Task Models as Basis for Requirements Engineering and Software Execution

Daniel Reichart
University of Rostock
Institute of Computer Science
Albert Einstein Str. 21,
18059 Rostock, Germany
dr007@informatik.uni-rostock.de

Peter Forbrig, Anke Dittmar
University of Rostock
Institute of Computer Science
Albert Einstein Str. 21,
18059 Rostock, Germany
[pforbrig|ad]@informatik.uni-rostock.de

ABSTRACT

In this paper we discuss an approach linking GUI specifications to abstract dialog models. Both specifications are based on task models describing behavioral features. It will be shown how first prototypes of interactive systems, which are generated from user interface models, can help to capture requirements. Users can interactively play with prototypes. Tool support is also provided for co-operative work of different users, which starts with abstract canonical prototypes that can evolve to concrete GUI specifications.

Author Keywords

Model-Based Design, Task Models, Object Models, Animated User-Interface Prototypes, XIML, XUL.

ACM Classification Keywords

HCI

INTRODUCTION

Model-based development of software systems becomes more and more popular. Even if it is up to now not used very extensively it is an attractive process with proven record of success, especially in the context of developing multiple user interfaces. There are approaches focusing on object models first like the model-driven architecture of UML [21]. However, we follow task-based approaches like ADEPT [24], CTTE [16] or Cameleon [1]. Here, models of existing and envisioned task situations help to derive specifications of interactive systems, and in particular, of user interfaces.

Fig. 1 summarizes our general view on a transformational model-based design process of interactive systems. We strongly believe that software engineers and user interface designers have to base their work on the same models. Furthermore, we consider software development as a sequence of transformations of models that is not performed in a fully automated way but by humans using interactive tools. Our work is especially focused on methods and tools supporting transformations by patterns. The transformation of class diagrams by patterns using Rational Rose is described in [11]. In [18], the idea of supporting the development of task models by patterns is presented.

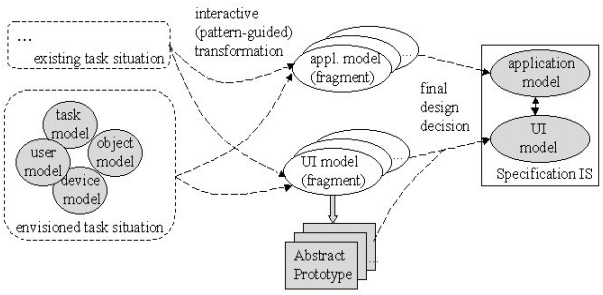

Fig. 1 General view on a transformational (pattern-guided) model-based development of interactive systems

Although model-based design techniques aim to reflect user perspectives they do not support active user participation. Often, the models are used by designers only. A generation of abstract prototypes is proposed to overcome this lack. On the one hand, prototypes support a better understanding between users and developers. For example, they can facilitate the discussion of design alternatives. On the other hand, abstract prototypes are based on abstract models and fundamental changes are only possible by modifying these models. Hence, the dangers of "quick-and-dirty" developments or of persisting on existing prototypes are reduced.

In this paper, the focus is on the user interface development. We present strategies and appropriate tool support (DiaTask) to derive user interface models (in this case, dialog graphs) and corresponding abstract prototypes on the basis of task models (see the bottom part of Fig. 1).

RELATED WORK

Most task-driven design techniques exploit hierarchically organized task structures with temporal dependencies between sub-tasks (like [16]) to derive models of user interfaces. It can be distinguished between approaches which try to automate this derivation process (e.g. [13]) and approaches which make the designers responsible for relating task models and user interface models (e.g. [15]).

TAMODIA'04, Prague, Czeck Republic.

However, the emergence of, for example, multiple and context-sensitive user interfaces required more elaborated relationships between task and user interface models.

In [17], an approach supporting the development of multi-device interfaces is introduced. The corresponding environment TERESA (Transformation Environment for inteReactivE Systems represAntation) [19] mainly integrates the whole functionality of CTTE [16] and provides additional features. Especially, it provides an automatic transformation of task models into abstract user interfaces consisting of presentations. For each presentation the associated logical interactors are identified. The tool offers declarative indications of how such interactors should be composed.

DynaMo-AID (which stands for Dynamic Model-bAsed user Interface Development) [2] is a design approach for developing context-sensitive user interfaces that support dynamic context changes. It is based on an extended version of CTT-task models. Decision nodes are introduced to collect distinct sub-trees from which one will be selected at runtime according to the current context of use. Task models control the execution of an application.

