Representación y uso del conocimiento de tareas en un entorno de diseño de interfaz de usuario

P. Markopoulos
S. Wilson
P. Johnson

Palabras clave: Diseño de interfaz de usuario, Modelado de tareas, Representación del conocimiento

Abstract: Un enfoque prototípico rápido al diseño de interfaz de usuario puede ser mejorado a través de la integración con un teoría de los usuarios' tareas. El documento presenta varias aproximaciones al modelado de tareas que implican estructuras de conocimiento de tareas (TKS) que proporcionan una visión de la estructura del conocimiento de tareas, un método para la identificación y modelado de este conocimiento. La especificación formal de lenguajes puede ser usada para proporcionar una descripción precisa de los modelos de TKS y para asistir en su validación; la utilización de LOTOS para este propósito se presenta. La integración del conocimiento de tareas con un enfoque prototípico rápido al diseño se introduce en el contexto de un trabajo sobre Adept. Un demostrador software para Adept proporciona un entorno gráfico donde la tarea modelada se utiliza para derivar una interfaz en una iteración inicial, un modelo que puede ser refinado para producir una prototipo ejecutable.

1. Introducción

El prototipado rápido ha sido un método popular, tal vez el 'estándar', para el diseño de interfaz de usuario y desarrollo. Este método aboga por el ciclo de diseño iterativo, donde el diseño de una iteración subsecuente está guiado por las teorías y modelos de interfaz de usuario, y no es guiado por cualquier teoría de las tareas. La prototipación práctica a menudo asume el diseño de cada prototipado a su vez, tal como lo hace con el desarrollo de las interfaces. El prototipado de interfaz de usuario (UIDE) dispone de una estructura de conocimiento de tareas (TKS) que proporciona una visión de la estructura del conocimiento de tareas, un método para la identificación y modelado de este conocimiento. Formales especificación lenguajes se pueden usar para proporcionar una descripción precisa de los modelos de TKS y para asistir en su validación; la utilización de LOTOS para este propósito se presenta. La integración del conocimiento de tareas con un enfoque prototípico rápido al diseño se introduce en el contexto de un trabajo sobre Adept. Un demostrador software para Adept proporciona un entorno gráfico donde la tarea modelada se utiliza para derivar una interfaz en una iteración inicial, un modelo que puede ser refinado para producir una prototipo ejecutable.

Recent research in human-computer interaction has suggested that users' knowledge of the tasks in which they participate could play a central role in the design and construction of user interfaces. A task is defined as an activity that is undertaken by one or more agents to bring about some change of state in a given domain. The investigation of what people do when they carry out tasks is termed task analysis and the results of such analyses are expressed as task models. This paper introduces the notions of task modelling and task-based user interface design, with particular reference to the use and representation of task models within a novel user interface design environment.

The Adept (advanced design environments for prototyping with task models) project [1, 2] has been investigating an approach to user interface design which provides a synthesis of a task modelling technique and a rapid prototyping approach to design. As suggested in Reference 1, this approach is based on an integrated framework expressing the relationships between models of users' tasks and models of user interfaces. The form of task model is a variant of task knowledge structures (TKS) [3, 4]. TKS is targeted at representing the knowledge that a person brings to a task, the so-called 'initial task knowledge'. Adept provides a graphical environment where the initial task knowledge is described and then used to derive the design of an interactive system which will provide effective support for that task. The need for a precise and unambiguous expression of task models prompted the use of the LOTOS formal specification language [5, 6].

2. Content and representation of task knowledge

Recent years have witnessed the development of a number of methods for analysing and modelling users' tasks, together with a growing awareness in the HCI community of the contribution which task analysis can make to user interface design and evaluation. Task analyses serve a variety of different purposes, primary among which is their use as an analytic tool for assessing the quality of a designed system, i.e. by describing the...
way users execute a task using an existing system, problems in the task may be identified.

Alternatively, task analysis may serve as a means of end user requirements capture, by describing task scenarios which a new system will be required to support. The implementation may then be judged in terms of its success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios. However, little work has been reported with regard to the constructive success in supporting those task scenarios.

