

Adept - A task based design environment

P.Markopoulos, J.Pycock, S.Wilson and P.Johnson.

Queen Mary and Westfield College

Abstract

Modern user interface development environments are based on fast prototyping which as a methodology does not incorporate any theory or design principles. Adept (Advanced design environment for prototyping with tasks) incorporates a theory of modelling users and user task knowledge known as Task Knowledge Structures [11], and extends it to a theoretical framework for modelling user, task and interface characteristics. This paper introduces the underlying framework, and discusses how this can be used to support task based user interface design.

1. Introduction: UIDEs and user oriented design of user interfaces

User interface management systems (UIMSs) have gradually become a focal point of research in software engineering and human computer interaction since they first emerged early in the last decade. The earliest examples of UIMSs (e.g. Menulay [4]) provided run-time support for user-computer interaction, but little else. A major aim was to facilitate the prototyping of user interfaces, but minimal assistance was given to the unfortunate interface designer, who was generally obliged to describe the appearance and behaviour of the interface using a textual notation. Subsequent generations of UIMSs have addressed this deficiency by providing the interface designer with increasingly sophisticated design tools [9]. Such tools support representation, design, implementation, execution and evaluation of the interface, supporting the methodology of rapid prototyping. This provides the ability for early observation and evaluation of the behaviour of the interface and its subsequent modification and amelioration, and by doing so promotes the collaboration of designers, implementors and users.

Unfortunately rapid prototyping offers no guarantee that design will be any quicker, nor that mistakes will not be repeated in subsequent iterations of the prototyping and evaluation cycle. The method produces no explicit understanding, or theory, beyond the particular

implementation. In short, it lacks principles for design and design decisions. Unfortunately, the method of rapid prototyping is consistent with the above scenario; and consequently tools designed to support this method are often no more than implementation aides. In practical terms, rapid prototyping tools alone suggest no principles by which the turn-around time in interface design can be gradually reduced while at the same time rapidly achieving a high degree of usability. A framework for HCI requires cumulative progression from application to theory, and demonstrable application of that theory.

The Adept (Advanced design environment for prototyping with tasks) project aims to construct a design tool, that will embody a theory of modelling task knowledge, the Task Knowledge Structure theory [11]. This theory is unified into a theoretical framework which encompasses user, task and user interface models and which expresses their relationships. Basing the design on such a framework allows the introduction of user task considerations early in the design process, and furthermore the evaluation of designs with regard to their effect on users' tasks.

A discussion on the need for a framework follows in the next section, and the models of the framework are presented. A case study for the design of a CAD tool for jewellery design is used to motivate the discussion in section 3 on the application of formal description techniques for the expression of the models of the framework and their use during the design. Section 4 discusses the Adept software environment which will support this task-based approach to user interface design and finally the conclusions are presented in section 5.

2. A framework for task-based design

Many authors, e.g. [3], have suggested viewing both systems and users as different types of information processing systems. However, it is also the case that these two forms of information processing systems contain internal constraints on the type and form of their

processing. Such constraints may act at many levels, from the level of the substrate (biological or electronic), to the architectural model they contain. The Adept project is extending this to consider how descriptions of users and descriptions of the system's specification can provide matches and mismatches which themselves provide a key input into understanding the interaction of the two. The basis of this approach is to develop an understanding of how a user conceives of a task scenario (and their corresponding knowledge about elements of that task, such as task objects), and in addition a high-level description of any device implementation related to the task scenario. Together these two models offer the potential for cross mapping between a user's task model and the model underlying a designed artifact. Hence, the implementation of an object in, for example, a PAC model [5] may differ from the users' corresponding understanding of the related task object. It is this relating of the models of the two information processing systems, and in particular their knowledge specifications, that highlights the demands and constraints that one places upon the other. Direct mappings or correspondences between models for a particular element, e.g. an object, lead to the prediction that no learning is necessary in this transfer.

The framework relating models of users and tasks to models of interactive systems, provides a mechanism for ascertaining whether a particular implemented design constrains, supports, compromises etc. the users' direct understanding of the objects. It provides a method of generating claims about the design and answering them according to explicit models of users and tasks. It also provides a foundation for metrics to compare an interface both with respect to other interfaces, and with respect to the models of the task and the user.

2.1 Models of the framework

It is first necessary in developing such a framework to decide upon which models need to be incorporated within it. Task, user, and interface models have been proposed, above, to be essential elements of the framework; it remains however to adopt one of the many modelling techniques available in each of these four areas.

