Appropriateness of User Interfaces to Tasks

Sandrine Balbo
Department of Information Systems
The University of Melbourne
Parkville VIC 3010, Australia
sandrine@unimelb.edu.au

Dirk Draheim
Institute of Computer Science
Freie Universität Berlin
Takustr.9, 14195 Berlin, Germany
draheim@acm.org

Christof Lutteroth, Gerald Weber
Department of Computer Science
The University of Auckland
38 Princes Street, Auckland 1020, New Zealand
{lutteroth, g.weber}@cs.auckland.ac.nz

ABSTRACT

In this paper we define the complex relation between task models and user interfaces in a declarative manner. We do this by describing how a task model can be transformed to other functionally equivalent task models, how it can be mapped to a user interface prototype, and how a user interface can be transformed to other functionally equivalent user interfaces. We use this relation in order to tackle the question whether a user interface is appropriate for a task, which leads us to a conceptual notion of usability. The user interfaces are modeled with form-oriented analysis.

Keywords

task models, use cases, form-oriented analysis, model-based user interface development (MB-UID), usability

1. INTRODUCTION

In tool-building approaches like MB-UID, mappings from task models to user interfaces have been well discussed. A typical tool is in essence a function, mapping one task model to one user interface, perhaps with customization options. For investigations about usability of existing user interfaces, however, such a forward engineering tool is not enough. For the question of whether a given user interface is appropriate for a given task, the relation between both has to be understood on a conceptual level. The main contribution of this paper is a *defined* relation between task models and user interfaces, which results in a workable definition of *support for a task*. This approach goes beyond tool-building approaches that deliver by default mainly operational semantics.

Within user interface modeling, task models are employed as a user centric model of interaction. On top of this we need a model of the user interface that is on the same level of abstraction as task models. If we then have a semantic framework that combines both models, a user interface model and a task model, then we can align them, compare them and reengineer them. In this paper we focus on form-oriented user interfaces since they are widespread in enterprise computing, sufficiently flexible to deal with the business logic, and there is a well-understood formal description of them with form-oriented analysis [4].

The relation between task models and user interfaces is not a simple one-to-one mapping, which explains why it cannot be captured by a naive tool approach. The reason for the ambiguity of that mapping lies in the fact that there are many task models and many user interfaces which are functional equivalent. Many equivalent task models correspond to many possible user interfaces, as the left part of Fig. 1 illustrates. In order to reduce the complexity of this relation we first discuss transformations on task models which preserve functionality in Sect. 2. We describe one possible mapping from a task model to a prototypical user interface. With this particular one-to-one mapping the sets of functionally equivalent task models and user interfaces are linked, and the overall relation between task models and user interfaces is defined. These transformations define the set of task models which are all supported by the same set of user interfaces. Then we describe functionality preserving transformations of user interfaces in Sect. 4. These, in turn, define the set of user interfaces which support the same set of tasks. This is illustrated in the right part of Fig. 1. Eventually, we suggest in Sect. 5 how this relation can be utilized in order to determine whether a task is supported by a user interface.

2. CONSERVATIVE TRANSFORMATIONS FOR TASK MODELS

Most task models use a concept of hierarchical decomposition [1], like, for example, the one in Fig. 2, which is a MAD-like task model [14]. While hierarchical decomposition allows the modeller to structure a task on different levels of detail, it does not change the task itself. If we consider, for example, the task "find the right flight" on the 4th level of the given task model, it would be possible to eliminate the level containing its subtasks by substitution. This is possible because the task order on both levels is sequential, indicated by the "SEQ" keyword, so that the two task sequences can be merged. The same is possible with two adjacent levels of parallel tasks. One of the properties of sequential and

TAMODIA'05, September 26–27, 2005, Gdansk, Poland.

