Chapter 1

THE INFLUENCE OF IMPROVED TASK MODELS ON DIALOGUES

Anke Dittmar and Peter Forbrig
Department of Computer Science, University of Rostock
Albert-Einstein-Straße 21 – D-18051 Rostock (Germany)
E-Mail: {ad,pforbrig}@informatik.uni-rostock.de
Tel.: +49-381-498 343{2,4} - Fax: +49-381-498 3426

Abstract Model-based approaches support a flexible design process and the development of consistent device-dependent applications. However, their full strength can only be achieved by expressive sub-models allowing to capture conceptual knowledge as early as possible. In this paper we demonstrate how improvements to task models, which can be seen as the heart of model-based techniques, help to develop more appropriate models and prototypes of user interfaces (UIs).

Keywords: Abstract user interface models, Model-based design, Task modelling.

1. INTRODUCTION

The development of user interfaces (UIs) is a co-operative, multi-disciplinary process. In particular, the views of all stakeholders need to be considered and agreements must be found. Model-based approaches accept this nature of design processes as to be seen e.g. in many CADUI-papers. Different declarative models and their relationships are used to represent important aspects of human-computer interaction on a conceptual level. For example, task models describe the actions and the goals of users and their domain knowledge. Abstract and concrete dialogue models are focused on the description of UIs themselves. There can also exist models of users or of specific environmental circumstances (e.g., specific tools and platforms). As a consequence, model-based techniques support the development of design spaces and prevent designers from following "first-solution strategies" too often.

In this paper, we concentrate on formal models about tasks, actions and task domains and their influence on abstract dialogue models and, to some extent, on abstract prototypes of UIs. Section 2 gives background information and points out some limitations of current approaches. Based on a formalization of domain knowledge (3.1) a hybrid notation for describing actions and states is suggested (3.2). This leads to more convenient task descriptions. An exploration of instantiating (3.3) and composing (3.4) actions evokes the reconsideration of temporal descriptions (3.5). Section 4 discusses the influence of the suggested improvements on developing abstract UI models. It is further argued that similar, but refined modelling techniques are applicable to task modelling as well as to dialogue modelling. Section 5 gives some conclusions. The ideas are illustrated by an example and tool support is discussed.

2. FROM TASKS TO DIALOGUES - BACKGROUND

Most techniques which support a task-driven design process of UIs mainly exploit task structures (e.g., [2,10,15,20]). Fig. 1 depicts part of a task model from [10] in CTTE-notation [15]. Task hierarchies, task types and temporal relationships between sub-tasks are used to draw conclusions concerning the structure and the behaviour of corresponding UIs.

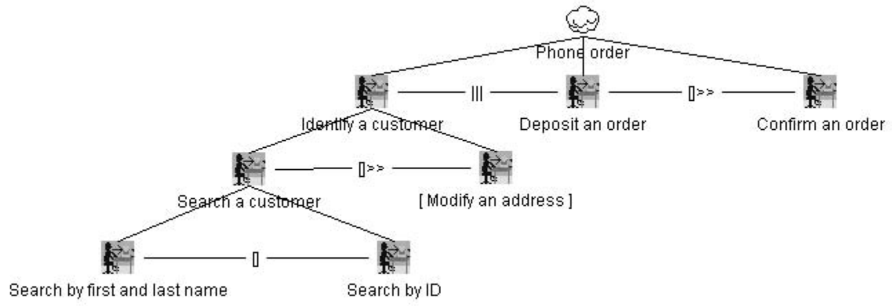


Figure 1. Part of a task model from [10].

A tool we developed to design dialogue models and to generate appropriate abstract prototypes of UIs is illustrated in Fig. 2. The designer can import a task model ((1) in Fig. 2) and assign several dialogue models to it which are based on *dialogue graphs* [5] ((2) in Fig. 2). A dialogue model consists of a set of nodes (views) and a set of transitions. Each view contains a set of elements which are created by mapping tasks to the view ((3) in Fig. 2). A transition is a directed relation between an element of a view and a view. It is distinguished between 5 types of views (single, multi, modal, complex and end view) and 2 types of transitions (sequential and concurrent). A dialogue model allows the generation of an abstract prototype in a WIMP style which can be animated (Fig. 3). Views are represented by windows, their elements by buttons and transitions by navigations between windows. In case of a sequential transition, the source view becomes invisible and the destination view will be visible and active if the associated button is pressed. Concurrent transitions allow for the source view to be visible. The temporal constraints of the underlying task model enable the buttons of abstract prototypes. Inconsistencies between the navigation structure of a dialogue model and the temporal relations between sub-tasks are reported.