Another method for adaptive user interfaces based on usability properties is proposed in [12]. It is called AL-BASIT, which means Adaptive Model-Based User Interface Method. It extends usual model-based user interface development methods to support the development of adaptive user interfaces in a seamless way. The adaptation rules are performed by an adaptation engine, which is based on agent technology and uses patterns.

Besides more sophisticated model-based design methods languages were developed allowing the description of multiple user interfaces. The development of UIML [20] started in 1997. The goal of this project was the specification of device-independent UI's. UIML is based on XML for specification. To generate a specific user interface a generation-tool called renderer is needed. For each possible end-device-type you need a specific renderer. A flexible reaction to new device-types is difficult. UIML has a commercial background by the company "Harmonia". We were not able to get tool support for really device independent user inter-face specifications.

Another commercial solution ("Redwhale") is the XIML-concept [25]. XIML started in 1999 and is focused on device-independence primarily of mobile devices. XIML is model-based but like the UIML-variant it needs a specific tool (converter) to create a specific type of user interface. Our tool DiaTask [6] was developed using XIML. Task models, user models, and object models with our metaphor of artifacts and tools are represented as XIML specification. However, there is still a lack of tool support for the representation of the user interface. That was the reason to look for a specification, which is already supported by tools. XUL was a candidate for that.

XUL [5,14] was presented in 1999 by the Mozilla project to specify Graphical User Interfaces of the Mozilla-browser in platform-independent matter. XUL allows the specification of interactive objects like buttons, labels, and text fields. We can find these objects in tools for creating GUI's like Java.AWT and Java.Swing. Unfortunately, XUL is restricted to Graphical User Interfaces.

"UsiXML (which stands for USer Interface eXtensible Markup Language) is a XML-compliant markup language that describes the UI for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. In other words, interactive applications with different types of interaction techniques, modalities of use and computing platforms can be described in a way that preserves the design independently from peculiar characteristics of physical computing platform"[22]. It seems to be that UsiXML could be a living standard to express models. It can play the role XIML originally wanted to gain.

FROM DIALOG GRAPHS TO ABSTRACT PROTOTYPES OF USER INTERFACES

It seems to be counter-intuitive but we start the introduction of our approach by explaining the transformation steps from a user interface model to abstract prototypes of user interfaces. Then, in the next section we will consider the transformation of task models to user interface models.

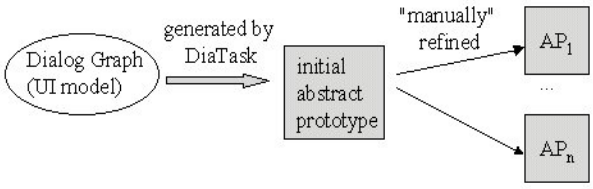

Fig. 2 From User Interface Models to Abstract User Interface Prototypes – our strategy

For modeling user interfaces we use dialog graphs, which will be explained in the following sub-section. Fig. 2 visualizes our strategy to derive abstract prototypes of user interface from a dialog graph by applying the tool DiaTask.

Dialog Graphs

A dialog graph consists of a set of nodes, which are called *views* and a set of transitions [10]. There are 5 types of views: single, multi, modal, complex, and end views. A single view is an abstraction of a single sub-dialog of the user interface that has to be described. A multi view serves to specify a set of similar sub-dialogs. A modal view specifies a sub-dialog, which has to be finished in order to continue other sub-dialogs of the system. Complex views allow a hierarchical description of a user interface model. End views are final points in (sub-)dialogs. Each view is characterized by a set of

(navigational) elements. A transition is a directed relation between an element of a view and a view. Transitions reflect navigational aspects of user interfaces. It is distinguished between sequential and concurrent transitions. A sequential transition from view v1 to view v2 closes the sub-dialog described by v1 and activates the sub-dialog, which corresponds to v2. In contrast, v1 remains open while v2 is activated if v1 and v2 are connected by a concurrent transition. Fig. 3 shows the graphical notation for the different types of views and transitions.

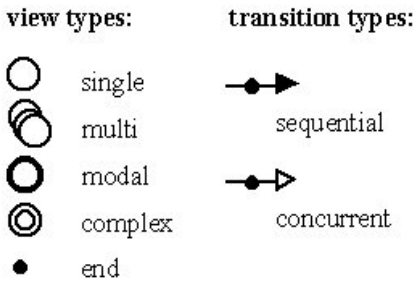

Fig. 3 Notations of view types and transition types of Dialog Graphs

The Tool DiaTask

DiaTask is a tool we developed in order to create dialog graphs. In addition, the tool allows the generation of an initial abstract prototype of a user interface on the basis of a dialog graph and its refinement as will be explained in the following sub-sections. DiaTask is implemented in Java and currently available as plug in for eclipse [9].