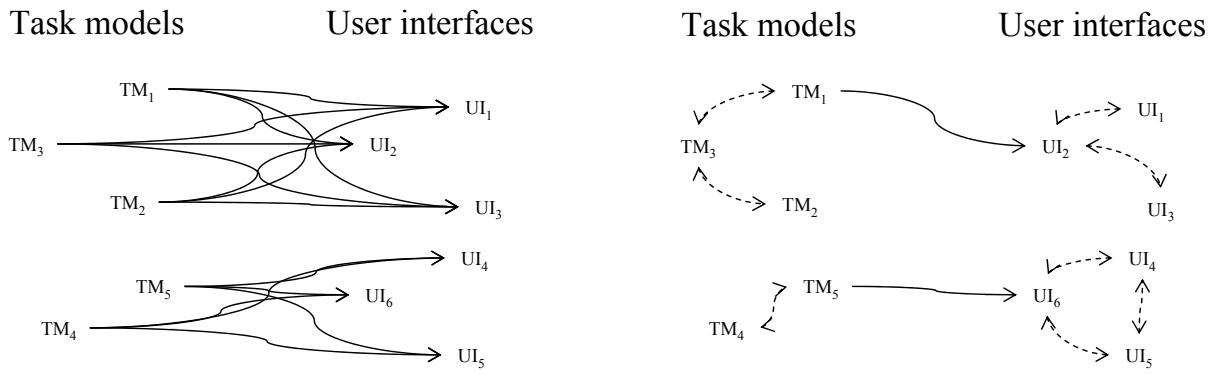

Figure 1: Direct mapping of task models to user interfaces and indirect mapping via prototype and conservative transformations.

parallel decomposition is associativity, which means that we can decompose three sequential or parallel tasks by moving the first two tasks or the last two tasks onto a subordinate level with both decompositions being functionally equivalent. The parallel decomposition is commutative, i.e., it is possible to change the order of the parallel tasks. To sum up, as long as we do not change the order of tasks in the model, which we could only if a sequential order is specified, decomposition of a task model is arbitrary in the sense that it does not change the functional properties of the task. Hence, different decompositions of the same task model can be considered functionally equivalent.

Within a task modeling approach as well as within a user interface modelling approach we can observe a partial order of models that fulfil the notion of specialization and generalization. Task model A is a specialization of task model B if scenarios according to A are also scenarios according to B. One specialization process is turning parallelism into sequences, another is the refinement of tasks. Parallelism is more general than sequence, but an even more general model can be defined with cardinalities. In task models like MAD, tasks can get cardinalities indicating repetitions. The most general decomposition of one task into several subtasks is the parallel decomposition with 0..* cardinalities.

3. MAPPING TASK MODELS TO USER INTERFACES

This section describes one possibility for mapping a task model to a form-oriented model. We need this one-to-one mapping as a link for the relation between task models and user interfaces in order to connect each set of functionally equivalent task models with the corresponding set of functionally equivalent user interfaces. We can use any mapping here as long as a task model is mapped to one of the user interfaces of the corresponding set. We chose a simple and straihtforward mapping, which we want to call *prototypical mapping*. The user interface mapped to a task model is called *user interface prototype*.

In the task model of Fig. 2 we distinguish three different types of tasks: bubbles represent *user tasks*, boxes represent *system tasks*, and rounded boxes represent *interactive tasks*, which involve actions from both the user and the system. Note that our definition of user task differs from the one in the MAD model: while in MAD bubbles represent purely manual tasks, we use the bubbles for any task in which the user acts without the system passing over into a distinctly new state. The user may, for example, read data from the screen, perform some manual activity and then enter data into the system. The translation of user tasks and system tasks is straightforward: user tasks are mapped to *client pages*, i.e., pages of information that present information and offer forms to enter data. The page corresponding to a user task may show some information about the task or offer the user possibilities to give a feedback. System tasks are mapped to *server actions*, i.e., processes happening within the system. They describe something that the system does without involvement of the user. In the following we will call client pages and server actions simply pages and actions. Interactive tasks are special because they can be decomposed into a sequence of user tasks and system tasks and hence leave the system partly unspecified. Consequently, the mapping of interactive tasks into the form-oriented model is not well-defined. If we do not choose to refine the task model, we can simply map an interactive task to an action-page-action sequences, with the first action retrieving the information that is needed for the user to perform the task, the page allowing the user to enter the result of his/her part, and the last action allowing the system to react to whatever input the user made. Note that an important property of a form-oriented user interface model is its *bipartiteness*, meaning that each page is always followed by an action and each action generates a following page. In order to preserve the bipartiteness, we need to insert an action if we have two user tasks in a row, which would map to two pages in a row, and merge actions if we would have two actions in a row, which could emerge if there are two consecutive system tasks or interactive tasks in the task model. The additional action between two pages allows the user to navigate from one page, representing one user task, to the next. The merging of consecutive actions does not affect the user interface at all, since they represent tasks performed solely by the system.

