Automatic User Interface Generation from Declarative Models

Egbert Schlungbaum and Thomas Elwert
Universität Rostock, Fachbereich Informatik, Albert-Einstein-Straße 21, D-18051 Rostock, Germany
Phone: +49-381-498-34{19, 27} – Fax: +49-381-498-3426
E-mail: {Egbert.Schlungbaum, telwert}@informatik.uni-rostock.de
WWW: http://www.icg.informatik.uni-rostock.de/~{schlung, telwert}

Abstract

Automatic user interface generation is a widely discussed topic in the research community. In recent years several approaches have been developed to support this kind of generation. There is a need to summarise this. This article should provide a basis for a founded discussion in this direction. The article gives an overview about model-based user interface software tools. The special attention is paid to the declarative models. The process of user interface generation is highlighted on a basis of a categorisation. The main section contains ideas of TADEUS about automatic user interface generation explained by an example.

Keywords

Model-based user interface software tools, user interface generation.

Introduction

User interface software is often large, complex and difficult to implement, to debug, and to modify. An average of 48% of the code of application is devoted to the user interface, and that about 50% of the implementation time is devoted to implementing the user interface portion [Myers92]. As user interfaces become easier to use, they become harder to create [Myers94].

A lot of user interface software tools was created in order to help the user interface developer. In our days the state of the art tools are called higher level tools [Myers95]. Higher level tools exist in a large variety of forms, for example UIMSs, UIDEs, IBs, UIDEs[1], Application Frameworks and further. They are built on the top of user interface toolkits.

Brad Myers overviews the current state of the art in user interface software tools [Myers95]. He introduced a classification of these tools. It is based on the way how

[1] Do not confuse this general term with Foley's UIDE - The User Interface Development Environment [Foley94] the state of the art tool in the area of model-based user interface software tools.

the designer can specify the layout and the dynamic behaviour of a user interface. There are tools which require the user interface developer to program in a special-purpose language (*language-based tools* in Myers' classification), which allow to design the user interface interactively (*interactive graphical specification tools* in Myers' classification), or which automatically generate the user interface from a high-level model or specification (*model-based generation tools* in Myers' classification).

Language-based tools as well as interactive graphical specification tools are commercially available and frequently used at present. But the development of user interfaces is still a difficult and time-consuming activity by using one of these tools. Language-based tools support the specification of the control of the user interface in an easy way. But the problem is that the developer must specify layout, placement, and format for each user interface object.

There is an opposite situation with interactive graphical specification tools. On the one hand the designer creates the layout of the user interface interactively what seems to be a natural way to develop a user interface and can be carried out by non-programmers. On the other hand the dialogue control specification has to be added by using a programming language or by using a special purpose language.

Furthermore, the language-based tools as well as the interactive graphical specification tools do not support the developer to follow existing user interface guidelines and style guides in order to maintain the internal consistency across the user interface as well as the external consistency with other applications.

A further important lack of language-based tools and interactive graphical specification tools is that the results of requirements analysis cannot be directly used for user interface development in most of existing user interface software tools. Solving this problem is a issue of extensive current research (e.g., [Coutaz94, EHCI95]).

Olsen et al. [Olsen93] suggest the automatic user interface generation as an essential part of future user interface development environments. The model-based generation tools were introduced to solve the mentioned problems. Several model-based user interface software tools have been built. Some of these are UIDE [Foley94], HUMANOID [Szekely93], ADEPT [Johnson95, Wilson96], ITS [Wiecha90], MECANO [Puerta94b, Puerta96b], TRIDENT [Bodart95a], BOSS [Schreiber94b], GENIUS [Janssen93], JANUS [Balzert95a], MASTERMIND [Szekely95], AME [Märtin-96a, Märtin96b].

As shown in figure 1 the common property of all these tools is that the desired user interface is automatically created from a specification represented by declarative models.

The model-based approach offers many potential benefits over traditional methods of building user interfaces (see also [Szekely95]), e.g., powerful design and runtime tools, support for early conceptual design, consistency and reusability, iterative development, integrated development of user interface and application core. But this

approach is still at the research level (see also [Myers95]), because the user interfaces that are generated are not good enough.

