Task Models in Interactive Software Systems

FABIO PATERNO'
CNUCE—C.N.R.
Via V. Alfieri, 1
56010 Ghezzano, Pisa, Italy
fabio.paterno@cnuce.cnr.it

Task models are logical descriptions of the activities to be performed in reaching user's goals. They have shown to be useful for designing, analysing and evaluating interactive software applications. This chapter introduces the main concepts underlying task models and discusses how they can be represented and, then, used in the various phases of the life cycle.

Keywords: task models, user interfaces, interactive software systems

1. Introduction

Interest in design and development of interactive software applications has increased considerably over recent years. The underlying reason for this interest is the need to allow the greatest number of people access to software applications for the largest number of purposes and in the widest number of contexts. However, making systems easier to use implies taking into account many factors in the design of an interactive application, such as tasks to support, context of use, user preferences, media and interaction techniques available, and so on. It is thus important to have structured methods for allowing designers to manage such a complexity.

Despite the many direct manipulations tools currently available to designers to enable rapid building of user interfaces with graphical icons and multimedia effects, the design of interactive applications is still difficult and one of the main problems they have is to identify the interaction and presentation techniques more effective to support the possible tasks. On the other hand, end users often find interfaces difficult to understand and use in order to attain their desired ends. One of the main reasons for this is that in many cases users have trouble understanding what tasks are supported or how to associate the desired logical actions with physical actions of the user interface. Such problems could be tackled more systematically if designers had models, methods and tools to explicitly represent task models and provide indications about the most effective interaction and presentation techniques to support the possible user activities.

Task models describe how activities can be performed to reach the users' goals when interacting with the application considered. They should incorporate the

requirements foreseen by all those who should be taken into consideration when designing an interactive application (designers, software developers, application domain experts, end users, and managers). They are the meeting point where the various perspectives to be considered in designing interactive applications are combined.

Wide agreement on the importance of task models has been achieved because they capture what the possible intentions of the users are and describe logically the activities that should be performed to reach their goals. These models also allow designers to develop an integrated description of both functional and interactive aspects thus improving traditional software engineering approaches which mainly focused on functional aspects.

More precisely, task models can be useful for different purposes:

- *Understanding an application domain*: as they require a precise identification of the main activities and their relationships, they help to clarify many issues that at the beginning may not be immediately recognised;
- *Recording the results of interdisciplinary discussions*: many people can be involved in the design of an interactive application: user interface designers, software developers, managers, end users, experts of the application domain. It is thus important to have a representation of the activities that can integrate all the requirements raised and supports focusing on a logical level that can be understood by all of them;
- *Designing new applications consistent with the user conceptual model*: because of the lack of structured methods often design is completely based on ad-hoc rules or the ability of designers. If applications were designed following a task-based approach they would be usable and would incorporate the user requirements captured in the task model;
- *Analysing and evaluating usability of an interactive systems*: task models can be useful in various ways to support the usability evaluation of an interactive application. They have been used to predict the users' performance in reaching their goals or to support analysis of user behaviour to identify usability problems.
- *Supporting the user during a session*: creating a correspondence between tasks and the interaction objects composing the user interface can be useful also at run-time, for example to provide context-sensitive, task-oriented help systems [24].
- *Documenting interactive software,* the description of how logical activities are supported by the current application is also a documentation useful for users to learn how to use it and for developers, as an abstract description of the implementation.

2. Basic Concepts

Tasks are activities that have to be performed to reach a goal. They can be either logical activities such as *Retrieving information about the movies projected tonight* or physical activities such as *Selecting the button in the top left corner*.

A goal is either a desired modification of the state of an application or an attempt to retrieve some information from an application. For example, *Accessing a flight's*

database to know what flights are available is a goal which does not require the modification of the state of the application, whereas *Accessing a flight's database to add a new reservation* requires a modification of the state of the application.

It is evident that tasks and goals are closely connected. Each task can be associated with one goal, that is the goal achieved by performing the task. One goal can be achieved by performing one or multiple tasks. In some cases it is possible to choose among different tasks to achieve a certain goal.

We can distinguish between the task analysis and the task modelling phases. *The purpose of task analysis is to identify what the relevant tasks are*. Task analysis can be obtained using different techniques:

- interviews or workshops;
- questionnaires;
- observing users in their work place;
- considering how activities are performed in the current environment;
- considering existing documentation and training methods.

The results of this analysis can be used to develop various types of abstractions (scenarios, domain models, task models, properties, user and system models). This analysis may have very different scopes, depending on the focus of interest. It can range from activity of a single user at work in a given environment, involving computer-supported activities as well as other types of activities, to the processes occurring in the whole environment, potentially involving several co-workers.

The result of task analysis is an informal list of tasks relevant for the application domain considered which can be supplemented with indications of possible problems, task attributes, and user preferences. The activities effectively performed by a user are sometimes complex processes which are difficult to understand fully or predict: an activity often results from a compromise between conflicting goals; goal-directed processes may be modified by opportunistic processes, the occurrence of particular conditions, exceptions, etc. It is thus important in the task analysis and modelling phases to be able to capture the main elements of such flexibility.

The task modelling phase occurs after the task analysis phase. *The purpose of task modelling is to build a model which describes precisely the relationships among the various tasks identified*. These relationships can be of various types, such as temporal and semantic relationships. The task model is usually structured in such a way as to address various logical levels. When we reach tasks which cannot be further decomposed we have basic tasks. In some cases basic tasks require one single physical action to be performed. The level of decomposition to reach in task modelling depends on its purpose. It is important that task models are rich in information and flexible so as to capture all the main activities that should be performed to reach the desired goals and the different ways to accomplish them. In terms of software engineering phases, task analysis can be particularly useful to support the requirements phase whereas task models can be mainly useful in the design and evaluation phases.