Task Models; A task model is a symbolic representation of that which the modeller considers significant about a task. Task models may be used descriptively, predictively, and prescriptively, though in practice few existing techniques have the power to do all.

TKS is a well established task modelling approach based on knowledge elicitation and description techniques called Knowledge Analysis for Tasks (KAT). It provides a

summary representation of the different types of knowledge recruited by a user in a task. This knowledge is stored in memory as conceptual structures which are described in terms of a goal-oriented substructure, a procedure set, and a taxonomic substructure. The goal-oriented substructure models the goal and sub-goal structure, and is the product of any planning activity that the user has to engage in to define the sequence of carrying out the task. The procedure set defines the executable task actions, and their control structures, which satisfy a sub-goal. There are relations of e.g. order, dependency etc. between procedures and between sub-goals. Finally, the taxonomic substructure contains categories of task actions and objects that have been identified as required for the task. It also models the properties, attributes, centrality, typicality, and inter-relationships of objects and actions.

User Models: There are many different definitions of the term 'user model', and a range of potential roles that 'user models' can play in design. A user model concerned with the knowledge characteristics of groups of users is being developed and further work on cognitive processing architectures, such as Interacting Cognitive Subsystems [2], is still under way.

Groups of users are modelled in terms of the normal knowledge systems (i.e. systems of domain knowledge, or systems of across-domain knowledge) that are associated with members of that group. The detailed content of a knowledge system is not elicited or modelled, but the list of knowledge systems, and levels of expertise, associated with a group of users are.

Users are grouped in terms of the knowledge systems required or associated with jobs and roles users are associated with, or involved in the training for them.

Interface Model: Models of user interfaces have been created to satisfy the need to represent abstractly the logical decomposition of an interface to its components and their dynamic behaviour. The most influential models have been the Seeheim [15] and the PAC [5] models.

The abstract user interface model (AUI) that Adept supports is based upon a composite object architecture [8]. The user interface is a hierarchical composition of user interface objects. The main abstractions related to this model are events which are discrete and momentary actions, and state, which is a mapping of attributes related to the object to values, and the state of its component objects. Each object, is modelled by the set of events it can recognise, the sequencing of those events and its behaviour, which determines the effect of the events on the state.

2.2 Relationships across the models

Relationships between objects and properties in a task model, and certain UIOs and their features, can be mapped in a relatively straightforward manner. Similarly, it is possible to identify relations between task actions and interface events, and between task states and interaction states.

The procedures within a TKS model are formed of one or more actions. The total set of objects and the set of actions can be listed. The associated actions can be listed for all objects, as well as objects which are related by sharing an association with an action.

In terms of human information processing systems and task models the talk is of actions upon objects, or with objects, or using objects, etc. However, the implementation in an artifact of an information processing system may involve modelling actions essentially as methods within an object, and the talk becomes one of the action-triggers i.e. messages or events. The development of these mappings, based on relations between objects and actions and between objects and other objects, also suggested that more permanent relationships of composition and class membership existed between both objects in task models and in implementation architectures.

A second, and important, source of inter-model mappings originated from considering the relationships between procedures, and between sub-goals in a goal substructure. TKS includes relationships of order, dependency, timing etc. between its structural planning units, such as procedures and sub-goals. These task plan relationships were seen as potential inputs into specifying

rules of dialogue control and therefore as a way to transfer the knowledge about the task performance into the design of the user interface.

Similarly, usability will be affected if conceptual components of the task model are unrepresented in the interface. The centrality and typicality ratings of objects may also contribute to these predictions regarding the importance of transferred knowledge.

Relating task models and user models has been considered from a slightly different perspective, because they both relate to the user information processing system. The user model, in terms of knowledge characteristics and expertise level in knowledge systems, was seen to be associated with the tasks, jobs, and roles. This information maps on to the task model in two senses. Firstly, the definition of task elements, such as task objects, in terms of feature sets, can be supplemented by making explicit the wider knowledge systems with which an element is also associated.

Secondly, the description of users in terms of their knowledge characteristics can act as a source of design choice rationales and validations. While the detailed task model of TKS provides correspondences with an interface, in terms of e.g. the taxonomic elements and the interface elements, it is also possible that other interface elements may be introduced in the design which have no correspondences with the task model. A valid rationale for their inclusion, and a prediction of the learnability consequence, may be obtained if the user model can justify their inclusion on the grounds that the introduced elements are part of the users' wider knowledge systems.