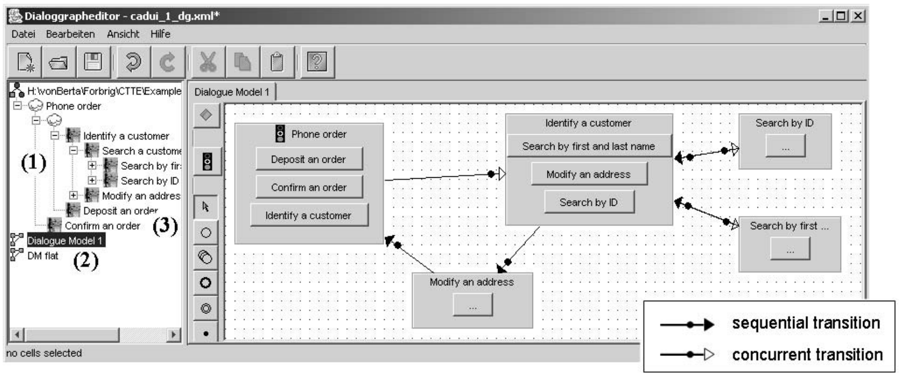


Figure 2. A dialogue graph assigned to Fig. 1.

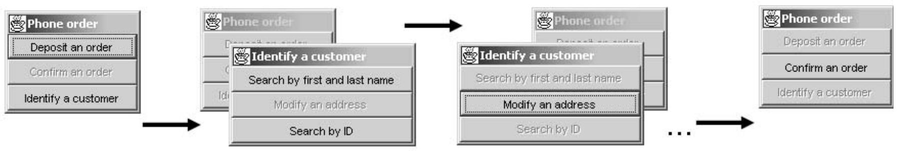


Figure 3. An illustration of the generated abstract prototype of Fig. 2.

Our method is similar to [14] insofar as both approaches make the designer responsible for relating tasks to parts of the UI model and supply some form of model checking. So, we deliberately allowed the designer to decide which part of the conceptual task structure has to be reflected by the transition structure of a dialogue model. In the example, a user cannot execute `Search by ID` until he has "said" the interface (by navigation) that he intends to `Identify a customer` (Fig. 2,3). The sub-task `Search a customer` is not directly mapped. In contrast, a grouping of sub-tasks is not possible in [11]. In [16], a set of task sets (comparable to views in our approach) is generated by applying heuristics to the enabled task sets of a task model and by determining so called transition tasks. Approaches like [12,16] rather try to derive UI models (and abstract prototypes) from task models than to compare them.

Perhaps, many of us would accept Fig. 4b as part of a UI a system could offer its users to perform the task of Fig. 4a. However, there is no hint in the task model telling us that a mail has a sender, a receiver and a content. Although models and abstract prototypes as presented so far can already be used for discussions with stakeholders during early design phases, their limited expressiveness is obvious.

A main reason is the missing "introduction of domain and/or user models along with the task model to derive presentation and dialog models" as stated in [10]. Actually, there are approaches to relate task and domain knowledge in general (e.g., [7]) and for deriving models of UIs (e.g., [1,6,15,17]). However, they are often restricted to specific fields of application or the grade of formalisation is too low.

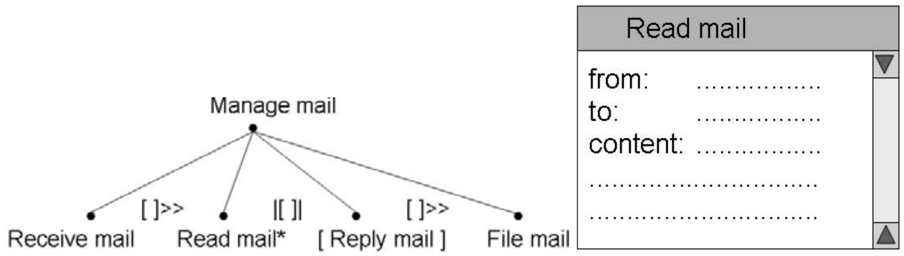


Figure 4. A task model of `Manage mail` (a) and a possible representation for sub-task `Read mail` (b).