Various types of task models can be considered:

- *the system task model*, describing how the current system implementation assumes that tasks should be performed; system task models are developed when people have to understand how a system works or in usability evaluation where the purpose is to evaluate how well the system task model supports users;
- *the envisioned task model*, where designers think about a new system and how users should interact with them; usually these models are not very refined because some more-implementation dependent aspects cannot be completely defined and are used to propose or prescribe new design solutions;
- *the user task model*, how users think that tasks should be performed in order to reach their goals. Depending on the user, this task model may not be particularly structured. Most usability problems arise when large discrepancy exists between the user and system task models.

Task models are not the only useful representation when designing interactive applications. In addition, they are sometimes hard to develop from scratch. When approaching the design of a new application or the re-design of an existing application, designers have often a lot of informal information available: documentation concerning existing applications, notes from meetings with users, requirements provided by customers, and so on. They have to refine this material to identify the task structure underlying the existing application to analyse or that corresponding to the new application to design. Scenarios are a well known technique often used during the initial informal analysis phase. They provide informal descriptions of a specific use in a specific context of an application. A careful identification of a set of meaningful scenarios allows designers to obtain a description of most of the activities that should be considered in a task model and task models can be used to identify scenarios. More generally, the main differences between a task model and a scenario are:

- a scenario indicates only one specific sequence of occurrences of the possible activities while the task model should indicate a wide set of activities and the related temporal relationships;
- a scenario contains some detailed information that usually are not considered in abstractions such as task models.

3. Main Approaches to Task Modelling

A number of approaches to task modelling have been developed and it is not possible to mention all of them. In this section some approaches, particularly interesting, are introduced and discussed.

3.1 Hierarchical Task Analysis

The first works on HTA (Hierarchical Task Analysis) date back to the late sixties [1]. The basic idea, to describe the set of activities to be considered logically structured in different levels, has proved to be successful, as can be seen from its application in a number of projects. However, this approach describes how the activities are related to each other in a rather rudimentary way. The representation is usually given including task names in boxes with numbers indicating the order of performance.

3.2 GOMS family

GOMS (Goals, Operators, Methods, Selection rules) [10] was the first systematic approach to the design of user interfaces. It is a method, originally introduced by Stuard Card, Thomas Moran, and Allen Newell, that has a long history and considerable influence. It is based on a cognitive model (the Human Processor Model) which is described by a set of memories and processors and a set of principles underlying their behaviour. More precisely, it is decomposed into three subsystems interacting with each other (perceptive, motor and cognitive subsystems).

GOMS provides a hierarchical description to reach goals in terms of operators. Operators are elementary perceptual, motor and cognitive acts. Actions at one level can be goals at a lower level. Methods are sequences of subgoals and operators used to structure the description of how to reach a given goal. The selection rules indicate when to use a method instead of another one. An example is when moving the cursor in a specific location of a document. If the desired position is close to the current one then it is sufficient to move the cursor by arrow keystrokes, otherwise it would be better to select the new position with the mouse support.

This notation is especially valid to describe the performance of tasks and it has been used for a number of industrial applications.

In Table 1 there is an example of a GOMS specification. It describes how a user segments the larger goal of *Access ATM* into a sequence of small, discrete operators, such as *Select withdraw service* or *Select amount of money*. In this example no selection rule is used.

GOAL: ACCESS ATM
GOAL: ENABLE ACCESS
INSERT CREDIT CARD
INSERT PASSWORD
GOAL: TAKE CASH
SELECT WITHDRAW SERVICE
SELECT AMOUNT OF MONEY
PRESS OKAY
TAKE MONEY
VERIFY AMOUNT OF MONEY

Table 1. Example of a GOMS specification

More generally, there are several different versions of GOMS in use today. In the first proposal there was both a description of how to express a goal and subgoals in a hierarchy, methods and operators, and how to formulate selection rules and a simplified version of GOMS called Keystroke-Level Model (KLM). KLM uses only keystroke-level operators, no goals, methods or selection rules. The analysis simply lists the keystrokes, mouse-movements, and mouse-button presses that a user must perform to accomplish a task, then uses a few simple heuristics to place a single type of coarse "mental operator" which approximates many kinds of internal cognitive actions. KLM models are easier to construct, but classic GOMS models provide more information for the qualitative design. NGOMSL includes a more rigorous set of rules for identifying the GOMS components and information such as the number of steps in a method, how goals

are set and terminated, what information needs to be remembered while performing the task [19].

One limitation of GOMS approaches is that it considers error-free behaviour and only sequential tasks. The latter limitation is partially overcome by one of the extensions [18] of the GOMS approach that has been developed, CPM-GOMS [15]. In CPM-GOMS separate sequences are constructed for each category of operators (cognitive, perceptual, and motor operators are considered). Each instance of operator is represented as a box with a name centred in it and the associated duration in milliseconds above the top right corner. Dependencies between activities are represented as lines connecting the boxes. For example, telephone operators helping customers cannot press the collect-call key until they hear the customer request a collect call. Therefore, there would be a dependency line drawn between a box representing the perception of the word *collect* and boxes representing cognitive operators that verify the word *collect* and initiate pressing the collect-call key. The boxes and their dependency lines are represented in a PERT chart, a common tool used in project management. The diagrams have one row for each type of operator. Such operators are classified according to the related human resource (visual, aural, internal cognitive, left hand, right hand, verbal, eye movement). Also system response time is considered. The box associated with each instance of operator is thus located in the row of the corresponding type, the position depends on when the action should be performed whereas the size is proportional to the length of the action. The overall diagram allows designers to easily identify the activities that can be performed in parallel. An important concept in analysing the total task time for complex parallel tasks is the critical path. The critical path is the sequence of activities that takes the longest time and determines the total time for the entire task. However, in CPM-GOMS operators for representing flexible temporal relationships (such as dynamic disabling of activities) are not provided.