Fig. 4 demonstrates an example of a mail system with two single views ("main window", "write mail"), a multi view ("read mail") and the end view. There are concurrent transitions from element "read mail" of view “main window” to view “read mail”, from element "write mail" of view “main window” to view “write mail” and three more sequential transitions. For reasons of clearness, transitions are drawn between views in the DiaTask editor. To get more precise information the dialog designer has to double click on nodes or transitions.

Generation of Abstract Prototypes

Given a dialog graph DiaTask can generate an initial abstract prototype in a WIMP style, which mainly reflects the navigation structure of the user interface. Windows represent views and the elements of the views are mapped to buttons as to be seen in Fig. 5 for the example dialog graph of Fig. 4.

An animation of a prototype reveals how transitions are transformed. Clicking on a button representing the start node of a sequential transition let the active window disappear. The window, which represents the destination view of the transition, becomes visible and active. Concurrent transitions allow for the corresponding window of the source view to be visible. The user can activate it at any time. For the example, this means that "main window" stays open after pressing the "read mail" or the "write mail" button. Multiple views go together with concurrent transitions. A generated prototype allows the creation of several instances of the corresponding window.

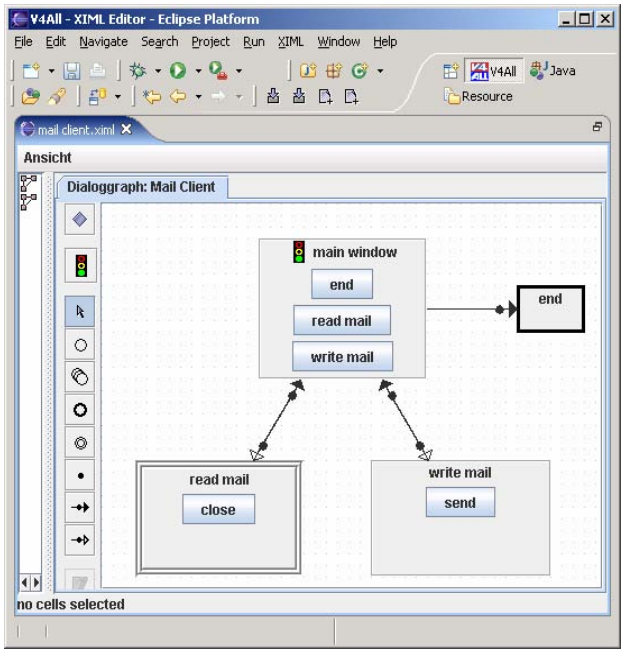

Fig. 4 Dialog graph for a mail system

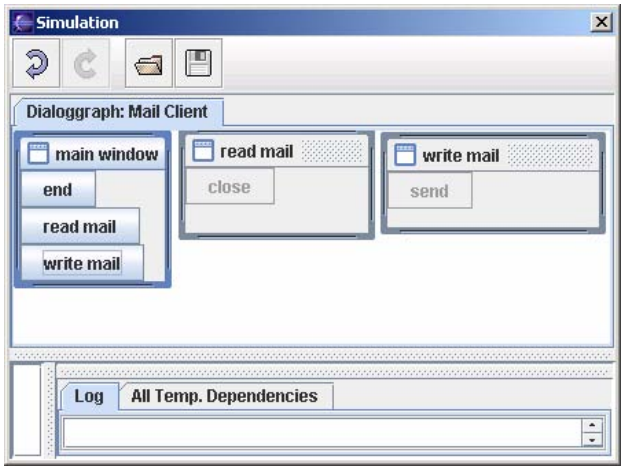

Fig. 5 The initial abstract prototype of Fig. 4 in animation mode

For our example mail system this means that a user can read several mails in parallel. In Fig. 5, a situation is captured where the user is reading one mail and writing another one. The “main window” is active. Pressing the "read mail" button causes the opening of a new window instance, pressing "write mail" button has no effect, and pressing "end" button closes the dialog.

The initial abstract prototype and its interactive animation can be considered as abstract canonical prototype in the sense of [3]. Users get a first impression of the interactive system under development. Some design decisions are already made and can be discussed. However, there is

still a big gap between this prototypical user interface and the envisioned version. To close this gap DiaTask facilitates an editing of generated interfaces. Hence, the initial abstract prototype can be refined differently as illustrated in Fig. 2.

Refinement of Abstract Prototypes

Based on the open source project v4All [23] a GUI editor was developed. It stores user interface specifications as XUL code [5,14], displays corresponding graphical user interfaces, and allows changes to it. The GUI editor also is implemented as eclipse plug in. Thus, it can easily cooperate with the DiaTask tool. Fig. 6 gives an impression of the editor.