Throughout the rest of the paper we will further explore this simplified mailing example to point out some modelling problems at the level of tasks. We will show how a stronger formalisation of task models and even minimal extensions like a new temporal operator can support a more appropriate mapping on abstract UIs. Let us start with the exploitation of formalised domain knowledge as proposed above.

3. IMPROVEMENTS TO TASK MODELS

In [3] we suggest to tighten the link between actions and states within task models. A task is seen as a meta-action which includes a permanent development of models about actions, current states and goal states. For completing a task, humans have to perform a sequence of actions using objects of their environment (in the role of means and resources).

This results in creating or manipulating other objects (in the role of artefacts and side-effects). A deeper discussion can be found in [3]. Additionally, action and task models are defined as objects with a specific structure. We will use this idea in the following section.

3.1 A Formalisation of Domain Knowledge

Objects are specified by sets of attributes (name-value pairs). A pattern-instance relationship is defined between two objects O_P and O_I if there is a subset of attributes in O_P (a pattern schema) so that we can find for each attribute A in this subset an attribute A' with the same name in O_I. The value of A' has to be an instance of the value of A. Consequently, an object can be characterized intentionally by its attribute structure or extensionally by its actual set of instances. In addition, partial descriptions can be introduced to describe special subsets of instances of a (pattern) object. This can be expressed by partial equations.

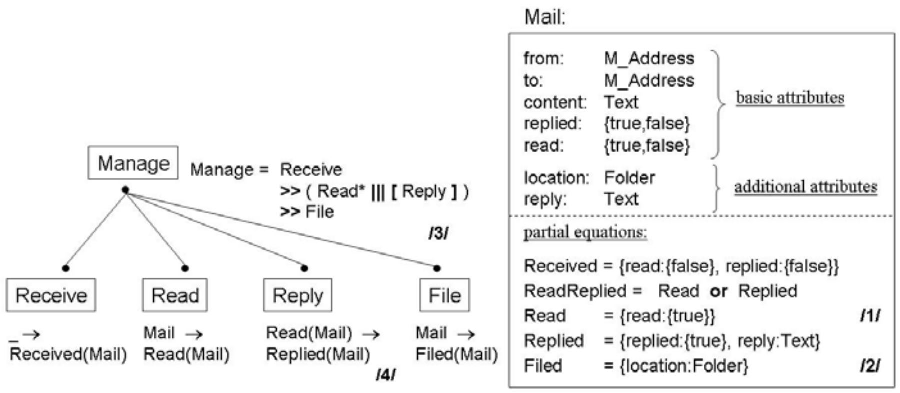


Figure 5. Action `Manage` enriched by preconditions and effects on artefact `Mail` (denoted by Pre→Eff) and the object description of `Mail`.

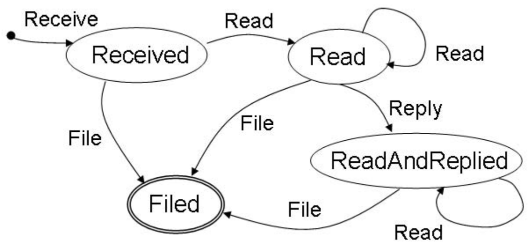
Object `Mail` (right part of Fig. 5) illustrates the proposed specification style. It represents a pattern for mail messages. There are partial descriptions specifying subsets of mail messages (instances) which are received, read, replied or filed by restricting ranges of attribute values (e.g., `/1/`) and introducing additional attributes (e.g., `/2/`). Furthermore, partial descriptions can be composed of others by using special operators. For example, `ReadReplied` describes all mail messages which are instances of `Read` or `Replied`. Hence, the operator `or` works on states.

The left part of Fig. 5 shows the enriched action model of Fig. 4a (outside an object *O* a partial description *PD* of *O* is denoted by *PD(O)*). For reasons of brevity, the artefact role is only considered in this paper. As mentioned earlier, actions are special objects. Their hierarchical decomposition is reflected in their sets of partial descriptions. The partial equation of a basic action contains predefined operations. For example, we support basic arithmetic and string operations, operations to read and write attribute values and to create and delete additional attributes. The partial equation of a non-basic action specifies the temporal relations between its sub-actions by using temporal operators. Tool support for creating and animating action models is illustrated in the appendix. However, some further points are worthy of mention regarding the example.