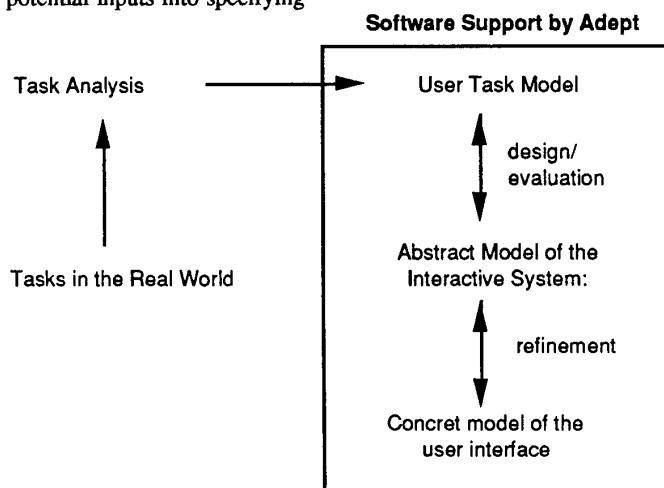


Figure 1. Adept aims to provide software support for the framework that guides the design from a user task model to an implementable prototype.

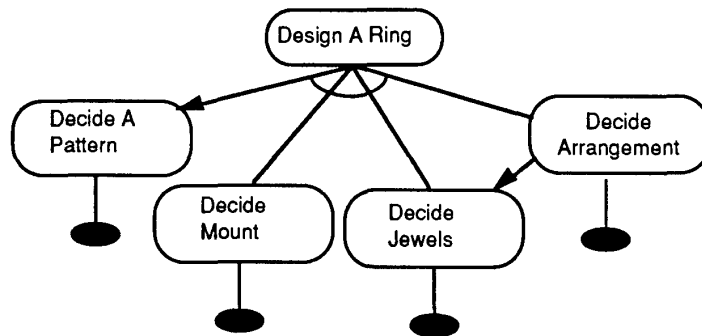
3. The formal description techniques used by the Adept tool.

It is self evident that a representation technique or rather a set of representation techniques is required for recording behavioural, structural and detailed representation of both visible and non visible aspects of human computer interfaces. These techniques should have a sound theoretical basis, and be independent of the tools on which they are implemented [17]. The formal description technique (FDT) suggested, is a combination of notations addressing the different components of the models that have to be described. The requirements for the FDT are:

- Compatibility of the specification of the task model with that of the user interface model. Preferably, the same notation should be used across the models. This

- will facilitate the expression of the framework as a set of mappings of components of the two models.
- Support of an event based model. This requirement is set by the asynchronous nature of modern direct manipulation interfaces.
- Explicit representation of concurrency, for support of multi-thread dialogues.
- The notation should be a means of communicating the design to people of varying backgrounds, from software engineers and programmers to human factors experts. The FDT should cater for a wide range of tastes and disciplines of its users.

At present, the project is investigating the feasibility of employing the formalism of Communicating Sequential Processes (CSP [10]). CSP is a simple and flexible notation that may also allow formal proofs to be performed upon a specification. A subset of this notation is used to express task models as represented in TKS.



Elements of the notation

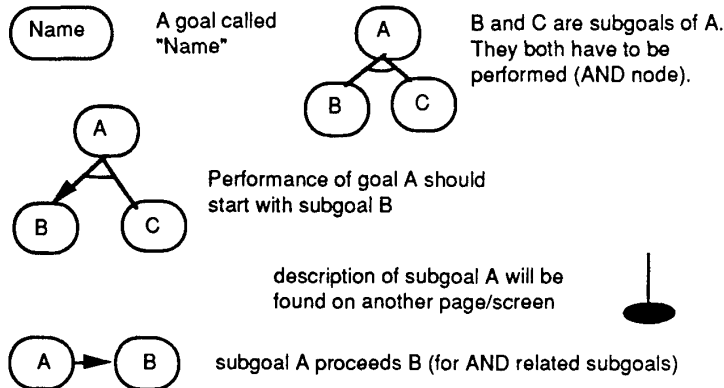


Figure 2. Specification of the goal substructure for the task of designing a ring.

