Deriving User Interfaces from Task Models

Andreas Wolff, Peter Forbrig
University of Rostock
Albert Einstein Str. 21
D-18069 Rostock
{Andreas.Wolff, Peter Forbrig}@uni-rostock.de

ABSTRACT

This position paper presents our view on model driven user interface development and relates it to other approaches. Our methodology is based on task models that are attributed and merged with a navigational model to derive user interface models. A toolset to support this development approach is introduced which is well-integrated and itself based on the eclipse modeling framework EMF.

1. INTRODUCTION

Model-based software development is becoming more and more popular because of its advantages in managing different versions of software for different platforms. The term model-based is today often only identified as a part of the object-oriented development strategies related to UML. However, the origins of using model specifications during software development were task-based ideas. Originally such task models were used to specify the behavior of people.

For several years our group has been working on combining object-oriented and task-based methods. Our research is especially focused on using patterns for this purpose.

Model-to-Model transformations based on Eclipse Foundation's EMF [5], an implementation of W3C MOF meta-model, have been used in many fields of research and business. This paper illustrates another appliance. Two EMF models are used as transformation source. One is describing the application as a task model and another EMF model describes the possible and allowed navigation through an envisioned UI of this application. Typical WIMP-style user-interfaces may, reasonably well, be described in this manner. The tools developed for our approach use openArchitectureWare (oAW) [9] workflow templates to combine task model and dialog graph and transform those models into a working mock-up of a user interface prototype or into a abstract user interface.

Abstract user interfaces can still be represented as EMF models. For concrete user interfaces another type of model is used. UIs can be described, defined or programmed in many different ways. Closely related to model-based development are any types of XML-based UI-languages. Their main advantage is a well-defined grammar and their hierarchical structure.

While several XML-based UI languages are wide-spread; XAML (Microsoft .NET), UsiXML[11] and XUL [13](Mozilla) certainly are of major importance. Transformations of this paper produce XUL-models as target model.

2. METHODOLOGY

Our software development methodology is based on task models of psychologists and manpower studies. According to these studies, a task has to be performed on an artifact by a person in a certain role using tools. Additionally, some tasks have to be executed on devices with certain features only. In an abstract way a task can be considered as specification of the following form: Task=(Goal, Sub-tasks, Temporal Relations, Pre-conditions, Post-conditions, Artifact, Tool(s), Role(s), Device). The notation of concurrent task trees (CTT) has nearly become a standard in representing tree structures of tasks. We also make use of an extended CTT-specification for designing applications as task models.

Our development strategy is focused on WIMP-style user interfaces and possibly limited to those. Support for multi-modal or 3-dimensional systems is not actively researched. A core aspect of window-based user interfaces to interactive systems is their navigational structure.

We specify this navigation structure by a special graph, which is called dialog graph. Dialog graphs consist of nodes, mostly interpreted as views, and of transitions between those nodes. There are different node types that distinguish views in complexity, hierarchy and modality. Transitions are directed relations from an element of a view to another view or element. Transitions reflect navigational aspects of user interfaces. Typical elements of a view are tasks.

The concept of dialog graphs may reflect different abstraction levels. It is not limited to platform independent models (PIM) or abstract user interfaces either. Therefore we use dialog graphs as one transition point from PIM to platform specific models (PSM). Allowing the assignment of differing dialog graphs to one task model is one possibility of adaptation to context-of-use within our development methodology.

The combination of task model and navigational structure is the basis to generate an abstract user interface model (AUI). AUIs describe the internal layout of each view, i.e. the relative positioning of tasks within it. Furthermore the class of user interaction elements for every task is assigned, e.g. '1:n' input (choose from list) or hierarchical selection (tree) or string output (label). Note that explanations in parentheses are only illustrative, but not definitive assignments.

An abstract user interface can be transformed to a concrete user interface (CUI). For this model-to-model transformation the abstract definitions of layout and user interaction elements are mapped to a precise layout. User interaction object classes are replaced by definitive implementing widgets. This model is eventually used by code generators to deliver an application or a final user interface respectively.

As mentioned before, we strive to automate the whole workflow from task models to final user interfaces. While we found that fully automatic generation does not produce feasible user interfaces, i.e. acceptable from a usability point of view, a

combination of automatisms and human designer assisted transformations yields sensible results.

Table 1: Available adaptations during development process

Model	Adaptation to context-of-use
Task model	sequence, iteration, concurrency
Dialog graph	(semantic) grouping of tasks into views, transitions and navigational flow
Abstract UI	basic screen layout, widget class per task
Concrete UI	implementing widgets graphical design (coloring, images, labels) positioning of elements

Table 1 summarizes the models used in our approach and indicates which kind of adaption is possible on each level.