3.2 A Hybrid Notation for Action Models

Fig. 4a defines a behaviour which was not intended. Fig. 6 shows a STN with the desired one instead and an equivalent temporal equation. A temporal description in CTTE-notation is even more complex because the completion of iterative actions must be described explicitly. In addition, 'artificial' nodes must be introduced which destroy the conceptual hierarchical structure of the model. A hybrid notation as used in Fig. 5 unifies structural and procedural knowledge and can lead to more concise and, possibly, to more 'natural' descriptions than pure state or temporal notations.

The temporal equation `/3/` guarantees that a mail message has been received before it can be read, replied, or filed. We do not need to specify this in the preconditions of the actions `Read`, `Reply`, and `File`. On the other hand, we can express knowledge about requested and resulting states of objects in preconditions and effects to shorten temporal descriptions. An example is the constraint that a mail message has to be an instance of the partial description `Read` (`/4/`) in order to reply to it. Of course, consistency checks of such hybrid models are useful, but their consideration is beyond the present paper.



```
Manage =
    Receive >>
    ( File
      [ ] Read >>
          ( Read* ||| [ Reply ] ) >>
          File
    )
```

Figure 6. A STN with the desired behaviour and an equivalent temporal equation.

3.3 Two-fold Instantiation of Actions

Action models are patterns in a two-fold way. They describe a set of possible execution sequences of basic actions. In addition, if a pattern object occurs several times in the preconditions and the effects of an action tree it can be replaced consistently by the same instance. Necessary pattern names can be indexed (e.g., `Mail[1]`, `Mail[2]`,...) to distinguish between several instances currently used (Section 3.5). An instantiation for Fig. 5 can be found in the appendix.

3.4 Action Composition

As Paul entered his office on Friday morning the phone was ringing. "Hi Paul. Here's Stephen. Could you do me a favour and send your source code for checking the consistency of action models? Could you send some documentation as well? I have a problem that seems to be quite similar. I've sent you mail. Could you take a look at it? It's urgent!" Besides Stephen's mail Paul has got 7 other messages. And before he started to work on Stephen's mail he had to see what Ann has written...

An action model describes sequences of (unconsciously executed) operations under actual conditions of the domain [8]. Fortunately, humans cannot plan their whole activity by only one action. In other words, actions are situated and goal-directed. So, in the scenario above Paul obviously opened his mailbox to read and answer Stephen's mail immediately. Why did he react to Ann's mail in the next moment?

As mentioned before, we regard a task as a meta-action involving permanent adjustments to situations, goals, and actions. Paul had to combine, for example, his action models concerning the treatment of Stephen and Ann. Within a task-oriented approach we try to describe by the envisioned task model all those tasks which are supposed to be supported by the interactive system under design. In [9], the term (Target) Composite Task Model is used. The scenario reveals a phenomenon that often occurs when combining action models. The resulting structure is not necessarily a concurrent composition of the single actions. Instead, new actions are introduced to replace a set of sub-actions from different models. Such replacements can cause further restructurings, the creation of auxiliary sub-actions etc.

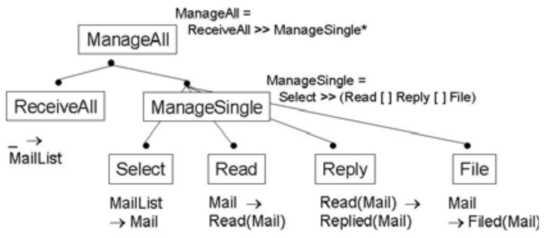


Figure 7. A composition of actions.

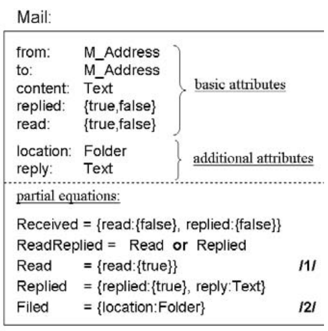


Figure 8. Paul has to reply to mail from Ann immediately…

Fig. 7 shows an action composition for managing many mail messages which is a more appropriate abstraction from the scenario as e.g. `Manage` ||| `Manage` ||| ... because Paul received several mail messages at once. In fact, every model has to handle different scenarios or lower-level abstractions.

Even so it has `Manage` (Fig. 4a). But, how can we describe, for example, that Paul has to reply immediately to all mail from Ann? Fig. 8 indicates two possible solutions. So far, only actions were informed about objects they use or manipulate.