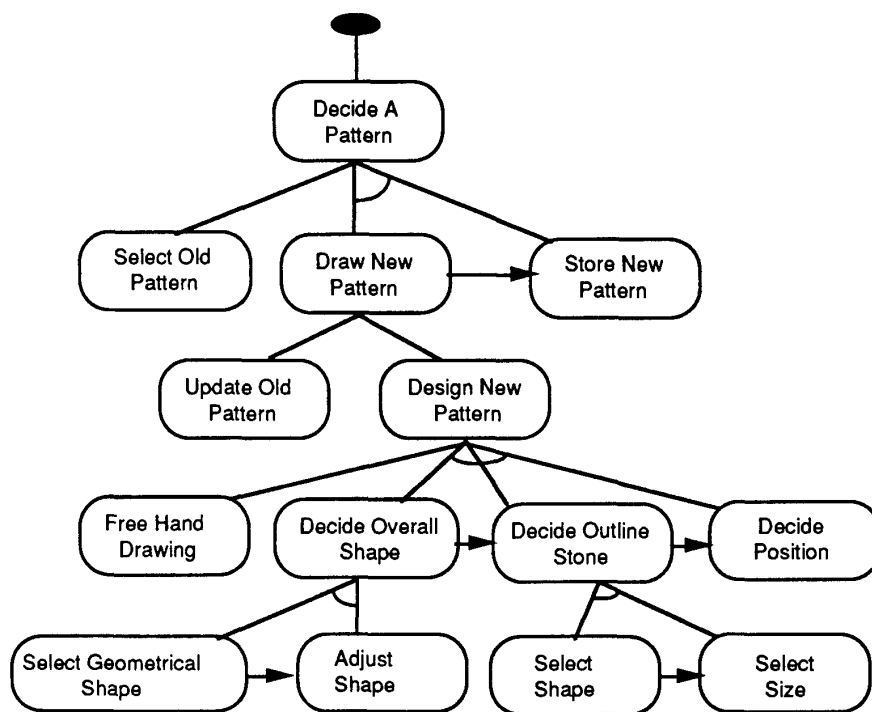
The requirement for the notation to be usable by various persons and under various roles, underlines the need for the ease of use of the notations. The conflict between textual and graphical notations emerges again. As a solution a combination of a diagrammatic notation and CSP has been adopted. The diagrammatic notation was developed as a means of expressing goal substructures in TKS, while CSP can also be used for the expression of the dialogue component of the interaction.

Goal substructures expressed in this diagram form can be translated to CSP process expressions. TA small subset of CSP is used and this can be found appendix 1. Minor, "syntactic sugar" extensions are employed as well.

3.1 A case study

A case study is presented in this section to illustrate how elements of the task model are transferred into the design of the user interface. The case study involved a task analysis to guide the design process. The objective of the design was to create a CAD tool to aid a jewellery designer in the task of designing a ring. Observations on this example clarify the nature of the proposed framework and its formal expression.

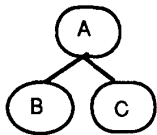
The basic abstraction unit is that of the goal. A natural abstraction for the goal substructure is a tree structure, as the goal-subgoal relationship is easy to represent. The tree structure is similar to AND-OR trees where a goal is represented by a node in the tree. Sub-



Meaning of symbols



Expansion of subgoal of a previous page/screen



B and C are subgoals of A. They are alternative strategies for the achievement of A (XOR).

Figure 3. Decomposition of the subgoal "Decide A Pattern"

goals related with OR arcs to their parents are alternative ways to satisfy the parent goal while those related with an AND arc are all necessary for the performance of the parent. Among the sub-goals related with an AND arc, a partial temporal ordering is established by the introduction of arrows which act as ordering operators between the subgoals. The notation provides for the specification of "subgoal A before B" relations, and "subgoal A is first". An example of this notation is given in figure 2, where a jeweller's goal substructure for the task of designing a ring is presented.

For example, in the task model of the case study, the task of designing a jewel has four subgoals, which all have to be performed.

This is abstracted as a CSP process:

```
DesignARing = DecideAPattern;
              (DecideMount !!
              (DecideArrangement; DecideJewels))
```

The above implies that DesignARing is carried out by first performing DecideAPattern followed by the sequential but unordered performance of DecideMount and the sequential performance of DecideArrangement and DecideJewels.