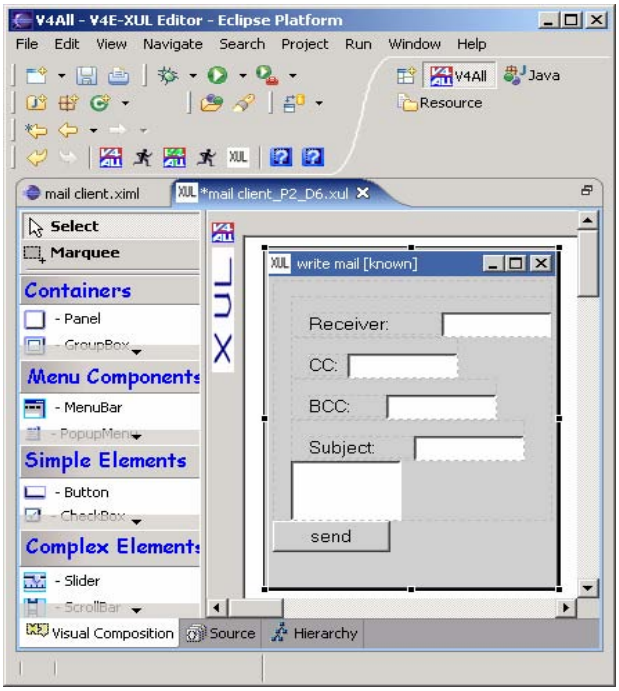

Fig. 6 GUI editor for XUL specifications

DiaTask generates the initial abstract user interface prototype of a dialog graph by producing specific XUL-files for each view of the graph and associated transitions. The GUI editor was designed in such a way that it allows to replace existing user interface elements of such a specification by new ones without destroying references to the underlying dialog graph and task model (see next section). To be able to animate prototypes an interpreter for rendering these XUL specifications was written. Fig. 7 visualizes a refinement of the initial prototype illustrated in Fig. 5. The generated window for "write mail" is replaced by the interface specification rendered in Fig. 6. Additionally, a fancier interface description for reading a mail is provided. The animation of the dialog graph becomes "more readable" for users and is more appropriate for discussions. In Fig. 7 the same animation state as in Fig. 5 is shown.

Generally, model-based approaches support an incremental software development. That is to say that those parts of models or specifications, which are not affected by changes in underlying models should not be destroyed. For this reason, the GUI editor is able to cope with already designed user interface specifications and additionally generated elements. If an XUL specification of a view of the underlying dialog graph exists it is used and interpreted. If no such file exists a new one is generated.

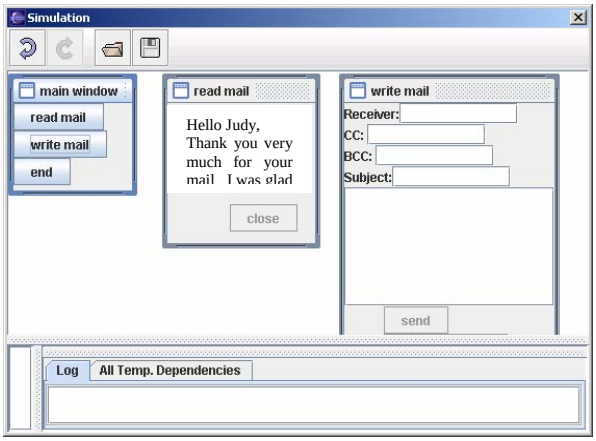

Fig. 7 A refined abstract prototype of Fig. 5 in animation mode

FROM TASK MODELS TO DIALOG GRAPHS AND CORRESPONDING PROTOTYPES

This section deals with how to exploit task models to come to task-oriented user interface models. In Fig. 8, it is to be seen that we prefer interactive transformation steps instead of deriving user interface models automatically. Consequently, designers are responsible for relating tasks and user interface elements (see the discussion in the section about related work).

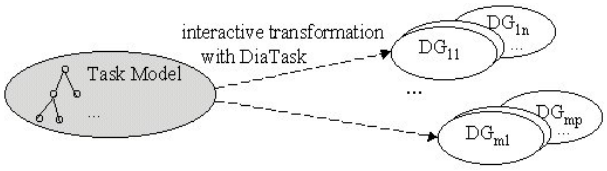

Fig. 8 From a task model to user interface models

DiaTask supplies an own editor for task models. It follows the idea of attaching artifacts, tools, roles and devices to tasks. However, models from CTTE [4] can also be imported.