3. TOOL SUPPORT

To support our user interface development process a number of tools was developed. Most of them started as separate tools which communicated using proprietary XML-based languages. While the number and variety of tools evolved this became a stalling factor in development. Therefore we decided to port them into the EMF environment and use XMI for communication.

For that purpose meta-models for task model and dialog graph were developed, other like those of abstract and concrete user interface were reverse engineered from their respective XML schema definitions. For model-to-model transformations we now use e.g. openArchitectureWare workflows.

3.1 Defining task model and dialog graph

Task models in CTT notation form the basis of our modeling approach. Besides standard CTT our graphical editor is capable of handling advanced concepts as for example prioritization or instance iteration. Also artifacts needed and presentation defaults may be defined with this tool.

Next step in our methodology is the navigational structure. The previously existing dialog graph editor was converted into an EMF/GMF editor. It allows manual creation of views and assignment of tasks to views. Its features include hierarchical dialog graphs, i.e. views can be composites of sub-view[s] and tasks; this was developed as adaptation concept to reduce complexity and repetition in dialog graphs.

Automatic generation of dialog graphs from task models is not a trivial task. There are basic approaches as: "put every task in its own view" or "put all tasks in a single view", which of course are not satisfying. Enabling task-sets, as known from CTTE [3], might be a promising approach, but for now we attempted a semi-automatic procedure.

We require that the modeler annotates tasks of his task model with markers. Those marks enable an interpreter to decide which tasks belong into the same view, in our case the interpreter is the dialog graph editor. Modelers can either manually attach the additional information within the task editor or they can specify general profiles for operators that are used during the transformation process. Any task may be marked as:

- *ICV* – Integrate Children into View: A task and its first level children are attached into the same view
- *IN* – Ignore Node: Task and all its children are ignored for navigational model
- *PUNM* – Pick Up for Navigation Model: Abstract tasks are included into the navigation model
- *VL* – View List: Explicit grouping and ordering of children

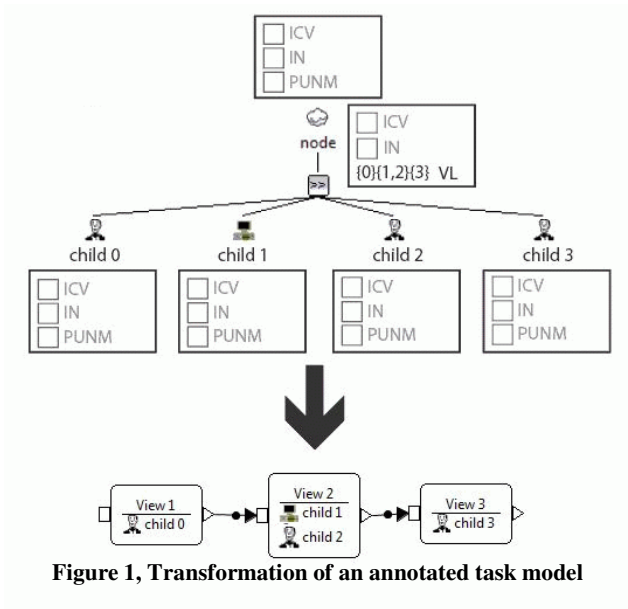

Figure 1, Transformation of an annotated task model

In **Figure 1** partial screenshots from two tools can be seen. The upper part is from our task editor and shows a very simple CTT-alike task model. In this notation the enabling node has the same meaning that separate enabling transitions between children nodes would have. Also the effective grouping annotations for each task are displayed. Here all grouping is defined in the "enabling" node, via its marker **VL**.

During the model-to-model transformation the abstract task is ignored and its children are grouped according to the explicit ordering specification *"{0}{1,2}{3}"*. Activating e.g. **ICV** instead of **VL** would result in one view only with all tasks within it.

The lower part is an excerpt from our dialog graph editor. This is how it displays the resulting model of three simple views with sequential transitions between them.

Marker profiles are preset configurations that may be applied to all or only selected nodes of task models; as such they may be platform independent. Manually attached information is considered to be platform specific as well. The concept and the corresponding tool support are discussed in depth in Diebow [4].

3.2 Generating a user interface

Figure 2 shows the meta-model of dialog graphs. Stereotypes Model and ModelElement refer to their respective EMF meta-model types, as well as EBoolean is EMF's Boolean type. Classes Task, DeviceType and UserRole are defined in other packages; those are non-displayed as their actual declaration is of no importance for this paper.

Any view of a dialog graph is modeled as DialogView object. A view object has references to each task it contains and if needed to artifacts of those tasks. Navigation between views is controlled and specified using the port-metaphor. A port is either source or target of a transition.