DecideAPattern is expanded in figure 3. It can be written as:

```
DecideAPattern = (DrawNewPattern;
                  StoreNewPattern) [] SelectOldPattern []
                  DrawNewPattern = UpdateOldPattern []
                  DesignNewPattern
```

Processes can be combined with the general parallel operator || which implies that they are performed in parallel

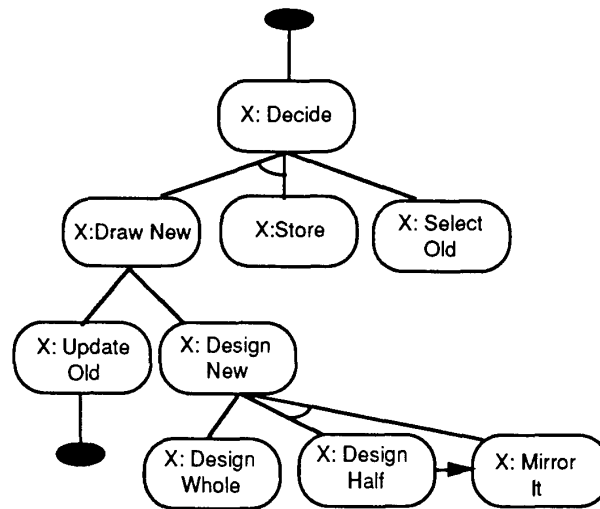


Figure 5. The parametrised task modelling component Decide, with parameter X, where X belongs to {Setting, Shank, Head}

synchronizing on common events and interleaving on the rest.

```
DesignNewPattern = FreeHand []
                  (DecideOverallShape;
                  DecideOutlineStone; DecidePosition)
DecideOverallShape = SelectGeometricalShape |
AdjustShape
DecideOutlineStone = SelectShape; SelectSize
```

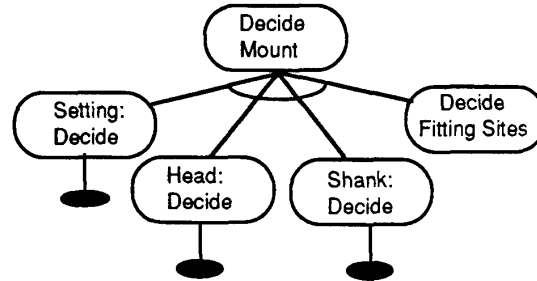


Figure 4. Expansion of subgoal Decide Mount. Introduction of Parameterised components

DecideMount is described by means of the parameterised component Decide(X), as in figure 4.

```
DecideMount = Decide(Setting); Decide(Head);
              Decide(Shank); Decide(FittingSites)
```

In the specification of DecideMount a parameterised component appears, which is specified graphically in figure 5.

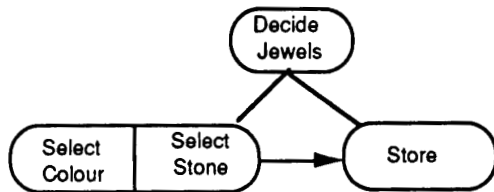


Figure 6. The general parallel operator between two subgoals, like SelectColour and SelectStone.

And the textual form is:

```

Decide(X) = SelectOld(X) [] (DrawNew(X);Store(X))
DrawNew(X) = UpdateOld(X) [] DesignNew(X)
DesignNew(X) = DrawWhole(X) [] (DrawHalf(X) []
MirrorCopy)
  
```

The notation provides for the specification of parallel performance of sub-goals as is shown in figure 6, where the top level subgoal DecideJewels is described as

```
DecideJewels = (SelectColour || SelectStone); Store
```

A task model editor is needed to support the construction of task models. This tool will allow editing of a goal substructure, by direct manipulation of its graphical representation, while in the background this will be translated to CSP notation.

The analysis of goal structure knowledge stops at the level of procedures, which are sequences of actions that the user performs without any "planning" activity taking place. The procedures are abstracted as CSP process expressions which specify event sequencing. Events are therefore the smallest units of abstraction used in the goal substructure, and are associated with an individual action of the task performer. The allowable sequence of events is that specified by a process expression. For example the process Store will recognise the event store and terminate successfully

```
Store = store -> SKIP
```

while the process MoveStone is defined recursively to recognise repeated occurrences of the event moveStone before a release event happens. After release it terminates successfully.

```
MoveStone -> ( moveStone -> MoveStone)
| (release ->SKIP)
```

The specification of the task model is completed with the taxonomic substructure, which can be specified in a simple frame based notation or more formally if desired. The project is now considering the combination of a process algebra with an algebraic specification technique, as is the case with the Lotos specification language [Eijk 89].

4.2 A The dialogue specification for the AUI, is distributed across the UIOs as their individual dialogues.

3.2 The AUI for the jewellery design tool. Comparison with the TM.

The dialogue part of the AUI is itself a CSP process, that is associated with the root of the composite AUI model and which combines the individual dialogue models of the UIOs. The dialogue of the case study will not be presented in full, rather its relation to the goal substructure will be highlighted.