We extend this uni-directional relation to a bi-directional one. In this case, an attribute `take_part:Manage` is added to the (artefact) object `Mail`. The partial description `FromAnnToPaul_Active(Mail)` determines a subset of sequences of basic actions of `Manage`. $A>>_T B$ means that `B` has to be started immediately after finishing `A`. Hence, if Paul has received mail from Ann his behavioural description is restricted in the intended way. `FromAnnToPaul_Passive(Mail)` specifies an order of states which mail from Ann to Paul has to go through.

Why can it be useful to 'source out' action knowledge to objects acted upon? We can argue in the same way as in Section 3.2 and even our simple example shows some advantages. If partial descriptions of actions restrict the kind of acting on single objects, it is often more convenient to attach these constraints to the objects themselves. Therefore, we could also weaken the preconditions in Fig. 7 because attribute `take_part` in Fig. 8 already says how to manage a single mail. In other words, `take_part` "saved" the knowledge about `Manage` from Fig. 5 into Fig. 7.

We are well aware that interactive systems have to support users in performing tasks in a variety of combinations under certain social, organizational and timing constraints. Before temporal operators are reconsidered in the next section, we would like to point out some effects of task composition. Roughly speaking, an action `A` with precondition `A` and effect `E` (denoted by `A{C→E}`) is described by a composed action `CA{CC→CE}` if `CC` guarantees `C` and `E` is achieved by `CE`. However, not only the effect is interesting but also an efficient execution. Let us take the action `ManageSingle*` of Fig. 7, which allows for an interleaved managing of mail.

It specifies sequences like ⟨`Select{ml→ml},Read{ml→ml},Select{ml→m2},Read{m2→m2},Select{ml→m2},Reply{m2→m2},…`⟩ where `ml` is an instance of `Mail-List`, `m1` and `m2` are instances of `Mail`. To observe the work on one particular mail message we hide all actions from execution sequences dealing with other mail and we get sequences like ⟨`Select{ml→m},Read{m→m},Select{ml→m},Reply{m→m},Select{ml→m},…`⟩. It is easy to see that the following modification of `ManageAll` gives better results (preconditions and effects are omitted).

```
ManageSingle   = Select >> ManageSelected*
ManageSelected = Read [] Reply [] File
```

3.5 Reconsideration of Temporal Operators

The *|-operator. Temporal operators serve to describe some basic composition “patterns'” of actions or tasks. The set of operators offered by CTTE [15] is reproduced in Table 1.

Table 1. Temporal Operators of ConcurTaskTree notation.

[ ]	Choice	|>	Suspend & Resume
| - |	Order independency	>>	Enabling
| | |	Concurrency	>>[ ]	Enabling with information exchange
| | | [ ]	Concurrency with information exchange	*	Iteration
[>	Disabling	[…]	Optional

With the two-fold instantiation as introduced in Section 3.3, operators with information exchange are superfluous. By assigning preconditions and effects, and by following the instantiation rules, information exchange between sub-actions can even be described more precisely. However, it also becomes obvious that we need more than the *-operator for action repetition. In Fig. ?, one action instance of `ManageSingle` has to be completed entirely before a second one can be started. In reality, humans often perform similar actions concurrently. A new operator *| is introduced to describe finite sets of concurrent instances of actions. It is inspired by the replication operator ! as known from the π-calculus [12] ($!P \equiv P \mid !P$). Thus, `ManageSingle`*| allows an action instance like ⟨`Select{m1→m1},Select {m1→m2},Read {m1→m1},Read{m2→m2},…`⟩. Additionally, in order to explain the full semantics of action repetition, a closer look at the attached objects is necessary. It makes a difference whether we read mail again and again (Fig. 5) or whether we repeat `ManageSingle` from Fig. 7 on different mail messages. Fig. 9 illustrates two common instantiation patterns which generalize above examples. For simplicity, we abstract from partial descriptions of object `O` in preconditions and effects. **B**`O` can be instantiated by an arbitrary set of instances of `O`.

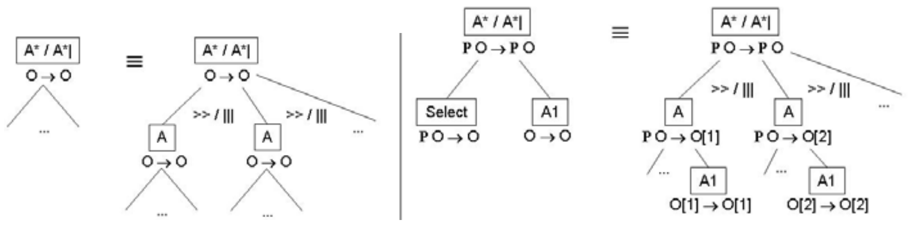


Figure 9. Two instantiation patterns for repetitive actions.