A problem in predicting time performance with GOMS-based approaches is that when distributed applications are considered (such as Web-based applications) the time requested by the application to respond to the user interactions is difficult to predict because it can depend on unpredictable external factors (such as networks delays). Detailed discussion of the respective strengths and weakness of the various versions in the GOMS family can be found in [18].

3.3 UAN

One common aspect between HTA and GOMS approaches is the logical hierarchical structure. A similar structure can be found in grammar based approaches to the specification of task models such as in TAG (Task Action Grammar) [30] and ETAG (Extended Task Action Grammar) [35]. In these grammars the logical structure is defined in the production rules where high level tasks can be associated with non terminal symbols whereas basic tasks are associated with the terminal symbols of the grammar.

UAN (User Action Notation) [16] has been another successful approach, still supporting this type of logical structure. The main purpose of UAN, a notation developed by Rex Hartson and others, is to communicate design. It allows designers to describe the dynamic behaviour of graphical user interface. It combines concepts from task models, process-based notations and user actions descriptions. It is a textual

notation where the interface is represented as a quasi-hierarchical structure of asynchronous tasks, the sequencing within each task being independent of that in the others. A rich set of operators to describe temporal relationships among tasks is available.

A UAN specification is usually structured into two parts:

- one part describes task decomposition and the temporal relationships among asynchronous tasks;
- the other part associates each basic task with one table. These tables have three columns indicating the user actions, the system feedback and the state modifications requested to perform it. Some specific symbols are used in these specifications, for example to indicate button pressing or releasing.

An example of specification of a task in terms of its subtasks and their temporal relationships is:

Task: Access ATM:
Enabling Access (Withdrawing Cash | Depositing Cash | Get Information)+

This expression describes that first the *Enabling Access* task and then one or more occurrences (+ operator) of the tasks composing the expression *(Withdrawing Cash| Depositing Cash| Get Information)* should be performed (the | operator means choice).

In Table 2 we have an example of a description of a basic task. The user has to select the withdraw button. In UAN the expression ~[object] means that the user moves the cursor to some arbitrary point on the object indicated. In our case the user moves the finger on the *Withdraw* button. The selection performed by pressing and releasing an item is represented by the V^ symbol. Then the interface reacts by displaying the possible amounts of money to withdraw and the state of the system changes because the currently selected service becomes withdraw money. Next, the user can select one specific amount of money and the interface provides the amount requested changing the amount still available in the bank account accordingly. As can been seen, the UAN specifications should be read sequentially left-to-right and then vertically (top to bottom from line to line) when there are the tables associated with basic tasks.

Task: Withdrawing Cash

User Action	Interface Feedback	Interface State
~[Withdraw] V^ ~[Amount] V^	Display (Possible amounts) Provide (AmountCash)	CurrentService=Withdraw Account=Account-Amount

Table 2. An example of a UAN specification

The UAN notation is suitable to specifying tasks, which are the components of the specification. It also provides good support for specifying low level actions sequencing. One of the possible limitations is that it can lead to large specifications sometimes with many details not always useful for the designer (for example, often it is not important to

specify all the elementary feedback provided by the user interface, for instance when the cursor is on a button then its colour changes slightly). Another limitation is that the order which has to be followed to interpret the tables of the basic tasks (left-to-right) can be rigid and inadequate. For example, when the modification of the state of the application triggers the performance of the basic task considered, it means that it is an item on the far right column that triggers events described by the other columns.

3.4 ConcurTaskTrees

The ConcurTaskTrees notation was extensively introduced in [26]. It is a notation aiming at supporting engineering approaches to task modelling. It provides a rich set of operators to describe the temporal relationships among tasks (enabling, concurrency, disabling, interruption, optionality). In addition, for each task further information can be given such as its type, the category (indicating how the performance is allocated), the objects that it requires to be manipulated and attributes, such as frequency of performance.

ConcurTaskTrees was developed after first studies [25] aimed at specifying graphical user interfaces by using the LOTOS notation [6]. LOTOS is a concurrent formal notation that seemed a good choice to specify user interfaces because it allows designers to describe both event-driven behaviours and state modifications. Markopoulos [22] provides an example of how this notation has been used to specify task models. However, LOTOS suffers from some limitations that make it unlikely to be widely used in the human-computer interaction domain. It was soon realised that there was a need for new operators to express a richer set of dynamic behaviours in human-computer interactions in a compact way and additional information useful in analysing and representing task models. Moreover, LOTOS has a textual syntax that can easily generate complex expressions even when the behaviour to describe is quite simple. Thus, a new notation was developed, ConcurTaskTrees. Its main aim is to be an easy-to-use notation that can support the design of real industrial applications, which usually means medium to large sized applications.

The main features of ConcurTaskTrees are:

- *Focus on activities:* it allows designers to concentrate on the activities that users aim to perform, that are the most relevant aspects when designing interactive applications that encompass both user and system-related aspects avoiding low-level implementation details that at the design stage would only obscure the decisions to take.
- *Hierarchical structure*: a hierarchical structure appears quite intuitive, in fact often when people have to solve a problem they tend to decompose it into smaller problems still maintaining the relationships among the various parts of the solution. The hierarchical structure of this specification has two advantages: it provides a wide range of granularity, allowing large and small task structures to be reused, and it enables reusable task structures to be defined at both low and high semantic levels.
- *Graphical syntax*: a graphical syntax often (though not always) is more easy to interpret, in this case it reflects the logical structure so it has a tree-like form.
- *Concurrent notation:* a rich set of possible temporal relationships among the tasks can be defined. This sort of aspect is usually implicit, expressed informally

in the output of task analysis. Making the analyst use these operators is a substantial change to normal practice. The reason for this innovation is that after an informal task analysis we want designers to express clearly the logical temporal relationships. This is because such ordering should be taken into account in the user interface implementation to allow the user to perform at any time the tasks that should be active from a semantic point of view.