The dialogue model for the jewellery design system, includes quit, confirm and cancel dialogues, which good interface design practice necessitates but which did not originally appear in the task model. The highest level of the dialogue for the jewellery design tool (JDT) is now:

```
JDT = Quit || DesignARing || Cancel || Accept
```

Where a Quit dialogue and a Cancel and Accept dialogue have been added to the dialogue structure derived directly from the goal substructure. These dialogues are always available in the prototype.

The goal substructure of designing a ring has been extended to include the act of naming a jewel, something alien to the task of jewellery design per se, but useful where information organization and retrieval is concerned.

```
DesignARing = JewelName; DesignARing = DecideAPattem;
```

```
(DecideMount!! (DecideArrangement; DecideJewels))
```

At the lowest level, components of the dialogue are predefined dialogues, of interactive objects. For example, the dialogue of a menu should be predefined, so the specification of the dialogue should not repeat it. This is an essential modularisation of dialogue, which should be enhanced so that the library of predefined interactions could be amenable to extension.

A further deviation from the task model is the dialogue for DecideMount

```
DecideMount = Decide(Setting) ||| Decide(Head) |||
Decide(Shank) ||| Decide(FittingSites)
```

The subdialogues do not have to be performed in strict sequence as was the case with the corresponding subgoals. They all have to be performed but their performance is interleaved. Further differences from the task model can be identified in the procedure specification, which corresponds to the lower level of the dialogue, where input and output actions dependent upon the interaction styles supported, enter the specification.

A direct mapping between the task model goal substructure and the dialogue model is therefore explicated when the goal substructure is written in CSP. Still the AUI is incomplete. A variety of different designs can implement this dialogue model, ranging from highly structured conversational interfaces to direct manipulation interfaces. In order to guide the designer

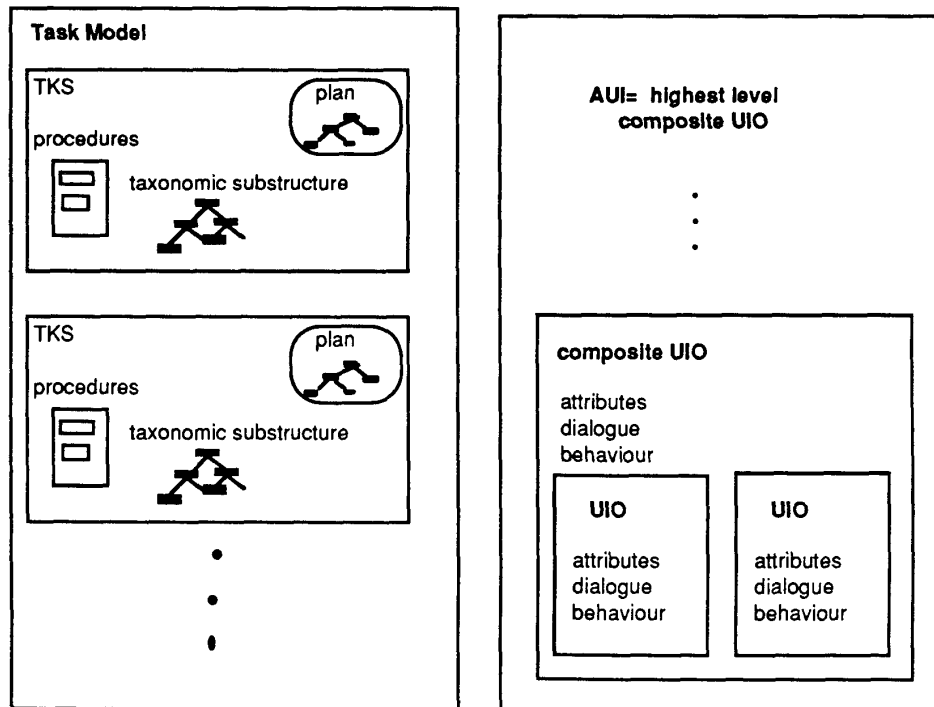


Figure 7. The framework aims to establish relationships between the components of the task models and the Abstract User Interface

further we ought to step back and look at the other elements of the task model, i.e. the objects and actions of the task model.