Formal grammars were one of the first techniques used for the description of task models. Extended task action grammar (ETAG) [7] is one of the more prominent approaches and uses a grammar to describe an idealised interaction language structured in terms of the tasks to be performed. ETAG has been used both as a design specification notation and as an analytical tool for the early evaluation of a designed system. The User Action Notation [8] approach is similar to ETAG in that it also tries to capture the content and structure of the end users' input language. An extension to UAN [9] employs a set of temporal ordering operators in describing the multitude of possible temporal relationships between activities of the task. MAD (Méthode Analytique de Description) [10] takes a hierarchical approach to the description of task models which includes temporal relations between activities similar to those of the extended UAN work. MAD focuses on the users' planning activities, following an approach comparable to that of TKS.

The Adept project chose to use a task analysis and modelling technique which expresses people's knowledge of their tasks, as opposed to describing the tasks in terms of idealised input sequences. TKS theory assumes that the knowledge a person has acquired about a task is structured and can be represented as a task knowledge structure. This knowledge is activated and processed in association with task execution and is reflected in the observed behaviour during task performance.

The central components of TKS are goals, procedures, actions and objects. Consider a person working in a given domain. This domain is characterised by a state. The person is aware of a meaningful state to be achieved; this constitutes that person's goal. An activity undertaken by the person to result in a change of state, directed towards achieving the goal, is the task. The process of achieving a goal, there may be identifiable, meaningful intermediate states that constitute subgoals. There is a structure to this decomposition of goals and to the order in which subgoals are performed. This goal structure, referred to as a plan, results from a problem solving activity which identifies a way to achieve a goal in terms of subgoals (and subgoals of subgoals). The most basic element of activity identified in task performance is a single action. Actions are applied to objects. Objects reflect knowledge of the task domain and are described in terms of characters such as their attributes and relationships to other objects. Parts of the task may involve well practised sequences of actions as a mechanism for achieving a particular goal or a subgoal. These well practised segments of behaviour are referred to as procedures.

A method known as knowledge analysis of tasks (KAT) [11] has been developed for analysing task knowledge and identifying TKS components. KAT produces a description of a person's or a group of people's knowledge of the tasks they perform. It involves data collection techniques such as structured interviews, questionnaires, observations, concurrent and retrospective protocols, experimental techniques etc. The results of the analysis are generalised over sets of users and are encoded as a summary representation in a TKS model.

3 Specification of task models in LOTOS

Existing texts such as Reference 3 tend to use semi-formal languages when expressing TKS models, leaving each analyst free to invoke their own favoured dialect. In an attempt at providing some precision, a prerequisite to using TKS as the basis for user interface design, this Section presents work towards constructing a formal representation for the model. TKS is rich in procedural information so that notations such as process algebras, automata, state transition systems, grammars, etc. are plausible modelling techniques. Use of the LOTOS formal description technique is demonstrated here.

LOTOS is a hybrid language with two components: basic LOTOS which is a process algebra based on CCS and CSP, and the abstract data type language ACT ONE which is based on the theory of algebraic equational models. It was developed for the specification of network protocols and has features for expressing concurrency which make it appropriate for the specification of interaction [12]. LOTOS has been successfully employed in sizeable problems, it is an international standard [3], and there exists powerful tool support [13]. The constructs of the LOTOS language are not emphasised in this paper; see Reference 6 for a tutorial introduction to the subject. The use of LOTOS for the specification of task models is introduced by means of an example. In the specification segments below, parts of the text not specific to the example are italicised and can be thought of as templates to be filled in by the specifier.

The task model used in our examples resulted from a task analysis performed with radiographers from the London Hospital at Whitechapel. The task is that of taking x-ray examinations for use in clinical diagnosis and involves administration of x-ray request forms, patient management, exposing and developing the x-rays and examining the results. The highest level goal in this task is 'to take an x-ray', denoted in the examples below as 'TakeAnXRay'. A first level of decomposition of the TakeAnXRay goal yields six subgoals which are performed sequentially.