Mapping from a Task Model to Dialog Graphs

On the one hand, a designer can assign alternative interface models (dialog graphs) to one task model. This is useful e.g. for designing multi-device or nomadic user interfaces as we have shown in [8]. A dialog graph specifying the interface for a PC might differ from a dialog graph for a PDA although they have the same underlying task model. As another example, a dialog graph of a mobile phone interface should rather contain sequential than concurrent transitions.

On the other hand, one can relate a set of dialog graphs to one task model as also indicated in Fig. 8. This makes sense when the task model reflects co-operative work like in [16]. Each graph could describe a user interface supporting the tasks of a specific role. In the following

example, which demonstrates the main ideas of our approach we model a simple co-operative system.

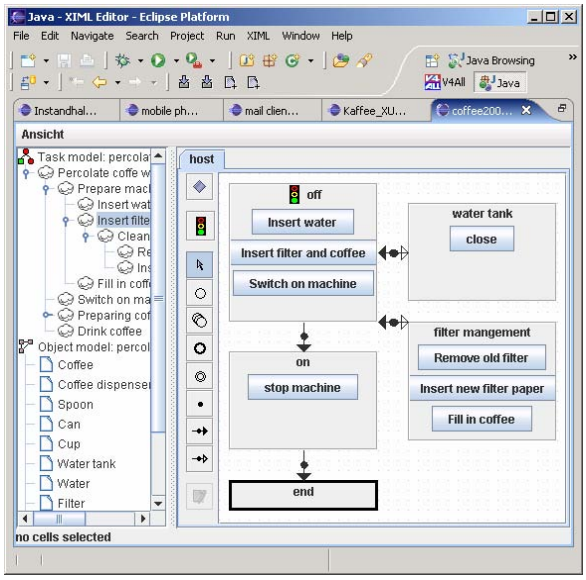

Fig. 9 Task model "Percolate coffee with machine" and dialog graph of role "host"

On the left hand side of the screen shot in

Fig. 9 one can see the underlying task model "Percolate coffee with machine". We do not want to go into too much detail. The task is divided into sub-tasks "Prepare machine", "Switch on machine", "Prepare coffee" and "Drink coffee". Sub-task of "Prepare machine" are "Insert water", "Insert filter and coffee", and "Fill in coffee" etc. Temporal relations between sub-tasks are not visible in

Fig. 9 but will be later, e.g. in Fig. 13.

Three dialogue graphs are assigned to the task specifying user interfaces for roles "host", "guest", and "system". “Host” has to prepare everything, “system” plays the role of the machine and has only to percolate coffee, and “guest” has to drink the coffee. Fig. 9 shows the dialog graph of "host". It is to be seen that sub-tasks of the task models are mapped to elements of views (by using equal names). The designer has to decide which sub-tasks are grouped together in a view. In the example, the grouping is rather oriented at objects of the task domain (e.g. "Water tank", "Filter") as the names of the nodes reveal. It is also possible to orient at sub-tasks of the upper level of the task hierarchy. While the designer has to create views independently from the task model, DiaTask supports the mapping of sub-tasks to elements of views. By selecting a sub-task (e.g. "Insert water") and a transition type (e.g. concurrent) a transition can be specified by drawing a line between two nodes (e.g. between "off" and "water tank"). If only a sub-task is selected and attached to a node by clicking on it, the corresponding element is not related to a transition. This is useful for a mapping of basic sub-tasks to elements, which have to play the role of a function call in a user interface. In order to decide about the type of a transition a designer can take a look at the task model but there is no automatic support for that. It is his design decision, which type of transition between views is used.

However, besides the transitions of a dialog graph the temporal relations of the underlying task model control the behavior of the generated user interface prototype in the animation mode as to be seen in next sub-section. It can be checked whether deadlocks occur by the specification of temporal relations, views and transition types.

In the example, the start view is “off” (in DiaTask denoted by a traffic light). The transition to “water tank” is assigned to element “Insert water”. Element “Insert filter and coffee” has got a transition to dialog view container “filter management”, where the old filter can be removed, the new one can be inserted and the coffee can be filled in. The transitions back to view “off” are related to “close” and “Fill in coffee” respectively. Additionally, the host can switch on the coffee machine and stop it.