Figure 3 shows a mapping of the task model in Fig. 2 to the visualization of a form-oriented user interface model which we call a *formchart*. In order to make the illustration of the mapping easier, we first transformed the task model

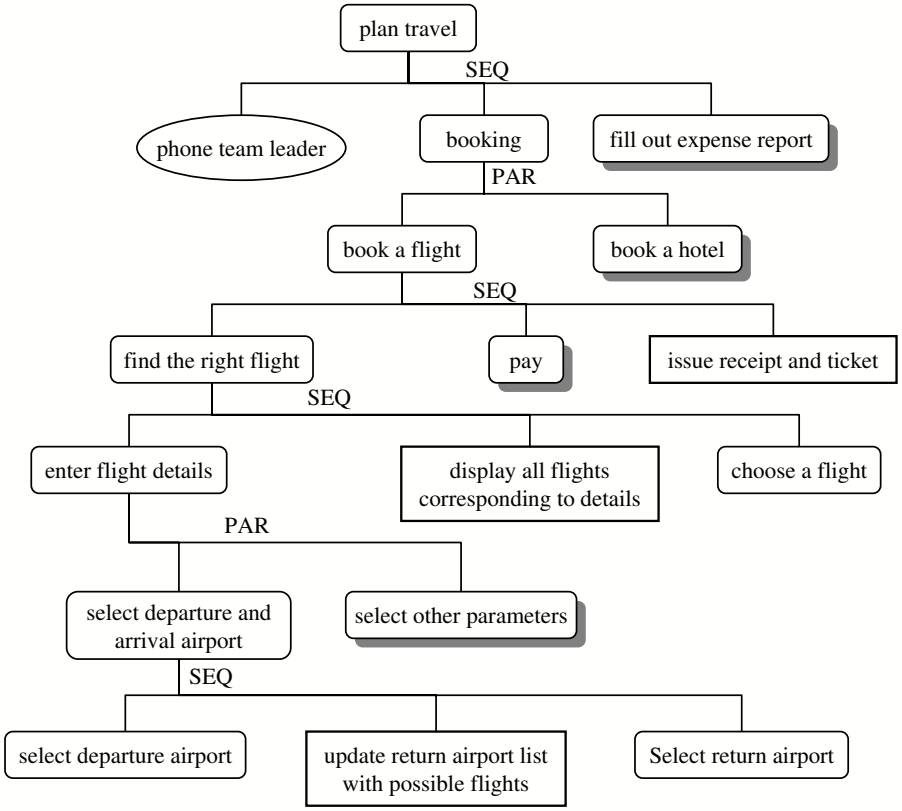

Figure 2: Hierarchicalal task model of a travel planning task.

into a workflow diagram, which is similar to an event-driven process chain [15]. Part of the mapping is visualized by dotted lines. We see, for example, that parallelism has been modelled with non-modal interaction, which we will discuss in the following section. We also see that the user interface is incomplete, which is due to the fact that there are references to submodels in the task model in Fig. 2 which are not defined in this paper. This is due to space considerations. The shapes representing submodels have a shadow, and the parts of the user interface corresponding to these shapes are missing. The missing parts are not really a problem because the rules of formchart decomposition allow us to complete the formchart without having to change the parts that we already have.

3.1 Modelling Parallelism in the User Interface

For the mapping of parallel tasks there are a number of different alternatives. We discuss specifically the following alternatives: sequential realization, todo-list realization and non-modal realization. In the sequential realization one sequence of the parallel tasks is chosen. The transformation to a sequential model can be done already in the task model. This operation is based on a specialization property of task models and we will discuss this specialization property. The todo-list realization allows the user to pursue the different parallel tasks and keep track on which tasks are already completed with the help of a system-given todo-list. This realization motivates to discuss the fact that task models may describe workload management and personal organizer aspects that may not be considered user interface aspects. Finally, parallel tasks can be translated into a stateless choice for the user. This is an instance of non-modality and therefore this solution is called the non-modal realization. This solution addresses another fundamental issue and this is the question whether the user interface is supposed to enforce sequentiality of the task model.