The set of actions in the TM should be a subset of the set of events of the AUI, i.e. for each action of the TM an event of the AUI, which the user can initiate, should exist, and more than this its behaviour should reflect the effect of the action on the task domain. This sort of correspondence between the two models, allows transfer of knowledge about the task in the AUI.

Figure 7 shows the elements of the two models that have to be related. There may be several mappings from a task model to a user interface model and naturally there is a significant proportion of the user interface model that cannot be derived from the task model.

The style of interaction is closely related and dependent upon the sustained relationship between task model and user interface model. A conversational style of interaction will impose different mappings to be sustained than a direct manipulation style. Thus supporting some mappings between task model and interface model may enhance a particular style of interaction.

For example objects included in the taxonomic substructure can be mapped directly to interaction objects, and the same holds for their class membership. The actions that can be performed on the task model objects can be mapped to events in the AUI, while the effect of the event on the task domain, can be mapped into the behaviour specification. Such a mapping not only effects transfer of task knowledge to the interface design but promotes a direct manipulation style. On the other hand, basing the design on a global dialogue model that corresponds to the goal substructure tends to lead to a more conversational style of interaction.

4. The Adept architecture

As mentioned before, an objective is to introduce scientific principles and engineering discipline into user interface design prototyping. The Adept framework brings user task considerations into the design cycle. Following the work by Carroll [91], design is seen as a following the task artifact cycle. The task artifact cycle can be supported by developing UIDEs

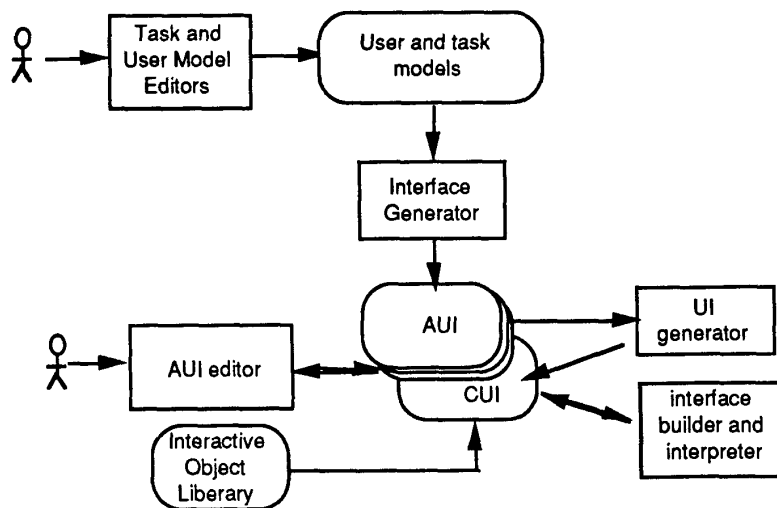


Figure 8. Software Components and Data Representations of Adept

that encompass user task models.

The UIDE will provide a set of elements of the AUI that should correspond to elements of the user task model. The designer will refine and modify before producing an executable specification of the AUI. In figure 8, the basic software components and data representations of the environment are presented, in order to demonstrate the above points. As an implementation consideration, suffice it to say that the Adept project needs a state of the art architecture for the run time system. The focus of this project is the introduction of task based considerations into user interface design.

5. Conclusions

The Adept project aims to provide the designer of interactive systems with a theoretical framework which will serve to guide the design process. The framework relates a model of the user and their task to a model of the interactive system. It has been shown that the goal substructure and procedures of the user-task model can be mapped onto the dialogue structures of the AUI. Furthermore, there is a relationship between the objects of the two forms of model, in terms of their attributes, composition, class membership and the actions in which they participate. The user-task models presented in this paper are an extension to task knowledge theory to include user modelling considerations.

An interface design methodology based on such a framework allows the introduction of user task considerations early in the design process and therefore is an improvement on the traditional trial and error

prototyping approach. The theoretical framework also offers a mechanism for ascertaining whether a designed artifact can support the user's task effectively and provides a basis for comparison of design alternatives.

The research on the Adept project is still under way. At present, the focus is on the investigation of notations to express the various models and capture the mappings of the framework. The next phase of the project will be largely concerned with the implementation of the software environment to support the task-based design methodology.

6. Acknowledgements

The Adept Project (Advanced Design Environments for Prototyping with Tasks) is a collaboration between Queen Mary and Westfield College (University of London), British Aerospace Plc, British Maritime Technology Ltd., and MJC². It is led by Queen Mary and Westfield College and is funded by the DTI's Information Engineering Advanced Technology Program (Grant No. IED 4/1/1573).