The $>>_T$-operator. In [4], the distinction between task enabling and triggering is discussed. Triggers are proposed because they describe when actions finally occur. The $>>_T$-operator as introduced in the previous section could be useful for describing the immediate trigger type and, also, for describing environmental cues if $>>_T$ occurs in objects of the task domain.

4. FROM TASKS TO DIALOGUE MODELS

While the focus of task models is more on the description of user needs abstract dialogue models focus on UIs (that is to say on the description of the artefact itself) [18]. As we have shown in Section 2, much of the requirements on dialogue models can (and should) be derived from task models. For example, the behaviour of a dialogue model has to obey the temporal constraints of its task model. Temporal descriptions of tasks can be mapped now more precisely on dialogue models, a conclusion we can draw directly from the previous section. So, the dialogue of Fig. 10 which allows for work on several mail messages concurrently would not be possible for the task `ManageAll` (Fig. 7) until `ManageSingle` * is replaced by `ManageSingle` *| in the temporal equation of `ManageAll`.

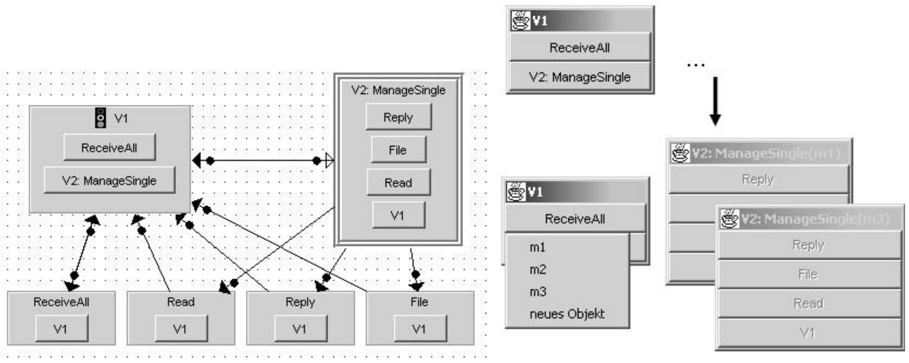
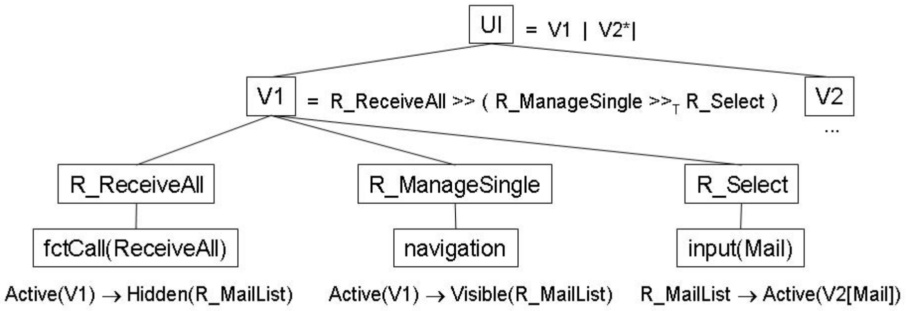


Figure 10. A dialogue graph for similar concurrent sub-actions (`V2` is a multiple view) and two screenshots of the generated abstract UI-prototype.

However, we can go one step further. The formalization of domain knowledge also supports a more precise mapping on presentations. Furthermore, an abstract dialogue model (= behaviour + presentation) can be considered as a specific action model in a specific domain represented by Abstract Interaction Objects (AIOs) [19]. Although mail in Fig. 7 can refer to ‘conventional’ letters we are particularly interested in the subset of email messages and an appropriate software tool to manage them. A specialty of interactive systems is that their UIs represent both tools (functionality) and artefacts. Consequently, AIOs have to be assigned to actions and objects of the task model or to groups of them in order to specify a UI. For the description of AIOs, the same mechanisms as introduced in Section 3 can be used.

It is argued in [10] that different notations for task models and UI models are of advantage because these models are built by different people and are used independently. However, looking at object-oriented concepts and notations, for example, they are applied in very different context AIOs specify at an abstract level which parts of the underlying objects are visible, enabled, or active at which stage of execution. Abstract user actions are represented by function calls, navigation and data input. We can take a look at the example of Fig. 7 to roughly illustrate this idea. We should especially note the mapping of actions to view `V1` (Fig. 10). For simplicity, an AIO associated to an object `O` is denoted by `R_O`. `R_A` is enabled/disabled if action `A` is enabled/disabled in the underlying action model.