One of the common features in a task model is the ability to specify parallelism of tasks. In the form-oriented model, there is no explicit parallelism, but there are different ways to deal with the fact that the order in which tasks are performed is not specified. Consider three tasks A, B and C which are parallelised in the task model. The simplest way to combine the subdialogues corresponding to A, B and C respectively is to arrange them in an arbitrary order. This approach is restrictive since the task model does not impose any order, but it is valid since if the order was relevant, it would have been defined in the task model. Another way is to model A, B and C as a *non-modal interaction*. This means that we have a single page from which all the subdialogues are accessible, and that we do not restrict the order in which these subdialogues are visited. However, this approach also does not prevent the user from visiting the same subdialogue more than once, which might conflict with the task model. This can be fixed by imposing dialogue constraints that make sure that each subdialogue is only visited once. In this approach, which we call *todo-list pattern*, the order in which the subdialogues are visited is still up to the user,

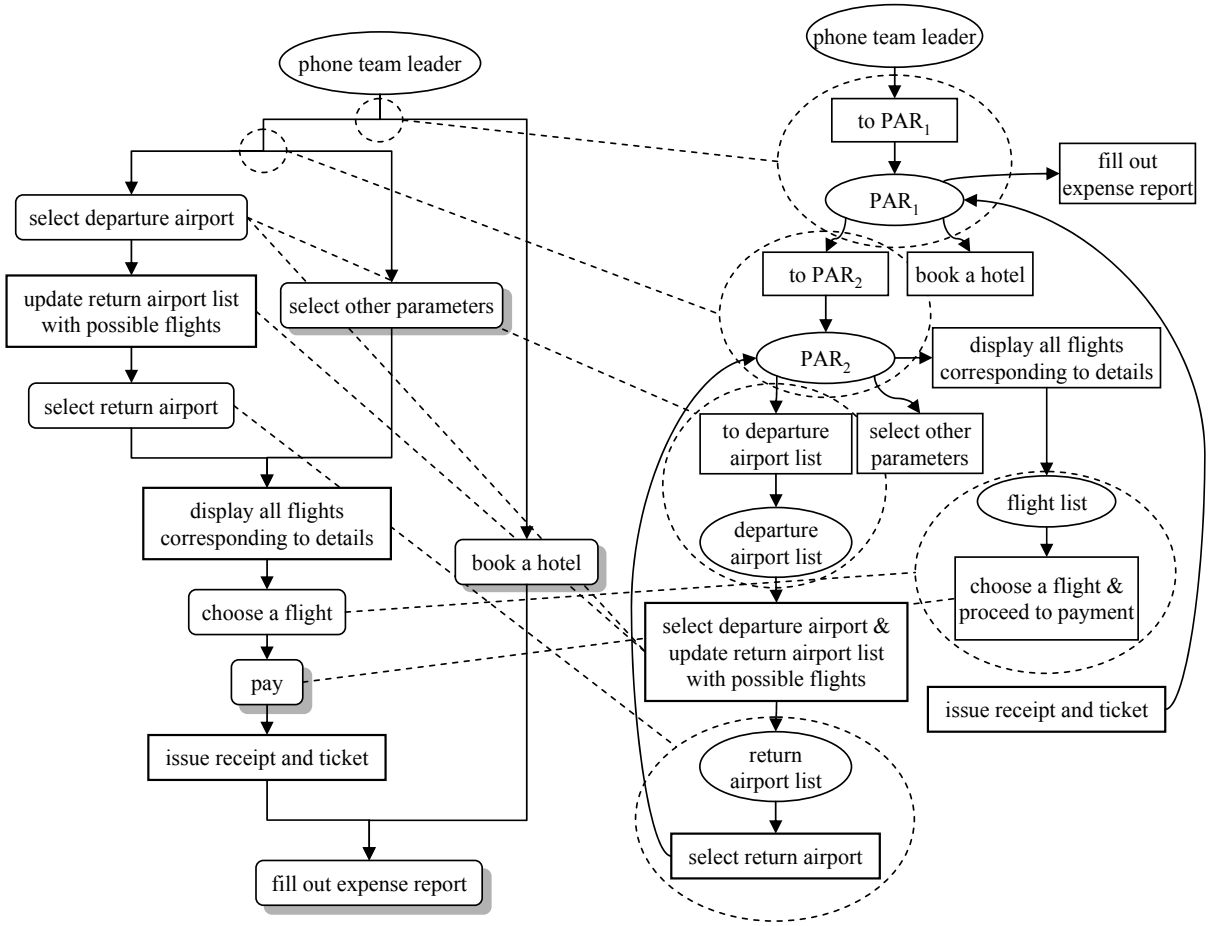

Figure 3: Mapping of workflow diagram to formchart.

therefore the todo-list pattern combines the advantages of an arbitrary sequence and non-modal interaction. Figure 4 shows the sequence approach and the todo-list pattern. The formchart for non-modal interaction is the same as the one for the todo-list, only that there are no dialogue constraints.