Furthermore, the specification languages are quite hard to learn and use. Extensive current research is done to address these problems. On the other hand, there are tools which primarily focus on design assistance during the user interface development process. Examples are EXPOSE [Gorny95], IDA [Reiterer94].

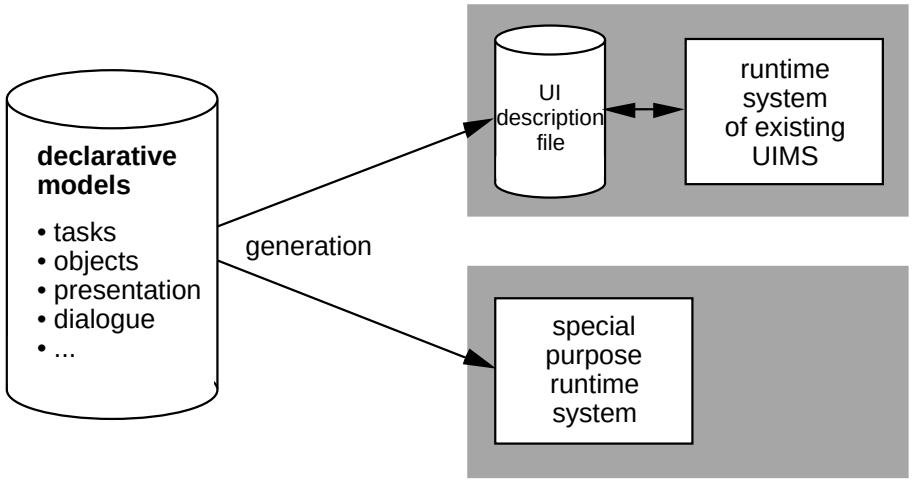

Figure 1. Model-based user interface generation

The purpose of this article was to encourage the discussion during the special track CADUI workshop. There is a long tradition in CADUI and in our opinion it is necessary to summarise the research results. For it, the paper is organised as follows. Different model-based user interface software tools are shortly surveyed in the next section.

The points of investigation are the use of different declarative models and the computer-based user interface generation from it. After it, the automatic user interface generation in the TADEUS approach is described in detail.

1 Model-Based User Interface Software Tools

1.1 Representing Information by Declarative Models

As mentioned above there are several model-based user interface software tools. All these approaches follow one key idea. That is, the information which is required for user interface development is explicitly represented in declarative models. These models are accessible by the user interface, the application core and external tools at design time and at run time.

Summarising shortly the mentioned model-based tools there are some kinds of models which are used commonly [Puerta94a].

A **Task model** is used to describe the tasks the end-user has to perform. Goals in a task model specify when a desired state is met, methods describe procedures to achieve a goal, where atomic methods achieve a goal in one step and composite methods decompose a goal into subgoals.

An **Application model** is to specify the services an application provides. It is mostly object-oriented; objects capture the state of entities and the operations change the state of objects. It is important that the operations correspond to the atomic methods specified in the task model.

A **Dialogue model** is used to describe the human-computer conversation. It describes when the end-user can invoke commands, select or specify inputs and when the computer can query the end-user and presents information.

A **Presentation model** specifies the object and operation appearance, the hierarchical decomposition of displays into components, the attributes and layout of each component.

A **Behaviour model** is used to specify the input behaviour. The use of a presentation model and a behaviour model allows to specify the layout and the dynamic behaviour of the user interface independently.

A **Platform model** can be used to describe platform characteristics, e.g., input devices, output devices.

A **User model** specifies the end-user characteristics. A user model can be used in order to generate individual user interfaces (adapted to stereotypes), to reconfigure the interface to the end-user, to provide adaptive user interfaces, to provide an appropriate level of help, to actively guide the user during interaction.