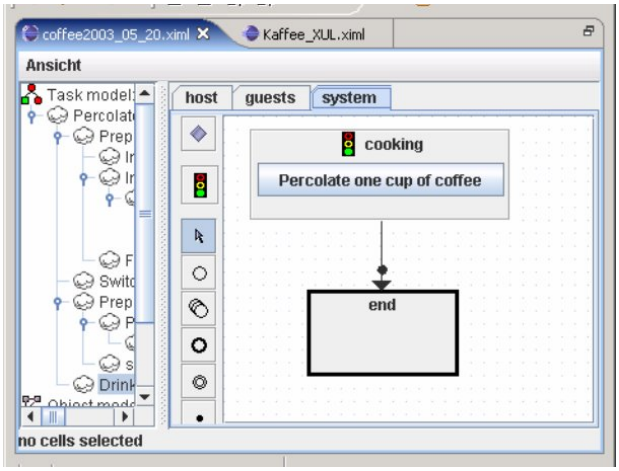

Fig. 10 Dialog graph of role “system”

Most of the tasks have to be performed by the host. The system is only responsible for percolating a cup of coffee and the guest can drink it. Hence, the corresponding dialog graphs of the role guest and the role system are much simpler (see Fig. 10 and Fig. 11).

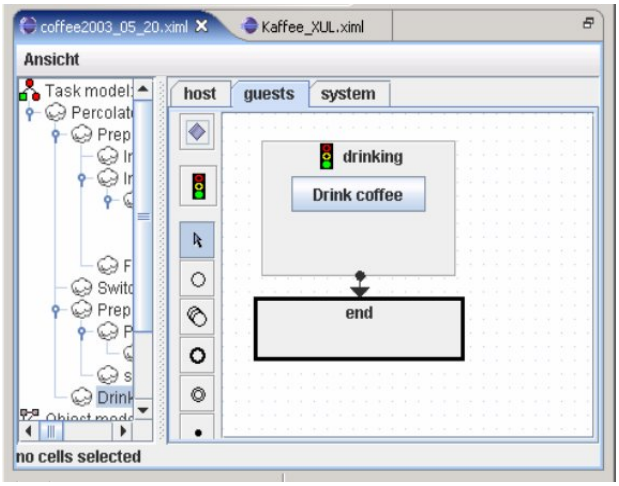

Fig. 11 Dialog graph of role “guest”

As for the generation of the initial abstract prototype, the user interfaces of “host”, “system”, and “guest” have to be coordinated.

Task-based Animation of User Interface Prototypes

Fig. 12 illustrates that, in addition to the dialog graph(s) from which an abstract prototype is generated the underlying task model controls the animation of this prototype. The temporal description of the task model is exploited to enable/disable buttons of a corresponding prototype. Inconsistencies between the navigation structure of dialog models and the temporal relations between sub-tasks are reported. To be more precise, a task model animator is integrated in DiaTask, which runs in parallel to the abstract prototype. The actual enabled task set determines which buttons are enabled in the user interface of the prototype.

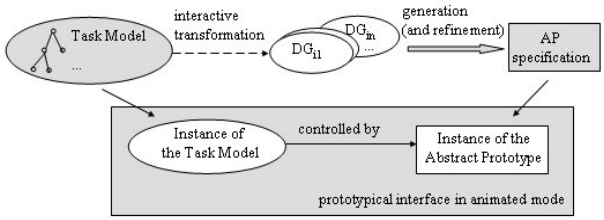

Fig. 12 Model-based control of the animation of an abstract user interface prototype as supported in DiaTask

It is not our intention to introduce a complex model for co-operative work and appropriate interactive systems within this paper. However, our approach allows a simple presentation of role-specific user interface prototypes of co-operating people.

If several dialog graphs for different roles are derived from a task model their corresponding abstract prototypes ‘observe’ an instance of this task model in the animated mode. That is to say, the task model serves to co-ordinate the behavior of the single (prototypical) user interfaces running in parallel.

In this way it plays the role of a controller and the cooperative work can already be discussed on the basis of the prototype. The consequences resulting from the task model can be demonstrated to the users in a very good understandable way.

Fig. 13 provides a screenshot of the animated prototype of role “host”. It visualizes the situation after filling water in the tank, removing the old filter and inserting the new one. At the bottom of the window the animated task model is presented. Circles visualize complex sub-tasks at higher levels of the task hierarchy. Squares visualize basic sub-tasks (the leaf nodes in the hierarchy). If a basic sub-task is enabled in the next animation step this is denoted by a green circle. Sub-tasks which been executed are ticked, disabled sub-tasks are marked by a cross. In the situation depicted in Fig. 13 the only enabled sub-task is “Fill in coffee”. Hence, only button “Fill in coffee” is enabled in the abstract user interface prototype.