In the task model in Fig. 2 we used different shapes for different subtasks depending on the degree of user involvement. User tasks are represented by bubbles, system tasks by rectangles and interactive tasks by rounded rectangles. For user tasks and system tasks the mapping to the user interface is simple: a user task maps to a single page and a system task to a single action. Interactive tasks, however, cannot be mapped to a user interface that easy. This is because they are underspecified, i.e., they can be decomposed into user tasks and system tasks. Once this decomposition is done, the mapping is straightforward. Note that it can sometimes be necessary to insert additional actions or merge existing ones. If the direct translation yields multiple pages in a row, we put extra actions between them, so that the user can navigate them sequentially. If we have multiple actions in a row, these actions have to be merged into a single action in order to preserve bipartiteness.

3.2 Conditional System Response and User Reaction

The mapping described in the previous sections results in a minimalistic user interface which lacks certain features that are desirable in a good user interface. This is because the task models provided for a system are in general only main streams of system usage, where, for example, error conditions are barely represented [11]. The envisaged user interface should be a natural generalization of the provided task models, which can be attained, for example, by conjunction with other models. This section is about two phenomena in user interfaces, *conditional system response* and *user reaction*, which are usually not captured in a task model.

Typical examples for conditional system behaviour are special behaviour for error and exception handling. Errors are usually triggered by erroneous user input, like incorrect passwords or missing data, and lead in most cases to a description of the error together with an opportunity to re-enter the data. Since the user can enter incorrect data an arbitrary number of times, a user interface model describing this behaviour will usually contain parts of cyclic nature. Apart from direct user input, many systems also rely on other sources of data when processing user requests. This can be, for example, data about the availability of some kind

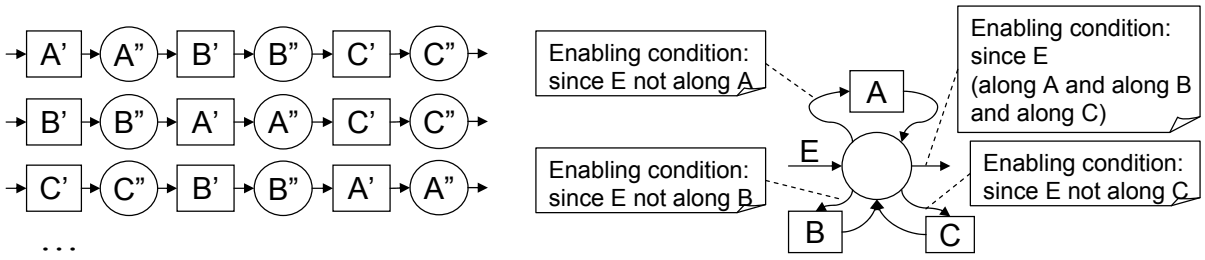

Figure 4: Modelling parallelism as arbitrary sequence, todo-list or non-modal interaction.

of resources, like flights, theatre tickets or other commercial articles. The cases where the resource requested by the user is unavailable or the request of the user is ambiguous are exceptions and are handled by the system in separate branches of the respective single user interface. Such branches can, for example, suggest possible alternatives or try to resolve ambiguities. In a user interface model, conditional system behaviour is reflected in the presence of multiple outgoing transitions on server actions, which handle the different cases that are distinguished in the system. Consider, for example, a system for a cinema: if the user tries to buy tickets for a show that is already sold out, the system will notify the user and may suggest a list of similar shows to proceed. Note that it is technically not feasible to always present the user with an up-to-date list of shows and available tickets, so as to rule out any unsuccessful ticket purchase attempts. This is due to the fact that most B2C systems make use of the Internet, which is a pull medium: when the user requests the most up-to-date version of available shows, he or she will get it, but during the interval between the system responding with an up-to-date version and the user making a choice, tickets may have been purchased by other customers.

Apart from choosing one of the available tasks given as a task model, the user often also has to make choices within the workflow of a single task model as a reaction to system response. Again, such choices can lead to different branches in the user interface model. This happens, for example, if the workflow offers different variants or options, which the user can choose or not. Another simple example for this is the error handling done by the user, who is often given the chance to review and correct the data that he/she entered, e.g., before a significant transaction. If the user decides to correct the data, the system will allow the user to modify it, and since the system then usually lets the user review the data again, this probably results in a cycle in the user interface model. A user interface model represents user reactions as different outgoing arrows on client pages.