A **Workplace model** can describe workplace characteristics, e.g., cultural characteristics, environment factors. These models are used in different ways. The first five of these eight are used mainly; the use of an explicit user model was suggested in the context of ADEPT only [Kelly92], neither an explicit platform model nor an explicit workplace model is used in any of the model-based approaches. Furthermore, there are differences in controlling the designed user interface, e.g., controlling by a special-purpose runtime-system that uses the specified models directly or generating a textual user interface description that is used to control an existing UIMS. Let's look into some of the mentioned tools.

In **UIDE** [Sukaviriya93] the designer has to specify an *application model* that consists of *application actions*, *interface actions*, and *interaction techniques*. Parameters, pre-conditions, and post-conditions are assigned to each action. The pre- and post-conditions are used to control the user interface during run time by means of the UIDE runtime system.

An extension to UIDE [Sukaviriya94] provides an *application model* and an *interface model*. The application model consists of tasks which will be performed by end-users, their operational constraints, and objects on which these tasks operate. Inter-

face components, application-independent interface tasks, and operational constraints on these tasks are specified in the interface model. The application semantic information which is stored in the application model is preserved from design time to run time. So it can be used for some sophisticated tools to support the end-user, e.g. automatic generation of context-sensitive help.

HUMANOID [Szekely92, Szekely93] provides a declarative modelling language that consists of five semi-independent parts: the *application semantics* represents the objects and operations of an application; the *presentation* defines the visual appearance of the interface; the *behaviour* defines the input gestures that can be applied to presented objects, and their effects on the state of the application and the interface; the *dialogue sequencing* defines the ordering constraints for executing commands and supplying inputs to commands; the *action side-effects* defines actions executed automatically when commands or command inputs change state (e.g., making a newly created object the current state). The presentation and the behaviour models are specified by using templates, the dialogue sequencing is specified implicitly and is derived from the application model. The designed user interface is controlled by the HUMANOID runtime system.

In **TRIDENT** [Bodart94b, Bodart95a] the designer has to specify a *task model* which is represented by an Activity Chaining Graph (ACG) and an *application model* in form of an entity-relationship diagram. The task model includes the interactive tasks the end-user has to perform, and the sequencing information for tasks in order to achieve the related goal. A *presentation model* represented by presentation units is defined over the ACG. Finally, a textual description of the user interface is generated.

In **GENIUS** [Janssen93] the designer uses the *existing data model* to design the user interface. On the data model he has to define *views* those are used for explicit *dialogue modelling* by means of Dialogue nets and for the layout generation. A textual description of the user interface is generated.

In **JANUS** [Balzert95a] the user interface is generated from an *object-oriented application model* (OOA model that results from object-oriented analysis) by using few knowledge bases. There are not any further models in JANUS. A textual description of the user interface is generated.

According to the notation of the central model (mostly the application or task model) the mentioned model-based approaches can be classified into two classes: the ones which use their own notation (e.g., UIDE, HUMANOID, TRIDENT) and the others which use notations well-known from software-engineering (e.g., JANUS, GENIUS). Especially JANUS is a good example how to use a software-engineering model to generate the user interface. In this way user interface engineering can be integrated into the general software engineering process what is mentioned as a future direction of research by a lot of authors (e.g., [Coutaz94, Curtis94]).

According to the generation target there also can be distinguished two groups (see figure 1): the systems which use their own runtime system (e.g., UIDE, HUMAN-

OID) and the systems which generate a textual description of the desired user interface to animate and to control by means of existing UIMS (e.g., GENIUS, JANUS).

Furthermore, there are some differences in modelling the dialogue sequencing. On the one hand, such systems like UIDE, HUMANOID, MECANO, or TRIDENT do not use an explicit dialogue model. The necessary sequencing information is specified by using pre's and post's (e.g., UIDE), or it is derived from the application model (e.g., HUMANOID, MECANO - extended application model called domain model, JANUS) or task model (e.g., TRIDENT).

On the other hand, some authors argue the importance of explicit dialogue modelling [Janssen96, Lauridsen95, Weisbecker95]. This approach allows to involve the end-user in a participatory user interface design process because of the whole dialogue structure can be shown and discussed at a glance. Furthermore, the generation of the user interface from an explicit dialogue model can lead to a higher quality of the user interface than the generation from other models.