- *Task allocation,* how the performance of the task is allocated is indicated by the related category and it is explicitly represented by using icons. There are four possibilities: user task (only internal cognitive activity such as selecting a strategy to solve a problem); application task (only system performance such as generating the results of a query); interaction task (user actions with possibility of immediate system feedback, such as editing a diagram), abstract tasks (tasks that have subtasks belonging to different categories).
- *Objects,* once the tasks are identified it is important to indicate the objects that have to be manipulated to support their performance. Two broad types of objects can be considered: the user interface objects and the application domain objects. Multiple user interface objects can be associated to a domain object (for example, temperature can be represented by a bar-chart of a textual value).

Still regarding the tasks associated with withdrawing money from an ATM, Figure 1 presents an excerpt from the relative ConcurTaskTrees model. First there is the *EnableAccess* task composed of three sequential subtasks (>> is the enabling operator). The *RequirePassword* task is performed by the system as indicated by the computer icon. The *Access* task is an iterative task (indicated by the * symbol) composed of the choice among three subtasks (*WithDrawCash*, *DepositCash*, *GetInformation*). *WithDrawCash* is composed of a sequence of tasks ([] >> is the enabling with information passing operator). It also has a user subtask (indicated by a different icon): the *DecideAmount* task, describing the internal user cognitive activity associated with deciding the amount to withdraw. For sake of brevity, Figure 1 does not show the decomposition of *CloseAccess*, *DepositCash* and *GetInformation* tasks.

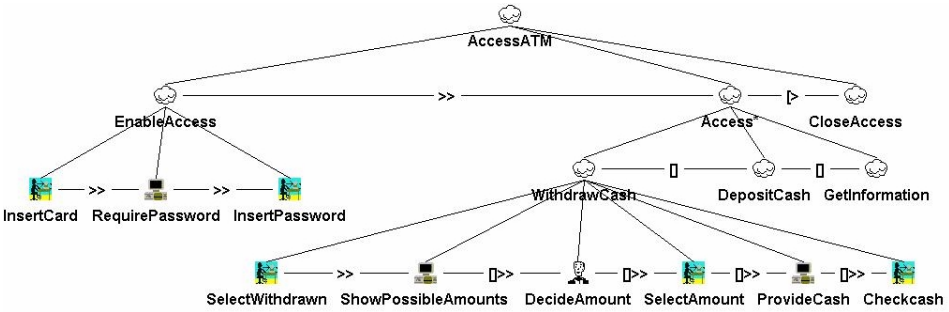

Fig. 1. Example of task model in ConcurTaskTrees

Providing support for cooperative applications is important because the increasing availability and improvement of Internet connections makes it possible to use many types of cooperative applications. In ConcurTaskTrees, when there are cooperative applications the task model is composed of various parts. A role is identified by a specific set of tasks and relationships among them. Thus, there is one task model for

each role involved. In addition, there is a cooperative part whose purpose is to indicate the relationships among tasks performed by different users.

The cooperative part is described in a manner similar to the single user parts: it is a hierarchical structure with indications of the temporal operators. The main difference is that it includes cooperative tasks: those tasks that imply actions by two or more users in order to be performed. For example, negotiating a price is a cooperative task because it requires actions from both a customer and a salesman. Cooperative tasks are represented by a specific icon depictng two people interacting with each other.

In the cooperative part, cooperative tasks are decomposed until we reach tasks performed by a single user that are represented with the icons used in the single user parts. These single user tasks will also appear in the task model of the associated role. They are defined as *connection tasks* between the single-user parts and the cooperative part. In the task specification of a role, (see for example Figure 2, top part) we can identify connection tasks because they are annotated by a double arrow under their names.

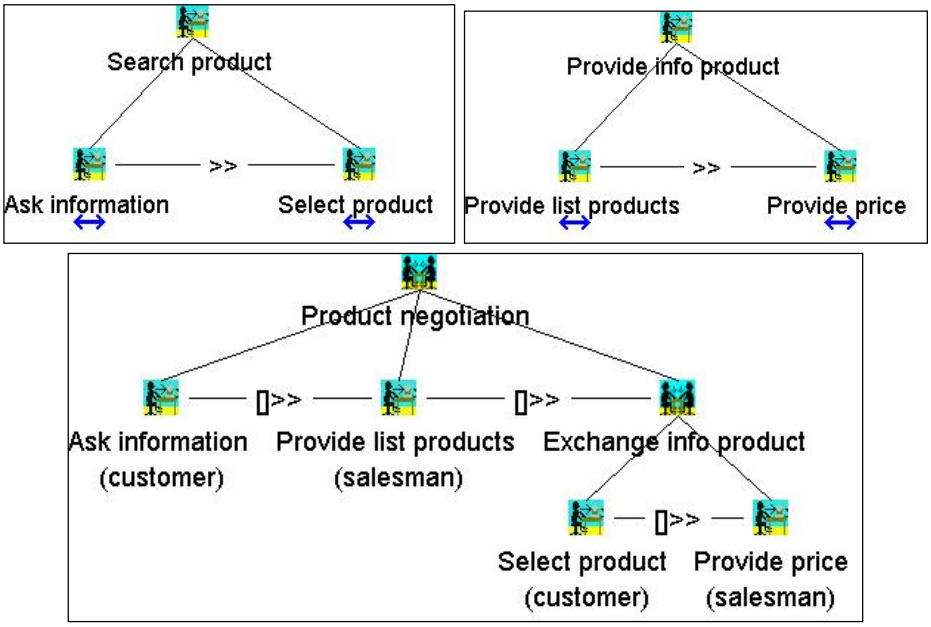

Fig. 2. Simplified example of cooperative task model

The effect of this structure of representation is that in order to understand whether a task is enabled to be performed we have to check both the constraints in the relative single user part and the constraints in the cooperative part. It may happen that a task without constraints regarding the single user part is not enabled because there is a constraint in the cooperative part indicating another task must first be performed by another user. If we consider the example in Figure 2 we can see the *Search Product* task performed by a Customer and the *Provide Info Product* task performed by a salesman. If we consider each part of the task model in isolation, these two tasks can be started immediately. However, if we consider the additional constraint indicated in the part

below of the figure we can see that the *Provide list products* task (by the salesman) needs to wait first the performance of the *Ask information* task (by the customer) in order to be enabled.