UI = V1 | V2*|
V1 = R_ReceiveAll >> (R_ManageSingle >>T R_Select)
V2
...
R_ReceiveAll
R_ManageSingle
R_Select
fctCall(ReceiveAll)
navigation
input(Mail)
Active(V1) → Hidden(R_MailList)
Active(V1) → Visible(R_MailList)
R_MailList → Active(V2[Mail])

Figure 11. Part of an abstract user interface model.

Fig. 11 shows part of an appropriate abstract UI model. For example, the partial description `Active(V1)` in the precondition of `fctCall(ReceiveAll)` defines that view `V1` has to be active. By executing `fctCall-(ReceiveAll)`, action `ReceiveAll` is completed. This enables the complex action `ManageSingle` in the action model and, thus, the associated AIO `R_ManageSingle` is enabled. If `V1` and `V2` are instantiated by windows, `R_ReceiveAll` and `R_ManageSingle` by buttons and `R_MailList` by a list of buttons, we get a presentation similar to the abstract prototype of Fig. 10. However, the UI description is only complete in connection with the underlying action model. Again, a hybrid notational approach was applied.

5. CONCLUSION

The pros and cons of using formal models to describe human work and, in particular, human-computer interaction are well known. On the one hand, formal models increase the guarantee that task knowledge and decisions made at a conceptual level are reflected in the final implementation of UIs. On the other hand, formal models are often rejected because they are "inappropriate for capturing the complexity of human experience and activity"

[4]. This is certainly true for at least two reasons. Often, formal models overemphasize a certain perspective on the subject of interest. Furthermore, some elements of the domain are not included in models but should be.

In this paper, an enriched, but more formalised description of tasks is proposed. However, the suggested hybrid notation of actions can ease modelling activities. It was shown that these improvements to task models (even a small modification like the introduction of a new temporal operator) allow a more precise mapping on abstract UI models. We provide prototypical tool support for some of our ideas, but there is still much to do. For example, domain knowledge is not fully exploited to derive abstract presentations.

REFERENCES

[1] Barclay, P.J., Griffiths, T., McKirdy, J., Paton, N.W., Cooper, R., and Kennedy, J., *The Teallach Tool: Using Models for Flexible User Interface Design*, in A. Puerta, J. Vanderdonckt (eds.), Proceedings of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999), Kluwer Academics Pub., Dordrecht, 1999, pp. 139-157.

[2] Dittmar, A. and Forbrig, P., *Methodological and Tool Support for a Task-Oriented Development of Interactive Systems*, in A. Puerta, J. Vanderdonckt (eds.), Proceedings of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999), Kluwer Academics Pub., Dordrecht, 1999, pp. 271-274.

[3] Dittmar, A., and Forbrig, P., *Higher-order task models*, in J. Jorge, N.J. Nunes, J. Falcão e Cunha (eds.), Proc. of 10th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Funchal, June 2003), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 187-202.

[4] Dix, A., *Managing the Ecology of Interaction*, in C. Pribeanu, J. Vanderdonckt (eds.), Proc. of 1st Int. Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2002 (Bucharest, 18-19 July 2002), Academy of Economic Studies of Bucharest, Economic Informatics Department, INFOREC Printing House, Bucarest, 2002, pp. 7-9.

[5] Elwert, T. and Schlungbaum, E., *Dialogue Graphs-A Formal and Visual Specification Technique for Dialogue Modelling*, in J.L. Siddiqi, C.R. Roast (eds.), Proceedings of the BCS-FACS Workshop on Formal Aspects of the Human Computer Interface FAHCI'96 (Sheffield, 10-12 September 1996), accessible at http://ewic.bcs.org/conferences/1996/formalaspects/papers/paper13.pdf

[6] Gamboa-Rodriguez, F. and Scapin, D., *Editing MAD* Task Descriptions for Specifying User Interfaces at both Semantic and Presentation Levels*, in Proc. of 4th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'97 (Granada, 4-6 June 1997), Springer-Verlag, Vienna, 1997, pp. 193-208.

[7] Johnson, P. and Wilson, S., *A Framework for Task-based Design*, in Proceedings of the 2nd Czech-British Symposium on Visual Aspects in Man-Machine Systems VAMMS'93 (Prague, 24-28 March 1993), Ellis Horwood, Chichester, 1993.