4. CONSERVATIVE TRANSFORMATIONS FOR USER INTERFACES

A task can be supported appropriately by several differently designed information systems. Therefore, it can be supported by several different user dialogues as well. Given a task model, several form-oriented user interface models may be appropriate. In this section we want to strengthen this understanding by discussing transformations of form-oriented user interface models that preserve the suitability for a given task. By doing so we will reencounter the importance to distinguish between functional and non-functional requirements. When asking the question whether two user interfaces are equally suited to support the user in fulfilling a particular task, we are interested in functional requirements. But a transformation can preserve either just the functional requirements, or also the non-functional ones, i.e., the dialogue itself.

We call user interfaces which support the same task *task-similar*. To consider the multitude of task-similar user interfaces at the level of user interface models has its advantages: the form-oriented analysis level provides a defined conceptual semantics that comprehends facts about the system and therefore enables to grasp these facts easier. Moreover this level is technology independent, in particular, it is a level where issues can be discussed without executable technology at hand. For more information on this refer to [4].

Two user interfaces that support the same task might be different in quality, and these differences might be easily subject to different opinions. Figs. 5 and 6 depict *task-preserving transformations*, i.e., both the original and the transformed user interface models support the same task, which is a functional criterion and therefore not subject to different opinions. We do not make any statement about the quality of any user interface. The figure uses a part from the task model in Fig. 2 where the user has to choose a departure and an arrival airport. It illustrates three different ways for modelling such a dialogue, which can be transformed to one another: the first is the *wizard pattern*, which presents the user with the different subtasks in a fixed sequential order, possibly allowing some limited degree of back and forth navigation. The second is the *superpage pattern*, which unites the user interface parts necessary to accomplish all subtasks in a single page. The third one is created by *outsourcing* subtasks from the superpage onto their own page.

The transformations depicted in Fig. 6 are not only functionally preserving; they also fulfil a stricter criterion: they are *dialogue-preserving transformations*, i.e., both the original and the transformed user interface presents the user with the same dialogue. So the transformation is rather one of the model than one of the user interface as it is visible to the user. The example is about how to deal with a failed login attempt of a user: this exception can either be dealt with on the original login page, which should be *general* enough to allow the optional display of an error message, or on a new *specialized* version of the login page.

Note that the transformations described in this section are

only examples and present by no means complete account on task-preserving transformations.

With respect to usability, specialization and generalization create a paradoxical situation: an interface that is sequential, like a wizard, can be seen as usable for a task that would allow for parallelism. This user interface would provide very little *freedom*. Vice versa, a user interface that allows to execute subdialogues in parallel can be seen as usable for a task that requires sequentiality. The system would provide less *guidance* than the task model requires, but the user could observe the sequentiality for himself. How to reconcile these seemingly contradictory positions? The method of form-oriented analysis offers a solution. A typical user interface pattern of form-oriented analysis offers both, parallel as well as sequential mode: in the default representation, a menu is shown for example at one place of the page, for example the left-hand side. It offers all option in parallel as so-called *menu supported* options. In the rest of the page there is a dialogue that offers *page-guided* interactions. Following only the page-guided links would give a sequential dialogue. Using the menu would support a parallel usage.

5. A DEFINED NOTION OF USABILITY

Usability has many aspects and is therefore hard to define formally. One of the features of the form-oriented model is that one can define a notion of usability that states whether a use case can be performed on a particular user interface. Since a use case always corresponds to a task, we call this *support or appropriateness for a task*. We define usability by examining the relation between form-oriented use case models, which can be created by applying the mapping described in this paper to a task model, and user interface models.

In the simplest case, a use case can be modelled as a linear formchart, i.e., a linear sequence of pages and actions as illustrated in the upper part of Fig. 7. If there exists a homomorphism mapping the pages, actions and transitions of the use case model onto corresponding ones of the user interface model, then the user interface fulfils the requirements that are necessary to perform the use case. If the use case model was created from a task model, then the user interface is said to be *appropriate to the task*.

6. DISCUSSION OF RELATED WORK

An overview of model-based user interface development MB-UIDE is given in [12]. MB-UIDE targets the developer, not the end user. Some of the approaches require the learning of exhaustive visual or textual modelling languages. At the same time an advanced understanding of a certain, non-trivial system metaphor is sometimes necessary to fully exploit the features of the tools in this category. Lightweight model-based approaches like [2] can be used to generate throw-away-prototypes for the communication between the developer and the end-user. However, if the generation of initial application stubs is the goal, a model-based tool must be integrated in an overall development framework in order to take off.