4. Engineering Task Models

Despite task models have long been considered, only recently developers and designers have realised the importance of engineering approaches to task models. An engineering approach should address at least four main issues:

- availability of *flexible and expressive notations* able to describe clearly the possible activities; it is important that these notations are sufficiently powerful to describe interactive and dynamic behaviours; such notations should be readable so that they can be interpreted also by people with low formal background;
- need for *systematic methods* to support the specification, analysis, and use of task models, to facilitate their development and support designers in using the knowledge that they incorporate to address the design of the user interface and its evaluation; we note that often even designers who developed some task analysis and modelling did not use them for the detailed design of the user interface because of this lack of structured methods which should give rules and suggestions about how to use information in the task model for the concrete design.
- support for the *reuse* of good design solutions to problems which occur across many applications (using for example task patterns [26]); this is relevant especially in an industrial context where developers often have to design applications which address similar problems, thus it would be useful to have design solutions structured and documented in such a way as to support easy reuse and tailoring in different applications.
- availability of *automatic tools* to support the various phases of the design cycle, including usability evaluation; for example, once structured methods for task-based design have been identified it is possible to incorporate their rules in automatic tools which can support designers giving easy to interpret representations of useful information and suggestions of possible solutions, still leaving the possibility of tailoring the criteria to the designer.

Task models have been used in various approaches and have been used to support different phases of the design cycle:

- *Requirement analysis*, where through a task analysis designers identify requirements that should be satisfied in order to perform tasks effectively (GTA [36]).
- *Design of interactive applications* (Adept [38], Trident [5], Mobile [32]), in this case the goal is to use information contained in logical models to identify the interaction and presentation techniques best suited to support the tasks at hand and the dialogues whose temporal constraints are closest to those of the model;
- *Usability evaluation*, it can be supported in various ways: for example, identifying user efficiency in performing the tasks (such as in the KLM [10]

approach) or analysing logs of user interactions with the support of task models (see for example RemUSINE [20]).

5. Use of Task Models in Design of User Interfaces

A user interface is composed of two main components: the presentation, indicating how the user interface provides information to the user, and the dialogue, describing how the actions that users and system perform can be sequenced. Often the dialogue model is part of the task model and it corresponds to the most refined level. Task models can be useful for designing both aspects. The temporal relationships among tasks can be useful to identify how the dialogue should be implemented whereas the type of tasks considered is useful to select the presentation and interaction techniques more suitable.

Since design is a complex activity it is preferable to avoid applying too rigid rules in task-based design. Rather, a set of criteria can be identified leaving the choice of what criteria should be used to the designer who may need to tailor them to the specific case study under consideration.

During a session the presentations associated with an application can change depending on user and system generated actions. Each presentation is an application-generated user interface whose elements are perceivable at the same time (this means, in a graphical user interface, that the elements are all on the screen at the same time). By analysing the temporal relationships of a task model it is possible to identify the enabled task sets. Each set is composed of tasks that are enabled over the same period of time according to the constraints indicated in the model. Thus, the interaction techniques supporting the tasks belonging to the same enable task set should often be part of the same presentation.

In general, when considering the tasks supported by an application there is a wide variety of possible choices in terms of the number of presentations supporting them. These choices vary from a single presentation supporting all tasks (this is possible with small, limited applications) to multiple presentations, whose maximum meaningful number is given by the number of enabled task sets. Indeed, if there are more presentations than such sets, this implies that there are tasks belonging to the same set that are being supported separately by different presentations. This means that a sequential constraint among some of them has been introduced even though it was not indicated in the task model and, thus, was not logically motivated.

Various approaches have been proposed to derive concrete user interfaces from task models. A number of criteria can be identified for this purpose. For example:

- the logical decomposition of tasks can be reflected in the presentation by explicitly grouping interaction techniques associated with tasks that share the same parent task.
- sequential tasks can be supported in various modalities such as: separate presentations for each task rendered at different times; all the interaction techniques corresponding to the sequential tasks rendered in the same presentation (this is useful when the tasks are closely linked either because the sequence of tasks is performed multiple times iteratively or because the tasks

exchange information); and lastly, through separate windows for each task (still active over the same period of time but which can be iconified if they take up too much space on the screen).

- the type of task, the type of objects manipulated and their cardinality is another useful element. For example, if we have a selection task then we can immediately delimit the choice of the corresponding interaction technique to use to those supporting selection, in addition we can further limit the possibility of choice depending on the type of objects that can be selected, and, finally, if we know what the data cardinality is we can find only a few suitable interaction techniques, for example for choice among low cardinality values we can select a radio-button.
- disabling tasks are tasks that interrupt other activities, in some cases to activate new activities. They can be represented in the same manner (for example buttons with specific graphical attributes) and located in the same part of the layout (for example the right bottom part of the screen) for consistency.

6 Use of Task Models in Usability Evaluation

A number of usability evaluation techniques have been developed in recent years. They can be classified according to many dimensions: for example, the level of refinement of the user interface considered or the amount of user involvement in the evaluation. More generally, usability evaluation methods can be classified as:

- model-based approaches, where a significant model related to the interactive application is used to drive the evaluation;
- inspection-based assessment, where some expert evaluates the system or some representation of it, according to a set of criteria;
- empirical testing, where direct use of the system is considered.

Task models have often been used [18] to produce *quantitative predictions* of how well users will be able to perform tasks with a proposed design. Usually the designer starts with an initial task analysis and a proposed first interface design. The designer should then use an engineering model (like GOMS) to find the applicable usability problems of the interface.