Goals, subgoals and procedures of the task are modelled as terminating LOTOS processes (indicated by the keyword exit). For example, the specification segment below shows a process definition which models the TakeAnXRay goal. This definition states that the 'behaviour' of TakeAnXRay is simply the sequential performance of its subgoals, where sequence is denoted by the enable operator '>>'.

```plaintext
process TakeAnXRay:exit  
 (* TakeAnXRay is a goal *)  
 InitialConsideration >>  
 FurtherConsideration >>  
 SetUpAndTakeShots >>  
 Processing >>  
 ConsiderXRay >>  
 LogRequestCompleted >>  
 (* . . definitions of sub-goals omitted . . *)  
 endproc (* TakeAnXRay *)
```

The subgoals InitialConsideration, FurtherConsideration etc. are also described as LOTOS processes, consisting of the composition of their subgoals, until the level of task
procedures is reached. Below is an example of a procedure, TakeShot, requiring the sequential execution of a series of actions, denoted by the action prefix operator "*:"

    process TakeShot : exit =
      (* Take Shot is a procedure *)
      hide action in
      action !levelInstructions;
      action !pressButton;
      action !hearFeedback;
      action !markUndevelopedFilm;
      exit
    endproc (* TakeShot *)

Task performance unfolds as subgoals are satisfied to enable the execution of subsequent sub-goals. However, task activities are not always performed in sequence; sometimes activities are multithreaded or one may interrupt another. A small set of primitive relationships defined between task components allows the description of the multitude of possible sequences of task activities. The radiography task contains two alternative ways of determining the x-ray exposure: either looking up the relevant values on a wall chart and then setting the controls manually, or using a ‘pre-set’ on the control panel. The example below shows how the DetermineExposure subgoal may be expressed in LOTOS, with the alternatives expressed using the choice operator \[ \lor \]:

    process DetermineExposure : exit =
      (* DetermineExposure is a goal *)
      (LookUpExposure \lor SetExposure)
    \[
    \]
    UsePreset
    where
    (* definitions of sub-goals omitted ... *)
    endproc (* DetermineExposure *)

The definitions of the various LOTOS operators and their formal semantics in terms of labelled transition systems can be found in Reference 6. Their informal interpretation in the context of TKS is summarised in Table 1. The Table refers to the LOTOS abstraction of processes which, in the present context, are used to model goals, subgoals and procedures. The action prefix operator is used only within procedures in accordance with TKS where actions are always part of a procedure.

The conventions adopted promote a consistent style for the description of task models, making it easy to read and write TKS specifications. However, this scheme does not make use of the full expressive power of LOTOS; the synchronisation operator which is crucial in interface specification and generally in the specification of communication is not used here. Further, attention has been focused on sequencing (or control) information, while the objects (or data) have been largely neglected. The formal task model could be extended to include the semantics of objects and actions expressed using ACT-ONE, the algebraic component of LOTOS. The objects would be represented as abstract data types and the actions as operators on them.

4 Graphical task model editor

A general purpose formal language such as LOTOS provides greater expressive power than is required for specifying task models, a fact which is demonstrated by the constrained use of LOTOS in the above examples. To encourage designers to construct explicit models of the users' tasks, it is desirable to provide them with graphical tools for describing the models which, at the same time, produce an equivalent formal description of the model. A well-designed graphical representation for the model has the advantage of illustrating clearly the relationships between components. A prototype version of such a tool for editing task models is shown in Fig. 1. This tool was implemented as part of a software demonstrator for the Adept project and provides the designer with a graphical representation of the model together with navigation and editing facilities. The editor can be regarded as syntax directed: the task model is guaranteed to be syntactically correct by construction.

The larger, upper view in the editor displays a hierarchical structure consisting of the goal, procedure and action components of the task model. The root of the tree denotes the top level goal, TakeAnXRay, for the radiography task. The two smaller, lower views display the taxonomic substructure component (the objects and their descriptions). Each goal and procedure has a 'type' which expresses the temporal relationships between its subcomponents, thereby expressing the temporal constraints which exist between activities of the task. The possible set of relationships is based on the LOTOS operators for
process composition (Table 1), making it trivial to convert such a task model into an equivalent LOTOS description.