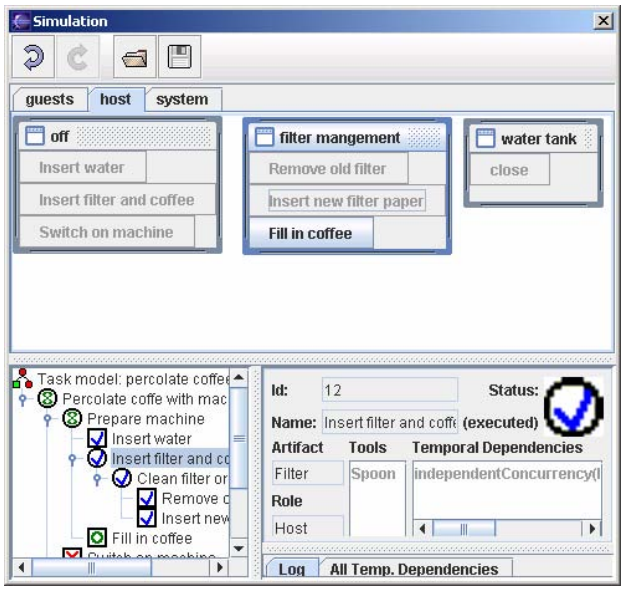

Fig. 13 Abstract prototype of role “host” and corresponding task model in animated mode

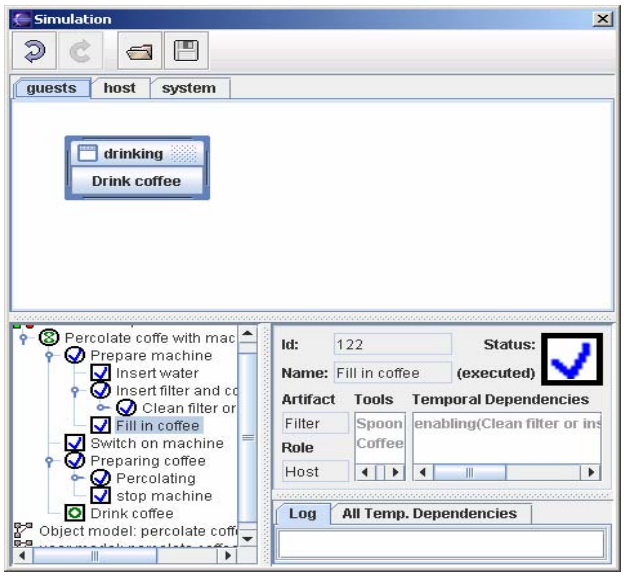

Fig. 14 Finally, the guest can drink a cup of coffee

Fig. 14 shows the abstract user interface of “guest” in the final state. The host has filled water into the machine, has removed the old filter, inserted a new one and has filled it with coffee powder. Then, the system became active by percolating a cup of coffee and the host has stopped the coffee machine.

The guest can drink a cup of coffee eventually (as to be seen in the bottom part of the window of Fig. 14 all sub-tasks of role “host” and “system” are accomplished).

How to integrate Task Domain Knowledge?

As mentioned earlier, abstract prototypes support discussions with users and other stakeholders during early design stages [7]. However, we also recommend a refinement of abstract user interface prototypes generated by DiaTask (see e.g. Fig. 7). In [6] we have shown how formalized task domain knowledge can be used to enrich

task models. Such models facilitate a mapping from domain knowledge to data input and data output elements of a corresponding user interface model. These more expressive models allow the generation of more realistic abstract prototypes and reduce refinement steps of the initial abstract prototype, which have to be performed 'manually' by a designer at present. So far, such more elaborated task domain models are not integrated in DiaTask but we work on that problem. In the future first versions of object representations like mails in our introducing example will be generated based on class models. A refinement can be performed with our existing GUI editor.

CONCLUSION AND FURTHER WORK

Tools for model-based software development have reached a development stage that allows applying them to real projects. We presented strategies and tool support for designing interactive systems, and in particular, user interfaces mainly on the basis of task models.

Our approach can be considered as complementary to approaches like TERESA [19] and DynaMo-AID [2]. These approaches prefer an automatic generation of user interface prototypes from task models. We strongly believe in transformation-based development process controlled by the designers. For that reason, we follow the strategy of leaving explicit design decisions to interface designers.

We provide tool support for the animation of abstract canonical prototypes and carefully designed user interfaces. Starting from an abstract user interface a prototype can evolve to his final appearance by designing step by step each view of the navigation dialog.

With our tools it is also possible to animate different navigation graphs concurrently. In this way co-operative work of different users can be demonstrated. It is also possible to look at different design decisions and let users compare the user interfaces. Usability tests can help to make the final decisions.

Especially a user-centered requirements analysis is supported. Consequences of the captured functional requirements can be visualized. Users can work with animated prototypes, which step by step evolve from abstract model to the final user interface. We strongly believe in transformation-based development process controlled by patterns. Such patterns can support the interactive design decision. We did not focus on this aspect in this paper. Some results were already published [18] and there is ongoing work on this topic. It should for instance be possible to provide a kind of floor plans for the user interface layout. Such a floor plan is a further abstraction. It does not consider elements themselves but their topological relations. Different kinds of user interfaces can follow the floor plan with different tapes of concrete elements. We are working on this problem for our GUI editor and believe that floor plans can be considered as patterns as well. Transformations can be based on such information.