Parts of the community considers classically task models as the primary models that ensure learned design beyond hacking [17, 16]. The mapping problem addressed here is the focus of a large and active research community on model-based user interface design. In [3] the mapping problem was addressed under special consideration of updates to the model. The DYNAMO-Aid approach taken there aims at balancing six different models expressing context-sensitive aspects. Our approach has a simpler model on the task modelling side, but adds a fully understood dialogue model with operational semantics, something that was lacking so far; the interface side remained informal. In [13] a mapping was discussed to Dialogue Graphs, a technique used in the TADEUS approach. Dialogue graphs represent GUI interfaces.

The questions of the mapping problem are in close relation to the usability aspects addressed in the international standard on ergonomic requirements [7]. In general, mapping task models to user interfaces is in the terminology used there a question of *controllability* of the application. The other superficially related topic there, suitability for the task, is not related to task modelling, but rather to ergonomics only.

An important class of tools for the development of IT systems according to use case models are still CASE tools. These tools support typically only a single type of task or use case model, classically following the use-case driven approach [9]. The creation of user interfaces according to user task models is primarily solved as an activity in the overall development process model that comes with the CASE tool [8], but there is no defined mapping between the model itself on the artefact level as it would be desirable. In contrast, this latter level of support is offered by our approach. With current CASE tools, task models and user interfaces can be constantly updated and hence aligned. This process is however not directly supported on a higher level. The only support that is currently given is generator technology, i.e. the generation of user interfaces from task models. This is however a very operational solution, and it leads in the case of redesign to the very technical roundtrip engineering problem. In contrast our approach aims at a descriptive and more flexible explicit modelling of the connection. A technology proposal that could be used in order to transform task models into user interfaces is model driven architecture [5, 10]. The model driven architecture discusses the transformation of a platform independent model (PIM) into a platform specific model (PSM). The task model could take the role of the platform independent model; but in model driven architecture the relationship is again a generative one and hence a purely operational description of the relationship. Another technology that does successfully practise the transformation of a task modelling notion into a running system are workflow systems [6] like IBM lotus workflow. However, in this process no classical user interface design can be performed; rather a very standardized user interface is generated. This approach is only suitable for interfaces that are used by users experienced with administrative work, the user interface has no possibilities for enhancing the usability.

7. CONCLUSION

The current discussion of the mapping problem focuses on mappings between task models and interface descriptions that are explicitly seen as lower level. This mapping is generally seen as starting with the task model and progressing

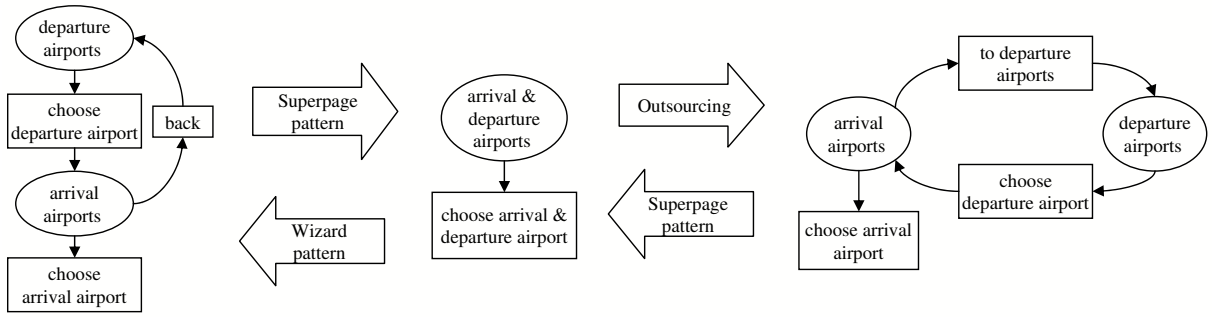

Figure 5: Task-preserving transformations.

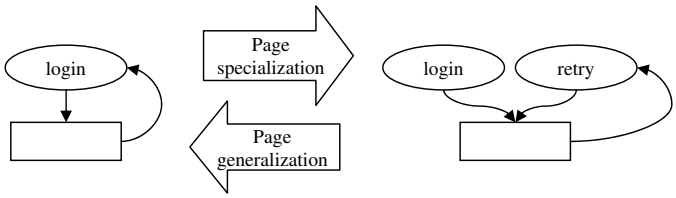

Figure 6: Dialogue preserving transformation.