1.2 Process of User Interface Generation

The idea of automatic user interface generation from some kind of declarative description (e.g., application model) is not new at all. The first of these tools were presented about ten years ago, e.g., COUSIN [Hayes85], MIKE [Olsen86]. Currently, there are a lot of various approaches of automatic user interface generation. They are different in the use of input information mostly represented by declarative models (from which the generation is done), the generation target, and the generation process itself. The first two points are shortly reported above. Now we will discuss the generation process.

Although there are differences, some common features of the user interface generation can be identified. Mostly, four basic steps are reported (e.g., [Puerta94b, Balzert95a, Janssen96, Vanderdonckt95b, Weisbecker95]):

- high-level dialogue generation,
- layout generation,
- low-level dialogue generation,
- layout and design revision.

There are also some extensions. Helmut Balzert [Balzert95b] describes not only the user interface generation but also extends to application generation too. Jean Vanderdonckt [Vanderdonckt95b] especially analyses the knowledge-based support of each generation step, e.g., suggestion of interaction style, selection of interaction objects, defining the layout of interaction objects, identification of windows, providing a guideline document.

High-level dialogue generation consists of identification of all windows of the desired user interface, specification of the navigation structure among these windows in the interface, and assigning of interface objects to each window. The term Abstract Interaction Object (AIO) is often used instead of the term interface ob-

jects, e.g., [Morin90, Johnson92a, Vanderdonckt93, Weisbecker95]. AIO takes into consideration behavioural aspects only, they are free of presentational aspects.

TRIDENT uses Presentation Units (PU) defined over the ACG. One or more windows can be identified inside a PU. For it, five identification strategies are suggested and supported by algorithms [Bodart95b]. The selection process of AIO inside a PU can be full automatic or computer-aided. For it, additional information from the task and application model is used [Vanderdonckt93].

GENIUS [Janssen93] automatically assigns a window to each view defined in the dialogue model. The views are defined by hand on the data model. The transitions of the Dialogue nets (Dialogue nets represent the dialogue model in GENIUS) are used for the generation of navigation structure among windows. AIOs are assigned to each attribute of entities related to a view.

JANUS [Balzert95a] assigns a window to each non-abstract class defined in the object-oriented model. The navigation structure among these windows is generated by using the relations between the classes defined in the OOA model and by using one of the knowledge bases in order to generate one pull-down menu item for each window.

MECANO [Puerta94b] is similar to JANUS. It also assigns a window to each class defined in the domain model. The navigation structure is derived from the relations between the classes. In both systems the AIOs are derived from model information, in JANUS an AIO is assigned to each attribute of a class, and in MECANO to each slot of a class.

During **layout generation** each abstract interaction object is assigned to a Concrete Interaction Object (CIO, e.g., dialogue widgets on toolkit-level) and all CIOs are placed on their corresponding windows by a layout algorithm that observes interface design guidelines. Various placement strategies are discussed in the literature (e.g., a summarising overview [Vanderdonckt94d]).

Low-level dialogue generation deals with the user interface behaviour on the CIO-level, e.g., disabling of application actions if there is not any selected object. On this level the systems, that preserve the application semantics from design time to run time (e.g., UIDE, HUMANOID), are good because of dependencies like that mentioned above are specified by pre's and post's and can be used to execute the user interface. In GENIUS the dialogue model was extended with constraints in order to describe low-level user interface behaviour [Janssen96]. This step is not described for all the other tools explicitly.

Layout and design revision is done in the most cases manually. It is an essential step because of automation during layout generation do not guarantee a satisfactory user interface in all cases. This step is used for participatory design steps on which the end user of the desired user interface is involved.

Considering the mentioned methodologies for user interface generation together with the described generation levels we are developing TADEUS – a task-based methodology supporting the user interface development process.

1.3 User Interface Development by Using TADEUS

At the begin we want to describe the TADEUS methodology in general. Then we outline the TADEUS dialogue model shortly.

The TADEUS approach divides the user interface development process of an interactive software system into three stages [Elwert95]. In the first stage, the requirements analysis, the designer specifies three domain models (task, problem domain, and user model) which contain the requirements for the desired user interface. In the second, the dialogue design stage, the designer develops the dialogue model. Its initial form is generated from the already created domain models.