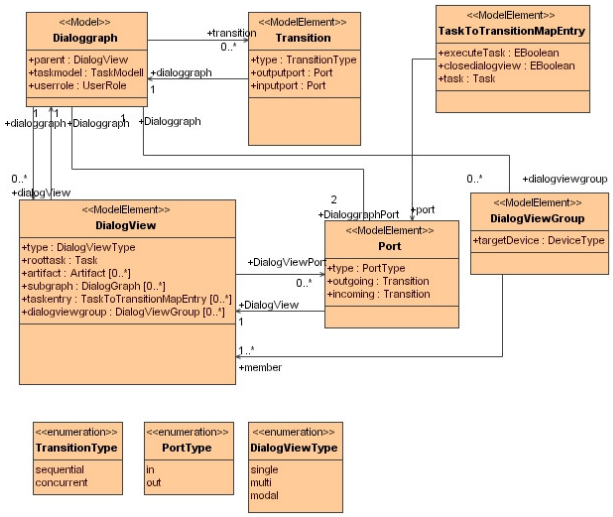

Figure 2, Meta model of dialog graph models

The completion of a certain task may result in or trigger a transition. Transitions are specified within objects of class TaskToTransitionMapEntry. Conditional transitions referring to ports of a view are stored in its DialogView.taskentry attribute.

The meta-model provides means to define hierarchical dialog specifications. Class DialogViewGroup is used entirely for hierarchical dialog graphs. A parent-child relationship is established between instances of Dialoggraph and DialogView, navigatable as parent or rather subgraph.

As discussed in section 2 it is possible to generate an abstract interface from a dialog graph. We want the AUI to serve as mock-up of an application. Thus it can be used to discuss the application workflow with any stakeholder in a very early development phase.

User interface prototypes, for WIMP-interfaces, can e.g. be defined in a number of XML languages. We decided on a combination of HTML and XUL for that purpose. A suitable viewer would be every Gecko-based web-browser; Firefox is probably the most well-known example. Actual UI generation is done by openArchitecureWare workflows who transform the EMF model of a dialog graph into XUL.

For the generation at first the application starting or entry point has to be identified. Our basic WIMP-style application consists of views and simple interaction objects arranged into those views. Each application, per definition, has a start view configured in its dialog-graph.

Dialog graph view-to-view transitions are defined using a port-to-port metaphor. Application start- and end-nodes fit into this system as well. There is a port defined as application-start, type outgoing and named ‘start’, and there is an opposite port, type incoming and named ‘in’, denoting the end of the application. Objects of type Dialoggraph contain references to those entry and exit ports within their attribute DialoggraphPort.

For generating our application’s startup mask the transformation begins in the context of the root dialog graph. Here the application’s entry port is identified by OCL and an openArchitectureWare transformation workflow is invoked.

3.3 CUI and patterns

After developing and testing the dialog structure on the level of simple prototypes and abstract user interfaces the next step would be the concrete user interface. We mainly use XUL for this model level, for the same reason as for section’s 3.2 prototypes. Also we have generators for UsiXML and are working on XAML.

But still most advanced support is available for XUL as CUI model. There is an EMF editor and the GEF XUL editing tool (XUL-E). It is a typical editor for graphical user interfaces and features drag and drop layout and WYSIWYG-layout as far as it is possible for XUL.

Beside its graphical editing features its main purpose is to integrate into our development approach and to provide an engine for user interface pattern application.

XUL-E may be used to refine or re-layout the prototypes generated in section 3.2 while leaving their navigational structure intact. Thus it is possible to keep the application’s prototype character while improving its graphical representation. This is achieved by tracking UI elements that are modified or replaced. This modification information is used to attach the correct model references to new user interface elements.

Another source for concrete user interfaces are AUI descriptions from the USG process [14], which we map in semi-automatic transformations to XUL.

Another of our main research topics is the integration of HCI patterns into MDD. The object-oriented patterns introduced by Gamma et.al. [16] are widely known as valuable aid and we consider HCI patterns to be of the comparable value. Therefore we try to integrate those patterns into our model-based UI development process.

We focus our pattern integration on the CUI level, but consider task and navigational patterns as well. We try to translate the idea of a certain pattern into an algorithm or a layout template and define this as a pattern instance component (PIC) [15]. A PIC is a machine readable definition and may be employed semi- or fully automatically to a user interface model.

PICs are stored as EMF models making our main tool chain completely based on eclipse’s [graphical] modeling and as such well integrated.

If a modeler is satisfied with its designed CUI, whether by pure editing or using PICs, he may use the source code generator and produce Java Swing source code of it.

4. RELATED WORK

A lot of tools and methodologies for model-based development of interactive systems have been developed. In this section we concentrate on approaches that we consider as most typical and promising.

These are TERESA [10] developed within the Cameleon [1] project and UsiXML [11] which is a language, a set of models and a tool set. Both approaches do not provide explicit navigation specifications like dialog graphs. This information is, if available at all, distributed over different other models.

In general TERESA generates the user interface based on its models by presenting all tasks active at the same time in one window, but it is possible to use additional heuristics.