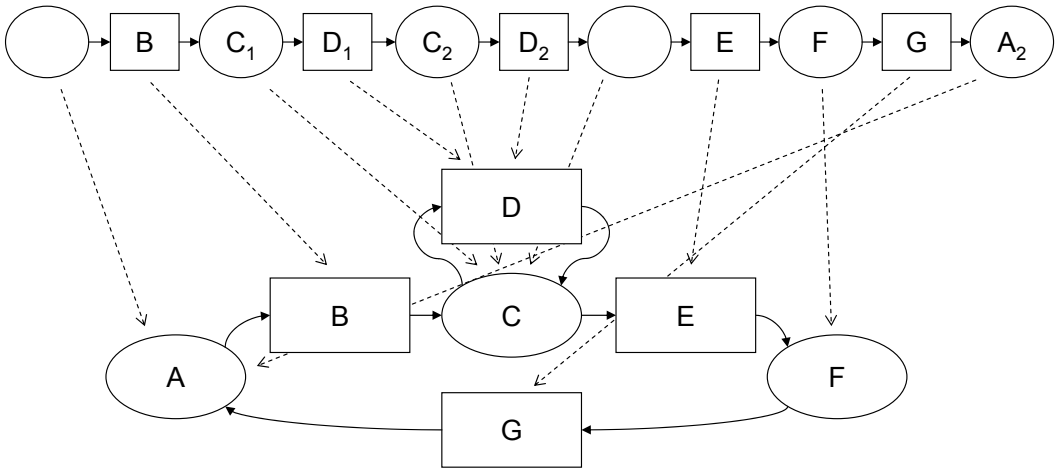

Figure 7: Usability as homomorphism between use case and UI model.

to the the specification of the lower level models [3]. We put forward that for validation we want to extend this approach: for validation we need a predicate on and not a function between task models and user interfaces, a declarative criterion and not an operational procedure. For this purpose we have accompanied the task model by a model that is on the same high abstraction level as task models. The form-oriented user interface we have employed is platform independent and even presentation independent. Finally, we have reflected on our declarative definition of the mapping and have argued that we find here a formal aspect of usability as support for a task.

8. REFERENCES

[1] S. Balbo, N. Ozkan, and C. Paris. Choosing the right task modelling notation: A Taxonomy. In D. Diaper and N. Stanton, editors, *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates, 2004.

[2] H. Balzert. From OOA to GUIs: The JANUS System. *JOOP*, 8(9):43–47, 1996.

[3] T. Clerckx, K. Luyten, and K. Coninx. The mapping problem back and forth: customizing dynamic models while preserving consistency. In *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*, pages 33–42. ACM Press, 2004.

[4] D. Draheim and G. Weber. *Form-Oriented Analysis - A New Methodology to Model Form-Based Applications*. Springer, October 2004.

[5] D. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. OMG Press, January 2003.

[6] D. Hollingsworth. The Workflow Reference Model. Technical Report Document Number TC00-1003, Workflow Management Coalition, January 1995.

[7] International Organization for Standardization. *Ergonomic Requirements for Office Work with Visual Display Terminals (VDT) – Part 10: Dialogue Prinicples*. ISO 9241-10, 1996.

[8] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[9] I. Jacobson, W. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering – a Use Case Driven Approach*. Addison-Wesley, 1992.

[10] J. Miller and J. Mukerji. MDA Guide Version 1.0.1. Technical Report omg/2003-06-01, Object Managment Group, 2003.

[11] F. Paternò and C. Santoro. Preventing user errors by systematic analysis of deviations from the system task model. *Int. J. Hum.-Comput. Stud.*, 56(2):225–245, 2002.

[12] P. Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. *Lecture Notes in Computer Science*, 1946:207–226.

[13] D. Reichart, P. Forbrig, and A. Dittmar. Task models as basis for requirements engineering and software execution. In *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*, pages 51–58. ACM Press, 2004.

[14] D. Scapin and C. Pierret-Golbreich. Toward a Method for Task Description: MAD. *Work with Display Units*, (89), 1990.

[15] A.-W. Scheer. *Aris: Business Process Modeling*. Springer, 2000.

[16] S. Wilson and P. Johnson. Bridging the generation gap: From work tasks to user interface designs. In J. Vanderdonckt, editor, *Proceedings of CADUI'96 – Computer-Aided Design of User Interfaces*, pages 77–94. Presses Universitaires de Namur, 1996.

[17] S. Wilson, P. Johnson, C. Kelly, J. Cunningham, and P. Markopoulos. Beyond hacking: a model based approach to user interface design. In J. L. Alty, D. Diaper, and S. Guest, editors, *People and Computers VIII*, British Computer Society Conference Series, pages 217–231. Cambridge University Press, 1993.