The dialogue model describes the static layout and the dynamic behaviour of the user interface. The third stage is the automatic generation of the prototype of the final user interface by using a software ergonomics knowledge base and additional information input provided by an auxiliary dialogue with the dialogue designer in order to request non-specified information. The generation result is a dialogue description file for an existing UIMS.

The TADEUS dialogue model distinguishes between two different types of dialogue, the navigation and the processing dialogue. The *navigation dialogue* describes the possible interactions between dialogue views which represent logical and functional groups of dialogue objects[1]. The dialogue objects represent objects and methods stored in the TADEUS problem domain model.

A group called *dialogue view* can exist in one or more instances. The dialogue views are transformed later on into windows of the final user interface. The navigation dialogue can be specified by means of Dialogue graphs.

The *processing dialogue* deals with the description of the dialogue within a dialogue view and is expressed by interaction tables. This interaction table stores the design decision about the representation of objects and methods coming from the problem domain model in terms of dialogue object, method, dialogue form, transition, abstract interaction object and concrete interaction object. The interaction table covers the development process from an abstract to a concrete level. In a further

[1] Dialogue objects are close related to task placed in the task model and their related objects and methods and objects and methods placed in the problem domain model. That means in particular a dialogue object represents a problem domain object or a releaser of a method of an object. In the following section we want to use the term interaction object instead of dialogue objects in order to emphasis the interactive nature of these objects. There is no difference between this both terms but in our opinion the term dialogue object fits the desired meaning at the best. We use both in this paper in order to make it easier to find relations to other existing methodologies.

development step of TADEUS we want to use the Dialogue graph notation for the description of the processing dialogue too.

2 Generation of User Interface Software in TADEUS

The development of the dialogue model and the generation of the user interface prototype are closely related. The exactness of the dialogue model influences the effort for the generation of the user interface and its quality. If there are missed information in the dialogue model the dialogue designer has to answer some questions during the generation process to add the missed information.

In TADEUS the desired user interface is primarily generated from the dialogue model which consists of two parts the Dialogue graph and the interaction tables. Additionally, information represented in the task and problem domain models is used during the generation process. The presentation layout of the user interface is generated using the interaction tables and the problem domain model. The dynamic behaviour of the user interface is generated using the Dialogue graph and the task model.

The generation process realised in the TADEUS system conforms to the four steps discussed in the paragraph 1.2. Furthermore, it is similar to the generation steps described in TRIDENT [Bodart95a], GENIUS [Weisbecker95]. The TADEUS generation process contains seven steps:

1. Defining and evaluating the default layout description.
2. Selection of abstract interaction objects for each dialogue form.
3. Mapping from abstract interaction objects to concrete interaction objects.
4. Defining the layout of concrete interaction objects by using the defaults.
5. Placing the concrete interaction objects inside the views automatically.
6. Creation the dynamic behaviour from the Dialogue graph.
7. Generation of the user interface description file for an existing UIMS.

In general, the dialogue designer performs only once the first step for each user interface project. The default layout description includes some presentation properties of CIO. For example, one important point of the defaults is the definition of background and foreground colour relations of CIOs themselves, among different CIOs, and between windows and CIOs which are placed inside the windows.

During the generation process these default settings support to achieve consistency and to speed up the generation procedure. The table 1 gives a short impression of the defaults.

CIO	background	foreground	font	cursor type
window	white	black	mask font	arrow cursor
group box	grey	black	mask font	arrow cursor
edit text	white	blue	text font	text cursor
push button	dark grey	black	button font	action cursor

Table 1. Default layout description (some examples)

The real and possible repeated generation process begins at step 2. The highest level of the TADEUS dialogue model describes views which are transformed into windows during the generation process. That means, the window identification procedure is done by explicit dialogue modelling before the automatic generation starts. Furthermore, the generation steps from 2 up to 5 must be repeated for each view (window). The dynamic behaviour among the windows is generated from the transitions of the Dialogue graph (see below).