One important issue has been to find a method that allows designers to apply meaningful models to some empirical information. An attempt in this direction was USAGE [9] that provided tool support to a method where the user actions required to execute an application action in UIDE [13] are analysed by the NGOMSL approach (one of the GOMS family). However this information is still limited with respect to that contained in the logs of the user actions performed during work sessions by users. More interesting results can be obtained using task models and empirical information. In [20] there is a description of how to use task models to support usability evaluation. The possible activities supported by the application described by the task model are used to analyse real user behaviour as revealed by log files obtained through automatic recording of user interactions with a graphical interface. This type of evaluation is useful for evaluating final versions of applications. Other approaches that share similar goals are described in [30].

However, task models can also be useful to support evaluation in the early phases of the design cycle when inspection-based methods are often applied to evaluate prototypes. These methods are less expensive than empirical testing. One of their advantages is that their application at least decreases the number of usability problems that should be detected by the final empirical testing. A number of inspection based evaluation methods have been proposed. Some of them do not use task models, such as heuristic evaluation [23], where a set of general evaluation criteria (such as visibility of the state of the system, consistency, avoid to provide useless information, ...) are considered and evaluators have to check whether they have been correctly applied. This method heavily depends on the ability of the evaluator. In cognitive walkthrough [31] more attention is paid to the notion of task, because the evaluators have to identify a sequence of tasks to perform and for each of them four questions are asked: will users understand what they should do? Will users identify the interaction techniques able to support the performance of the task? Will users be able to correctly associate the interaction technique with the task to perform? After the interaction will users receive feedback from the system? While this method is clear and can be applied with limited effort, it has a strong limitation: it tends to concentrate the attention of the evaluator on whether or not the user will be able to perform the right interactions, little attention is paid on what happens if users perform wrong interactions (interactions that are not useful for performing the current task) and on the consequences of such wrong interactions.

Methods with more systematic use of task models have been developed [12] to help designers analyse what happens if there are deviations in task performance with respect to what was originally planned. It indicates a set of predefined classes of deviations that are identified by *guidewords*. In the analysis, the system tasks model is considered: how the design of the system to evaluate assumes that tasks should be performed. The goal is to identify the possible deviations from this plan that can occur. Interpreting the guidewords in relation to a task allows the analyst to systematically generate ways the task could potentially go wrong during its performance. This then serves as a starting point for further discussion and investigation. Such analysis should generate suggestions about how to guard against deviations, as well as recommendations as to user interface designs that might either reduce the likelihood of deviations or support detecting and recovering from them. In the analysis, first basic tasks are considered. Then, evaluators analyse high level tasks, each of them identifies a *group of subtasks* and, consequently, it is possible to analyse deviations that involve more than one basic task. Such deviations concern whether the appropriate tasks are performed and if such tasks are accomplished following a correct ordering. It is important that the analysis of the deviations is carried out by interdisciplinary groups where such deviations are considered from different viewpoints and background in order to carry out a complete analysis. Since the analysis follows a bottom-up approach (first basic tasks and then high-level tasks), it allows designers first to focus on concrete aspects and then to widen the analysis to consider more logical steps. The types of deviations considered concern what happens if the task is not performed or it is performed wrongly or it is performed at the wrong time.

7. Tools for Task Models and Task-based Design

One of the main problems in task modelling is that it is a time-consuming, sometimes discouraging, process. To overcome such a limitation, interest has been

increasing in the use of tool support [7]. Despite this wide-spread interest, tools developed for task models have been rather rudimentary, mainly research tools used only by the groups that developed them. More systematically engineered tool-support is strongly required in order to ease the development and analysis of task models and make them acceptable to a large number of designers. If we consider the first generation of tools for task models we can notice that they are mainly research prototypes aiming at supporting the editing of the task model and the analysis of the task relationships and related information. This has been a useful contribution but further automatic support is required to make the use of task models acceptable to a large number of designers. For example, the tool support provided for GOMS approaches is still far from what designers and evaluators expect from GOMS tools [4] because of limited pragmatics, partial coverage of the GOMS concepts, limited support for cognitively plausible models and lack of integration in the development process.