TERESA and UsiXML do not really consider the abstraction level of a dialog graph. They have the level of abstract user interface, which is a little bit more concrete than our approach. Our dialog graphs are more concrete than task models but more abstract than their AUIs. Unlike TERESA and UsiXML our dialog graph gives the opportunity for an explicit design process of the navigation structure. Our approach may be offered as an alternative strategy to software developers, if the generation process does not deliver the expected results. It might depend on the application domain and especially on the number of tasks in the model, which approach fits best. Especially for larger task models our approach might fit better because not all tasks are presented that could be executed in a certain moment.

The approach proposed by Costa et. al. [2] uses a kind of task trees represented in UML notation for dialog modelling. Each task element is detailed by the Dialog Model using a UML-compliant adaptation of the CTT notation. Tool support is available by DialogSketch [8]. Canonical abstract prototypes are used to specify the first level of user interfaces. In this way it does not have this level of abstraction we want to support too. It is focused on Web interfaces and related papers do not discuss the problem of different roles and a variety of platforms.

Most publications related to navigation specifications seem to be related to the area of web applications. This was the result of a current survey of the literature. Leung et. al. [7] specify the navigation for web applications by state charts. They present specification solutions for intra-page, inter-page and frame-based navigation by hierarchical states. They already raise the problem of multiple windows that are specified in dialog graphs as multiple views. Their solution suggests separate state charts for each window. The authors identified dynamic content as a special problem of their kind of specification. From our point of view the readability is another problem. The notation is much more complex than our dialog graphs. It also does not allow different kinds of transitions. Roles and devices were not considered.

Koch [6] as well focuses her work on web modelling. She bases her specification on UML. UWE (UML-based Web Engineering) is a model-driven development approach. Class diagrams with special profiles specify the navigation model. More or less only the menu structure is represented by the specification. It is the goal of this approach to transform navigation models together with business specifications in form of activity diagrams and state charts to service-oriented applications.

5. SUMMARY

In this paper we briefly described a set of tools for model-driven user interface development. Those tools form a tool chain which is embedded into the eclipse modelling framework and uses standardized XML dialects for specification and user interface models.

The presented methodology is based on task models and uses specialized navigational models to derive abstract and concrete user interfaces and to eventually generate source code. It is adaptive on any model level and able to integrate HCI pattern into the development process.

Future work will especially focus CUI integration of UsiXML, the model representation of pattern instance components and pattern for task model to dialog graph transformations based on our newly introduced marker concept.

6. REFERENCES

[1] Cameleon: http://giove.cnuce.cnr.it/cameleon.html (visited November 29, 2008)

[2] Costa, D., Nóbrega, L., Nunes, N.: An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes, TAMODIA 2006, Hasselt, Belgium, October 23-24, 2006.

[3] CTTE: The ConcurTaskTree Environment. http://giove.cnuce.cnr.it/ctte.html (visited November 2008)

[4] Diebow, Ch.: Entwicklung eines Konzeptes zur interaktiven Trtansformation von Aufgabenmodellen in Navigations-modelle, Master Thesis, University of Rostock, 2008.

[5] EMF: http://www.eclipse.org/modeling/emf/?project=emf (visited November 4, 2008)

[6] Koch, N.: Transformation Techniques in the Model-Driven Development Process of UWE, ICWE'06 Workshop, Palo Alto, 2006

[7] Leung, K.R.P.H., Hui, L.C.K., Yiu, S.M., Tang, R.W.M.: Modeling Web Navigation by Statechart, Proc. COMPSAC'00

[8] Nóbrega, L., Nunes, N. J. and Coelho, H.: DialogSketch: Dynamics of the Canonical Prototypes, TAMODIA'2005, Gdansk, Poland, September 26-27, 2005.

[9] openArchitectureWare:http://www.openarchitectureware.org/ (visited November 4, 2008)

[10] TERESA: http://giove.cnuce.cnr.it/teresa.html (visited November 4, 2008)

[11] UsiXML: http://www.usixml.org/ (visited November 4, 2008)

[12] Tool Support for an Evolutionary Design Process using User-Interface Patterns

[13] XUL: http://www.xul.org (visited November 4, 2008)

[14] Müller, Andreas: Spezifikation geräteunabhängiger Benutzerschnittstellen durch Markup-Konzepte, PhD dissertation, University of Rostock, 2003

[15] Wolff, Forbrig,Reichart: Tool Support for Model-Based Generation of Advanced User Interfaces. *MDDAUI '05, Proc. of the MoDELS'05 Workshop on Model Driven Development of Advanced User Interfaces*, CEUR Workshop Proc. 159. CEUR-WS.org, 2008.

[16] Gamma, Helm, Johnson, Vlissides: Patterns-Elements-Reusable-Object-Oriented-Software, Addison-Wesley, 1994, ISBN 0-201-63361-2