A direct advantage of this approach to modelling the users' task explicitly is that the model can be validated with the subjects of the task analysis. Early validation of the task model allows the designer to proceed with confidence, or to correct the represented knowledge if necessary, and involves the user in further stages of the design than would otherwise be possible. At present, validation is achieved by using the Lite toolset [13] to provide a symbolic animation of the LOTOS version of the task model; a longer term goal is to provide animation of the graphical representation.

5 Models in design of user interfaces

The previous Sections have discussed how task modelling techniques such as TKS can be used to represent the knowledge possessed by users and recruited in performing some existing task. However, rather than employ such techniques in isolation, we wish to position task models within the context of a user interface design methodology which embraces all aspects of the design process from analysis through to implementation. Software systems (and other artefacts) are created to support users in performing particular tasks and the hypothesis underlying this wish is that 'better' designs for artefacts should result from basing the design on users' existing task knowledge.

This Section summarises work on the Adept project on integrating considerations of users' tasks with a rapid prototyping approach to user interface design; for further details see Reference 2. The integration is achieved by basing the design process on a series of interrelated models, with the tools of Adept supporting the expression of the various models and the progression from one modelling stage to the next. Taking a general overview, the design process in Adept commences with a task analysis procedure, resulting in a model of the users' existing task knowledge which the designer may specify using the task model editor. Components of this task model have direct relevance to the design of an interactive system, most notably, sequencing information in the task model can flow into the design of interaction sequences in the user interface; task objects may be realised as interaction objects in the interface and task actions may be realised as actions on the appropriate interaction object. This task knowledge is used as the basis for creating a first prototype of a system to support the task, thereby incorporating some theory into the rapid prototyping cycle.

Making the simplifying assumption that the new system is to support the users' task as it stands, the next stage of the design process involves creating a complementary 'abstract interface model' (AIM), a process which is partially automated in Adept. The AIM is a high-level description of an interface to support the task without any commitment to a detailed design: it says what the interface should do but not how it should be done. It has a hierarchical structure consisting of two types of components: groups and abstract interaction objects (AIOs). A group is a collection of other components, which may be either further groups or AIOs. AIOs are abstract descriptions of the interaction objects (or 'widgets') in a user interface and support the actions of the task. AIOs are categorised in terms of the type of input they support, for example, the action of setting the exposure time in the radiography task might be supported by a numeric AIO in the AIM. Temporal relations similar to those of the task model may be defined between the components of the AIM, thereby defining interaction sequences in the user interface.

Another graphical tool, shown in Fig. 2 facilitates the viewing, navigation and editing of the AIM. The root of the AIM tree represents the complete user interface, each node in the tree is a group and the leaves of the tree are the AIOs. The AIM is still at the level of a specification. The transition from this specification to an executable implementation (a 'concrete interface model') is performed by an automatic generator tool in the Adept demonstrator. This tool selects one or more widgets for every AIO in the AIM, performs complex layout calculations to instantiate the widgets and derives sequencing information from the temporal orderings given in the model. The generator is guided by input from a user model [14] which produces design recommendations based on information about the users additional to that captured by the task model.

In most circumstances, our assumption that the new artefact is to support the existing task will not be valid; new systems are generally designed to meet specific design goals or to address problems identified in existing systems. Moreover, design should be a creative process which encourages and facilitates the emergence of new ideas. Hence, the initial task model may be supplemented with a description of problems in the existing task and/or design goals and these are used as a basis for redesigning the task, creating a designed task model. The designed task model is an expression of a new task to be performed with the assistance of the artefact to be created. This redesign of the task involves various activities, such as determining which of the tasks mentioned in the initial task model are to be supported by the new artefact; those
which are not to be supported are deleted in the designed task. Other activities might include modifications to the temporal relations, restructuring of the task or the introduction of new subtasks. It is then this model of the new task for which an AIM is produced. Although the discussion here has focused on the top-down design of artefacts from task models, it is not our intention to be prescriptive about the design process and therefore designers are free to work at whatever level of abstraction they find appropriate. Design activities similar to those involved in the production of the designed task may be carried out at any modelling stage; Reference 2 discusses this flexible approach to task-based design in greater depth.