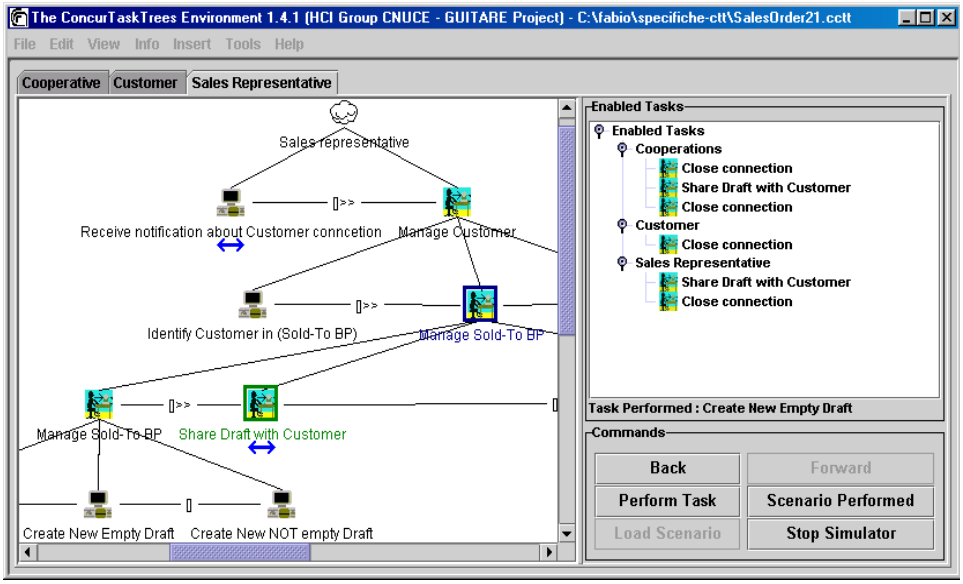

Fig. 3. The interactive simulator of task models

Thus, there is a strong need for engineered tools able to support task modelling of applications, including cooperative and multi-user ones which are on the rise, and the analysis of such models' contents which may can be complex in the case of many real applications. Euterpe [37] is an example of tool for groupware task analysis based on an ontology for task models. CTTE is an engineered tool, which supports a number of possibilities, for example automatic support to improve the layout, modify the specification, furnish information on the structure of the task model, and check whether the specification is complete. The tool is publicly available at http://giove.cnuce.cnr.it/ctte.html. CTTE also supports interactive simulation of the behaviour of the task model. The tool shows the list of enabled tasks (see Figure 3, right side) and the designer can select one of them and ask the tool to display the next enabled tasks if the selected one is performed. It is possible to identify an abstract scenario by selecting a list of tasks. The scenario can be displayed by selecting the *Scenario*

Performed button and can also be saved and loaded later on even from a different task model in order to check if it can still be performed.

These features are useful for designers to understand whether the dynamic behaviour described by the model corresponds to the desired behaviour. It also serves as interactive documentation of the application in question for end users.

To facilitate the development of task models a number of tools have been developed by various research groups: CRITIQUE [17] helps designers obtain KLM models from user action logs, U-TEL [34] is a knowledge elicitation system supporting the development of interface data and task structures from informal descriptions that aims mainly to identify nouns and verbs and associate them with objects and tasks in the models, and an approach to support the use of scenarios to derive ConcurTaskTrees task models is given in [28].

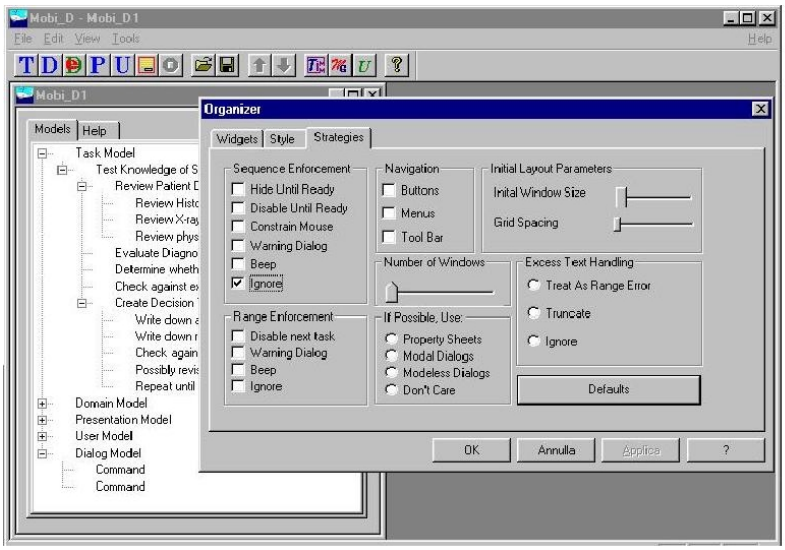

Fig. 4. The Mobi-D support for model-based design

A number of automatic environments supporting user interface design and generation has also been proposed. A review of these approaches is in [33]. Examples of such environments are ADEPT [38], TRIDENT [5] and MOBI-D [32]. The last one is a particularly comprehensive set of tools developed because it provides a set of model editors to create relations between abstract and concrete elements, and a layout tool that can be reconfigured to reflect the decisions made at previous stages in the design process. As can be seen in Figure 4, Mobi-D supports the development of a number of models; designers can then specify various parameters to define how the information in such models can be used to design the user interface. This enables designers to tailor general design criteria to the specific application considered.

8. Conclusions

Task models are entered in the current practise of design of interactive software systems. We have discussed how such models can be represented and used for analysing an interactive application and supporting both user interface design and usability evaluation.

They can effectively complement the support given by object-oriented approaches, such as UML [8], in interactive software system design. UML is more suitable to model the internal part of an interactive software system whereas task models are more effective to indicate the activities to support and corresponding user interface interactions.

Recent years have seen the development of first engineering approaches to task models that make them suitable to be used in software companies and not only by researchers in the field.

References

1. Annett, J., Duncan, K.D., *Task Analysis and Training Design*, Occupational Psychology, 41, pp.211-221, 1967.
2 Barclay, P., Griffiths, T., McKirfy, J., Paton, N., Cooper, R., Kennedy, J., *The Teallach Tool: Using Models for Flexible User Interface Design*, Proceedings CADUI'99, pp.139-158, Kluwer Academic Publisher, 1999.
3 Bastide, R., Palanque, P., *A Visual and Formal Glue between Application and Interaction*, International Journal of Visual Language and Computing, Academic Press Volume 10, Number 6, 1999.
4 Baumeister L., John B., Byrne M., *A Comparison of Tools for Building GOMS Models*, Proceedings CHI'2000, pp.502-509, ACM Press, 2000.
5 Bodart F., Hennerbert A., Leheureux J., Vanderdonckt J., *A Model-based approach to Presentation: A Continuum from Task Analysis to Prototype*, in Proceedings DSV-IS'94, Springer Verlag, pp.77-94, 1994.
6 Bolognesi T. and Brinksma E., *Introduction to the ISO Specification Language LOTOS*, In Computer Network ISDN Systems 14(1), 1987.
7 Bomsdorf B., Szwillus G., *Tool Support for Task-Based User Interface Design*, Proceedings CHI'99, Extended Abstracts, pp.169-170, 1999.
8 Booch, G., Rumbaugh, J., Jacobson, I., *Unified Modeling Language Reference Manual*, Addison Wesley, 1999
9 Byrne, M., Wood. S., Noi Sukaviriya, P., Foley, J., Kieras, D., *Automating Interface Evaluation*, Proceedings CHI'94, pp. 232-237, ACM Press, 1994.
10 Card, S., Moran, T., Newell, A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, 1983.
11 Eisenstein, J., Puerta, A., *Adaptation in Automated User Interface Design*, Proceedings IUI 2000, ACM Press, pp.74-81, 2000.
12 Fields, R., Paternò, F., Santoro, C., Tahmassebi, T., *Comparing Design Options for Allocating Communication Media in Cooperative Safety-Critical Contexts: a Method and a Case Study*, ACM Transactions on Computer-Human Interaction, Vol.6, N.4, pp.370-398, December 1999.
13 Foley, J., Sukaviriya, N., *History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Development*, in F. Paterno' (ed.) Interactive Systems: Design, Specification, Verification, pp. 3-14 Springer Verlag, 1994.