[8] Leontiev, A.N., *Activity, Consciousness, Personality*, Prentice Hall, Englewood Cliffs, 1978.

[9] Lim, K.Y. and Long, J., *The MUSE Method for Usability Engineering*, Cambridge University Press, Cambridge, 1994.

[10] Limbourg, Q., Vanderdonckt, J., and Souchon, N., *The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results*, in F. Paternò, Ph. Palanque (eds.), Proc. of 7th Int. Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, 5-6 June 2000), Lecture Notes in Computer Science, Vol. 1946, Springer-Verlag, Berlin, 2000, pp. 227-246.

[11] Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J., *Derivation of a Dialog Model from a Task Model by Activity Chain Extraction*, in J. Jorge, N.J. Nunes, J. Falcão e Cunha, J. (eds.), Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 203-217.

[12] Mori, G., Paternò, F., and Santoro, C., *CTTE: Support for Developing and Analysing Task Models for Interactive System Design*, IEEE Transactions on Software Engineering, Vol. 28, No. 8, August 2002, pp. 797-813. Accessible at http://giove.cnuce.cnr.it/ctte.html.

[13] Milner, R., *Communicating and Mobile Systems: the π-Calculus*, Cambridge University Press, Cambridge, 1999.

[14] Navarre, D., Palanque, P., Paternò, F., Santoro, C., and Bastide, R., *A Tool Suite for Integrating Task and System Models through Scenarios*, in C. Johnson (ed.), Proc. of 8th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2001 (Glasgow, 13-15 June 2001), Lecture Notes in Computer Science, Vol 2220, Springer-Vrlag, Berlin, pp. 88-113.

[15] Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*, Springer-Verlag, Berlin, 2000.

[16] Paternò, F., and Santoro, C., *One Model, Many Interfaces*, in Ch. Kolski, J. Vanderdonckt (eds.), Proc. of 4th International Conference on Computer-Aided Design of User Interfaces (Valenciennes, 15-17 May 2002), Kluwer Academics Publishers, Dordrecht, 2002, pp. 143-154.

[17] Puerta, A., Cheng, E., Ou, T., and Min, J., *MOBILE: User-Centered Interface Building*, in Proceedings of ACM Conf. on Human Aspects in computing Systems CHI'99 (Pittsburgh, 15-20 May 1999), ACM Press, New York, 1999, pp. 426-433.

[18] Traetteberg, H., *Model-based User Interface Design*, Ph.D. thesis, Mathematics and Electrical Engineering, Faculty of Information Technology, Norwegian University of Science and Technology, 2002.

[19] Vanderdonckt, J. and Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.

[20] Vanderdonckt, J., Limbourg, Q., and Florins, M., *Deriving the Navigational Structure of a User Interface*, in M. Rauterberg, M. Menozzi, J. Wesson (eds.), Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003), IOS Press, Amsterdam, 2003, pp. 455-462.

APPENDIX: TOOL SUPPORT FOR TASK MODELLING

Task models can also be animated (but in a command-line style only). For reasons of clarity the output of the following scenario ⟨`Receive,Read,Reply,Read,File`⟩ has been shortened. Actions executable in the next step of the animation and the actual state of the instance `mail1` are shown.

```
Parsing samples/mail/mail1.in...okay
Possible actions:
(1) [Manage, Receive]
Enter a number between 1 and 1 (0 to exit): 1
Environment:  [1]  Mail  -  replied=false  read=false  con-
tent="hallo paul..."
to="Paul" from="Ann"
Possible actions:
(1) [Manage, Read]
(2) [Manage, \_Read]            Enter...: 1
Environment: [1] Mail - read=true replied=false ...
%Possible actions:
(1) [Manage, Read]
(2) [Manage, \_Read]
(3) [Manage, Reply]            Enter...: 3
Environment:  [1]  Mail  -  replied=true  read=true  reply="hallo
Ann"
...
%Possible actions:
(1) [Manage, Read]
(2) [Manage, \_Read]            Enter...: 1
Environment: [1] Mail - read=true ...
%Possible actions:
(1) [Manage, Read]
(2) [Manage, \_Read]            Enter...: 2
Environment: [1] Mail - read=true ...
%Possible actions:
(1) [Manage, File]             Enter...: 1
Environment:  [1]  Mail  -  read=true  replied=true  content="hallo
paul..."
to="Paul" from="Ann" location="trash" reply="hallo Ann"
Action finished
```