7. References

- [1] Heather Alexander, *Structuring dialogues using CSP*, in M.D.Harrison and H.W.Thimbleby (eds.), "Formal Methods in Human Computer Interaction", Cambridge, 1990.
- [2] Barnard P., *Cognitive resources and learning of dialogues*, in Carroll, J. "Interfacing Thought: cognitive aspects of human computer interaction", 1987.

[3] Philip J. Barnard and Michael D. Harrison, *Towards a Framework for Modelling Human Computer Interaction*, Amodeus Project Document: RP3/WP5, May 91.

[4] W. Buxton, M. R. Lamb, D. Sherman and K. C. Smith, *Towards a Comprehensive User Interface Management System*, ACM Computer Graphics, Vol. 17, No. 3, 1983, pp 45-42.

[5] J. Coutaz, *PAC An Object Oriented Model for Implementing User Interfaces*, Laboratoire de Genie Informatique (University of Grenoble), BP 68, 1987.

[6] P.H.J. van Eijk, C.A. Vissers, M. Diaz (eds), *The formal description technique Lotos*, North Holland, 1989.

[7] James Foley, *Transformations on Formal Specifications of User-Computer Interfaces*, Computer Graphics, Vol.21, No. 2. April 1987.

[8] Ralph D. Hill, *Supporting Concurrency, Communication, and Synchronization in Human Computer Interaction-The Sassafras UIMS*, ACM Transactions on Graphics, Vol.5, No.3, July 1986.

[9] Deborah Hix, *Generations of User Interface Management Systems*, IEEE software, September, 1990.

[10] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[11] Robert J. K. Jacob, *A Specification Language for Direct-Manipulation User Interfaces*, ACM Transactions on Graphics, Vol 5, No. 4, October 1986, 283-317.

[12] Peter Johnson, Hilary Johnson, Ray Waddington and Alan Shouls, *Task Related Knowledge Structures: Analysis, Modelling and Application*, People and Computers IV, Cambridge University Press, 1988.

[12] Peter Johnson, *HCI models in software design: Task models of interactive software systems*, from "Software Engineering Environments 89", ed.K.H.Bennet, Ellis Horwood 1989.

[13] Peter Johnson and Hillary Johnson, *Knowledge Analysis of Tasks: task analysis and specification for human-computer systems*, Downton A. (ed.), "Engineering the Human Computer Interface", McGraw Hill, England, 1991.

[14] Brad A. Myers, et. al., *GARNET - Comprehensive Support for Graphical Highly Interactive User Interfaces*, IEEE COMPUTER, November 1990.

[15] G. E. Pfaff (ed.), *User Interface Management Systems*, Springer-Verlaag, 1985.

[16] John M. Carroll, *The task artifact cycle*, in , J.M.Carroll(ed.), *Designing Interaction: Psychology at the human computer interface*, Cambridge University Press, 1991.

[17] H.R.Hartson and D.Hix, *Human Computer Interface Development: Concepts and Systems for its management* ACM, Computing Surveys, Vol.21, No.1, March 1989.

Appendix 1: A subset of the CSP specification language

The interested reader is encouraged to look at [10], but for reasons of comprehension of this text a small list of the symbols used is presented in this section.

CSP notation is used to describe the behaviour of systems of communicating processes. In the case of the Adept project, the task model will be represented as a collection of processes. A TKS is represented as a process, each made of events. An event is not analyzed any further it is the basic descriptive unit of the language. A process is defined by a process expression, i.e. a definition of how events can follow.

A process expression can be

P the name of a process, (by convention starting with an uppercase letter)

(e -> P) an event prefixing a process, (starting with a lowercase letter)

e1 -> P1 | e2 -> P2 of processes according to events

P[]Q a deterministic choice of processes, i.e. the first event that will occur will determine which of them defines the resulting process behaviour

P[]Q non-deterministic choice. Any of the two operand processes is a valid behaviour for the resulting process. This kind of non-determinism is to be resolved by implementation

P||Q parallel operator. The process is the combination of operand processes, which synchronize on common events and interleave (non-deterministically on non shared events)

P||Q the events of the two processes can be interleaved, i.e. they are distinct, and at any moment an event either of P or of Q can occur.

P !! Q behave as P first then as Q, or as Q first and then as P (unordered sequencing of processes), i.e. P!!Q = (P;Q) [] (Q;P)

STOP Is the termination process. Recognises no events at all.

SKIP A process that successfully terminates before stopping.