14 Garavel, H., Fernandez, J., Kerbrat, A., Mateescu, R., Mounier, L., Sighireanu, M., *CADP (C{\AE}SAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox*, Proceedings of the 8th Conference on Computer-Aided Verification, LNCS 1102, pp.437--440, 1996.
15 Gray, W., John, B., Atwood, M., *Project Ernestine: A Validation of GOMS for Prediction and Explanation of Real-World Task Performance*, Human-Computer Interaction, 8, 3, pp. 207-209, 1992.
16 Hartson R., Gray P., *Temporal Aspects of Tasks in the User Action Notation*, Human Computer Interaction, Vol.7, pp.1-45, 1992.
17 Hudson S., John B., Knudsen K., Byrne M., *A Tool for Creating Predictive Performance Models from User Interface Demonstrations,* Proceedings UIST'2000, pp.93-102, ACM Press, 2000.
18 John, B., Kieras, D., *The GOMS Family of Analysis Techniques: Comparison and Contrast*. ACM Transactions on Computer-Human Interaction, Vol.3, N.4, pp.320-351, 1996.
19 Kieras D.E., *Guide to GOMS model usability evaluation using NGOMSL*, in The Handbook of Human-Computer Interaction, 2nd edition, North Holland 1996.
20 Lecerof, A., Paternò, F., *Automatic Support for Usability Evaluation*, IEEE Transactions on Software Engineering, October, pp. 863-888, IEEE Press, 1998.
21 Limbourg, Q., Ait El Hadj, B., Vanderdonckt, J., Keymolen, G., Mbaki, E., *Towards Derivation of Presentation and Dialogue from Models: Preliminary Results*, Proceedings DSV-IS 2000, Springer Verlag, 2000.
22 Markopoulos, P, Gikas, S., *Formal Specification of a Task Model and Implications for Interface Design*, Cognitive Systems 4-3, 4, pp.289-310, 1997.
22 Nielsen, J., *Usability Engineering*, Academic Press, 1993.
24 Pangoli, S., Paternò, F., *Automatic Generation of Task-oriented Help*, Proceedings UIST'95, pp. 181-187, ACM Press, 1995.
25 Paternò, F., Faconti, G. *On the Use of LOTOS to Describe Graphical Interaction*, Proceedings HCI'92, pp.155-173, Cambridge University Press, 1992.
26 Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*. Springer Verlag, ISBN 1-85233-155-0, 1999.
27 Paternò, F., Mancini, C., *Designing Usable Hypermedia*, Empirical Software Engineering, Vol.4, N.1, pp.11-42, Kluwer Academic Publishers, 1999.
28 Paternò, F., Mancini, C., *Developing Task Models from Informal Scenarios*, Proceedings ACM CHI'99, Late Breaking Results, pp.228-229, ACM Press, 1999.
29 Paternò, F., Santoro, C., and Sabbatino, V., *Using Information in Task Models to Support Design of Interactive Safety-Critical Applications*. Proceedings AVI'2000, pp.120-127, ACM Press, 2000.
30 Payne, S., Green, T., *Task-Actions Grammars: A Model of the Mental Representation of Task Languages*, Human-Computer Interaction, 2, pp.93-133, 1986.
31 Polson, P.G., Lewis, C., Rieman, J. and Wharton, C., *Cognitive Walkthroughs: A Method for Theory-based Evaluation of User Interfaces*, International Journal of Man-Machine Studies, 36, pp. 741-773, 1992.
32 Puerta, A.R. and Eisenstein, J. *Towards a General Computational Framework for Model-Based Interface Development Systems*. IUI99: International Conference on Intelligent User Interfaces, pp.171-178, ACM Press, January 1999
33 da Silva, P.P., *User Interface Declarative Models and Development Environments: a Survey*, Proceedings DSV-IS'2000, Lecture Notes in Computer Science, N.1946, Springer Verlag, 2000.

34 Tam, R.C.-M., Maulsby, D., and Puerta, A., *U-TEL: A Tool for Eliciting User Task Models from Domain Experts*, Proceedings IUI'98, pp.77-80, ACM Press, 1998
35 Tauber, M., ETAG: *Extended Task Action Grammar—A language for the Description of the User's Task Language*, Proceedings INTERACT'90, pp.163-174, Elsevier, 1990.
36 van der Veer, G., Lenting, B., Bergevoet, B., *GTA: Groupware Task Analysis—Modelling Complexity*, Acta Psychologica, 91, pp. 297-322, 1996.
37 van Welie M., van der Veer G.C., Eliëns A., *An Ontology for Task World Models*, Proceedings DSV-IS'98, pp.57-70, Springer Verlag, 1998.
38 Wilson, S., Johnson, P., Kelly, C., Cunningham, J. and Markopoulos, P., *Beyond Hacking: A Model-based Approach to User Interface Design*. Proceedings HCI'93. Cambridge University Press, 1993.
39 Yvory, M., Hearst, M. (1999), *State of the Art in Automated Usability Evaluation of User Interfaces*. 2000. Report available at http://www.cs.berkeley.edu/~ivory/ research/web/papers/survey/survey.html