The models for our tools are specified in XIML [25], user interface prototypes are stored as XUL code. It seems to be that this idea is no longer supported. There is no information on the web site and there are no further tools publicly available. UsiXML[22] follows very similar concepts. From our point of view it can be the standard for the future. UsiXML supports all necessary models, especially presentation models. It might be useful to reengineer our tools in such a way that they support this language.

Our experiences with task models show that they have the potential to control an application. Same results are reported in [12]. The implementation of a model interpreter running as web service is under way. Based on this tool we will control a nomadic application for machine maintenance, the domain of one of our industrial partners.

REFERENCES

1. Cameleon: http://giove.cnuce.cnr.it/cameleon.html.
2. Clerxkx, T.; Luyten K.; Conix, K.: DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Develoment, *Proc. EHCI-DSVIS'04*, p. 142-160, 2004.
3. Constantine L.L: Canonical Abstract Prototypes for Abstract Visual and Interaction Design, in Jorge J. A. et. al (Eds): *Proceedings DSV-IS 2003*, LNCS 2844, Springer Verlag, Berlin, 2003, P. 1-15.
4. CTTE: The ConcurTaskTree Environment. http://giove.cnuce.cnr.it/ctte.html.
5. Deakin, N.: XUL Tutorial. XUL Planet. 2000.
6. Dittmar, A., Forbrig, P.: The Influence of Improved Task Models on Dialogues. *Proc. of CADUI 2004*, Madeira, 2004.
7. Dittmar, A., Forbrig, P., Heftberger, S., Stary, C.: Tool Support for Task Modelling – A Constructive Exploration. *Proc. EHCI-DSVIS'04*, 2004.
8. Dittmar, A., Forbrig, P., Reichart, D.: Model-based Development of Nomadic Applications. In *Proc. of 4th International Workshop on Mobile Computing*, Rostock, Germany, 2003.
9. Eclipse: http://www.eclipse.org.
10. Elwert, T., Schlungbaum, E.: Dialogue Graphs – A Formal and Visual Specification Technique for Dialogue Modelling. In Siddiqi, J.I., Roast, C.R. (ed.) *Formal Aspects of the Human Computer Interface*, Springer Verlag, 1996.
11. Forbrig, P.; Lämmel, R.; Mannhaupt, D.: Patterns-oriented develpment with Rational Rose, *Rational Edge*, Vol. 1, No. 1, 2001.
12. López-Jaquero, V.; Montero, F. ; Molina, J.,P.; González, P.: A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties, *Proc. EHCI-DSVIS'04*, p. 372-389, 2004.

13. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J.: Derivation of a dialog model from a task model by activity chain extraction. In Jorge, J., Nunes, N.J., e Cunha, J.F. (ed.), *Proc. of DSV-IS 2003*, LNCS 2844, Springer, 2003.
14. Mozilla.org: XUL Programmer's Reference 2001.
15. Navarre, D., Palanque, P., Paterno, F., Santoro, C., Bastide, R.: A Tool Suite for Integrating Task and System Models through Scenarios. In Johnson, C. (ed.), *Proc. DSV-IS 2001*, LNCS 2220, Springer Verlag, Berlin, 2001.
16. Paterno, F.; Mancini, C.; Meniconi, S: ConcurTaskTrees: A Diagrammatic Notatiob for Specifying Task Models, *Proc. Interact 97*, Sydney, Chapman & Hall, p362-369, 1997.
17. Paterno, F., Santoro, C.: One Model, Many Interfaces. In *Proc. of the Fourth International Conference on Computer-Aided Design of User Interfaces*, p. 143-154. Kluwer Academics Publishers, 2002.
18. Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P., Seffah, A.: Patterns in Model-Based Engineering, *Proc. of CADUI 2004*, Madeira, 2004.
19. TERESA: http://giove.cnuce.cnr.it/teresa.html
20. UIML Tutorial, http://www.harmonia.com
21. UML: http://www.omg.org/uml
22. UsiXML: http://www.usixml.org/
23. v4All: http://v4all.sourceforge.net/index_start.html
24. Wilson, S.; Johnson, P.: Bridging the generation gap: From work tasks to user interface design, In Vanderdonckt, J. (Ed.), *Proc. of CADUI 96*, Presses Universitaires de Namur, 199, p. 77-94.
25. XIML: http://www.ximl.org