The preceding discussion has encompassed three important models which support different stages of the design process: task model, abstract interface model and concrete interface model. The explicit nature of these models facilitates evaluation at all stages of the design process (by either analysis or animation of the models). This, together with the automated tool support provided by Adept, encourages iteration round design alternatives and therefore a rapid prototyping approach to design.

6 Discussion

There are two interesting and novel aspects to the work: the use of formal description techniques to express task knowledge and the integration of task models with a rapid prototyping approach to design. Turning first to the former, existing research in the application of formal methods to human-computer interaction has tended to ignore task knowledge, such that those task descriptions which do exist within formal approaches are high-level definitions of application functionality with little psychological basis. The work reported here has demonstrated how an established representation for modelling users' task knowledge may be expressed precisely using a formal language. LOTOS provides the expressive power at a high level of abstraction for capturing properties of task and interface models, however, it is not yet clear whether LOTOS is the most appropriate formalism for this purpose. Rather, it has proved hard to develop and interpret models with LOTOS and the work on Adept has demonstrated a requirement for graphical formalisms.

Related work on the formal specification of user interfaces has provided design notations for the specification of interfaces, for example, the notations described in Reference 12 and 15 are based on process algebras. However, relatively little research has been presented on the relationship between these interface models and task models. Bosser [16] suggests that the task model, described as a set of action sequences, can be used for the verification of the design to establish (informally) that the design can indeed perform all the action sequences of the task model. Abowd [17] formalised the relationship between task and interface behavioural specifications as a refinement relationship, based on the simplifying, but very restricting, argument that there is a one to one correspondence between task events and interface events.

Arguments in favour of a formal expression of the relationship between task models and interfaces include the suggestion that such an achievement would improve our understanding of user interface design, and result in a task-oriented early evaluation method and therefore a formal method for interactive software development. However, such an approach lacks flexibility and enforces a top-down approach to design, where all major design decisions must be taken in terms of their consequences for the task. We do not believe that designers work within such a rigid framework and therefore do not wish to prescribe a design method in which all design decisions must be made at the task level. A less formal view of this relationship between tasks and artefacts has been central to the second novel aspect of the Adept work: the integration of task models with a rapid prototyping approach. The tools of the Adept demonstrator successfully support this integration and facilitate the transfer of knowledge from the task model to the artefact design, although further work is required to develop the tools beyond their present prototype type status. A number of interesting new research issues have arisen from our experience with the demonstrator. One such issue is concerned with the fact that the tools do not provide the designer with any guidance in designing tasks and artefacts. A first step in this direction will be to incorporate a design rationale facility, allowing designers to document their design decisions and the reasoning which lies behind them; the development of design principles to guide this process is an area for longer term research. Adept has focused on the contribution which models of users' tasks may make to the design process; another area for further investigation is extending the models to cover other relevant factors, for example, models of the organisation or application domain.

Interface design is a creative yet problematic process. We wish to provide designers with models and tools which will enhance and support their creative skills. It is our belief that explicitly modelling users' tasks within a graphical prototyping environment, together with a framework expressing the relationship between task components and user interface components, is one way of achieving this. Designers may be encouraged to introduce some theory into the design cycle by basing the prototypes on task descriptions but yet are not overly constrained by early design decisions which may later prove inappropriate.

7 References


13 CANEVE, M., and SALVATORI, E. (Eds.): 'Lite user manual'. LOTOSPHERE consortium, 1991


16 BOSSER, T., and MELCHIOR, E.M.: 'The SANE toolkit for cognitive modelling and user centred design', in GALER, M., HARKER, S., and ZIEGLER, J. (Eds.): 'Methods and tools for user centred design for information technology' (North-Holland, 1992), pp. 93-125