straightforward querying of attributes. Besides the relationships from the ontology Euterpe also uses derived relationships such as *involved_role* that was mentioned earlier.

6 Operationalizing the Ontology

Figure 4 shows some views that are generated based on the ontology. The user of the tool does not need to be completely aware of the underlying ontology. For the user it can be just a useful aid in a task analysis. However the ontology is the Structuring principle of the tool and the user is implicitly offered a structured approach to task analysis.

The operationalized ontology in the tool also contains an extra attribute *media_object* in every concept. It allows designers to attaches any kind of media such as movies, audio files or images to a concept. In this way results of ethnographic studies are also immediately available to the design team.

7 Conclusions

In this paper, we have covered:

1. Ingredients common to most task models, including task decomposition, task flow, and task world modeling.
2. How task modeling relates to the design of user interfaces.
3. Our proposed new ontology for task analysis, allowing multiple visual representations to be generated from the same data.
4. A task analysis tool showing how this ontology can be used to support the design of user interfaces.

We believe that task models should be based on a clear underlying ontology that captures all relevant aspects of the task world, and allows multiple visual representations to be generated from it. We hope that this will facilitate the process of using task models to aid the design of user interfaces.

Acknowledgements

We wish to thank Melissa G. Schofield and Steve P. Guest for their enthusiastic help and invaluable suggestions.

References

1. Bevan, N.: *Ergonomic requirements for office work with VDTs*. part 11, ISO DIS 9241-11
2. Card, S.K., Moran T.P., Newell, A.: *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Erlbaum
3. Johnson, P., Johnson, H.: *Task Knowledge Structures: Psychological basis and integration into system design*. Acta Psychologica 78, pp 3-26, 1991
4. Johnson, P., Wilson, S., Markopoulos, P., Pycock, J.: *Adept - Advanced Design Environment for Prototyping with Task Models*. Proceedings InterCHI'93, Demonstration Abstract, Addison-Wesley, April 1993.
5. Johnson, P., Johnson, H., Waddington, R., Shouls, A.: *Task-Related Knowledge Structures: Analysis, Modelling and Application*. People and Computers IV 1988, Proceedings 4th British Computer Society HCI group
6. Kirwan, B., Ainsworth, L.K.: *A Guide to Task Analysis*. Taylor & Francis Ltd 1992
7. Mayhew, D.J.: *Principles and Guidelines in Software User Interface Design*. ISBN 0-13-721929-6, Prentice Hall PTR, New Jersey, 1992.

8. Markopoulos, P., Gikas, S.: *Towards a Formal Model for Extant Task Knowledge Representation*. In C. Stary (ed.), First Interdisciplinary Workshop on Cognitive Modelling and User Interface Development, Vienna, December 1994.

9. Paterno, F., Mancini, C., Meniconi, S.: *ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models*. Proceedings of Interact 97, 14-18 July 1997

10. Scapin, D., Pierret-Golbreich, C.: *Towards a method for task description: MAD.* Work with display units 89, Elsevier, Amsterdam

11. Sebillotte, S.: *Hierarchical planning as method for task analysis: The example of office task analysis*. Behaviour and Information Technology *7(3)*, 275-293, 1988

12. Szekely, P., Luo, P., Neches, R.: *Beyond Interface Builders: Model-Based Interface Tools*. In Proceedings of INTERCHI '93 April, 1993, pp. 383-390

13. Tauber, M.J.: *ETAG: Extended Task Action Grammar - A language for the description of the user's task language*. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel, Proceedings INTERACT '90, Amsterdam, Elsevier

14. van der Veer, G.C., Lenting, B.F., Bergevoet, B.A.J.: *GTA: Groupware Task Analysis - Modeling Complexity*. Acta Psychologica 91, 1996, pp. 297-322 Acta Paper

15. van der Veer, G.C., Mariani, M.: *Teaching Design of Complex Interactive Systems*. Learning by Interacting TeaDIS, Teaching Design of Interactive Systems, Schaerding, Austria, 20 - 23 May 1997

16. van der Veer, G.C., van Welie, M., Thorborg, D.: *Modeling Complex Processes in GTA*. Sixth European Conference on Cognitive Science Approaches to Process Control (CSAPC), pp. 87-91, Rome, Italy, 23-26 september 1997

17. Rex Hartson, H., Siochi, A.C., Hix, D.: *The UAN: a user-oriented representation for direct manipulation interface designs*. ACM Transactions on Information Systems Vol.8, No. 3 (July 1990), pp. 181-20