An interaction table is defined for each view of a Dialogue graph in order to describe the processing dialogue. There are some examples of interaction tables in the following section. The dialogue designer can define a dialogue form for each transition of the Dialogue graph which is assigned to the current view. If there is no additional information from the task or problem domain model, the default for the dialogue form is a function call, but the dialogue designer can change this value. The use of this additional information is a topic of our current research. The information about the dialogue forms is used to choose AIOs by rules which are derived from table 2. In the following step the abstract interaction objects are mapped to CIOs by rules which are derived from table 3.

dialogue form	type	AIO
function call		action trigger
data input	free	input field; input group
	1:n	single selector
	m:n	multiple selector
data output		output field; output group

Table 2. From the dialogue form to the AIO (some examples)

AIO	type	CIO
input field	free	edit text
single selector	1 : n, (n = const., n ≤ 7)	group box + radio button
	1 : n, (n = const., n > 7)	list box
	1 : n, (n = variable)	list box
multiple selector	m : n, (n = const., n ≤ 7)	group box + check boxes
	m : n, (n = const., n > 7)	list box
	m : n, (n = variable)	list box

Table 3. From the AIO to the CIO (some examples)

In the next steps each concrete interaction object is extended by layout parameters and placed in the corresponding window. The step 4 is solved by the usage of the

default layout description. This description contains information about layout parameters of each concrete interaction object type (e.g., foreground colour, background colour, see table 1). The step 5 is supported by grouping information which is described in the interaction table. This information is not required, but it helps a lot to improve the quality of the generated layout.

When the static layout of all views (windows) is generated the dynamic behaviour among these windows is implemented. All transitions of the Dialogue graph are transformed into executable rules by generation pattern. A generation pattern is defined for each transition, the following figure gives an impression on the essence of a sequential transition (left hand side) and a concurrent transition (right hand side).

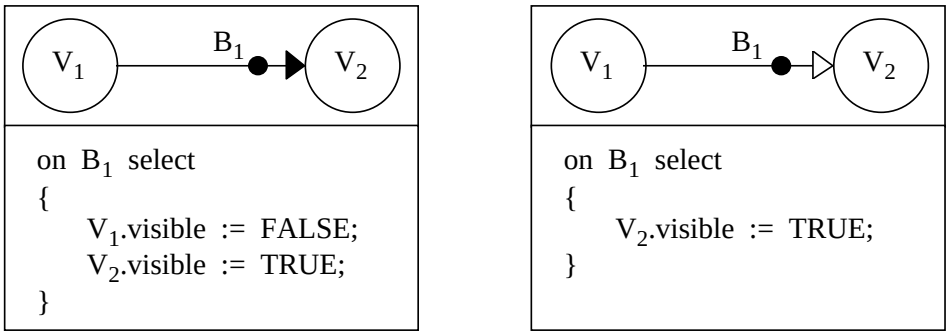

Figure 2. Generation pattern

Up to this point the result of the TADEUS generation process is an internal user interface description which is independent of a concrete UIMS. In the last step it is transformed into a user interface description file of an existing UIMS. So, it is possible to create the same user interface for different UIMS or on different platforms.

3 Example

Let's use a concrete example to demonstrate the TADEUS generation process. The example explains a part of the user interface of the TADEUS environment, the user modelling component [Elwert95]. The necessary parts of the task model and the problem domain model for the user modelling component are shown in figure 3.

Furthermore, figure 3 shows the views the dialogue designer identified over the task model. With it, the Dialogue graph shown in figure 4 can be generated (view 1 = *TADEUS*, view 2 = *user model*, view 3 = *role*).

This example explains two elementary dialogue structures of a GUI of an information system like a database application. The first one describes the situation: the end-user uses the interactive system to support a lot of sub-tasks (e.g., process tasks, process roles) which he/she can carry out concurrently. It is represented with a concurrent transition in the Dialogue graph between the main view and the view of a sub-task.

The second one describes the situation: the end-user wants to process a set of objects of the same type (e.g., different end-users of an interactive application are modelled by different roles). This situation is represented with an object-related concurrent transition.

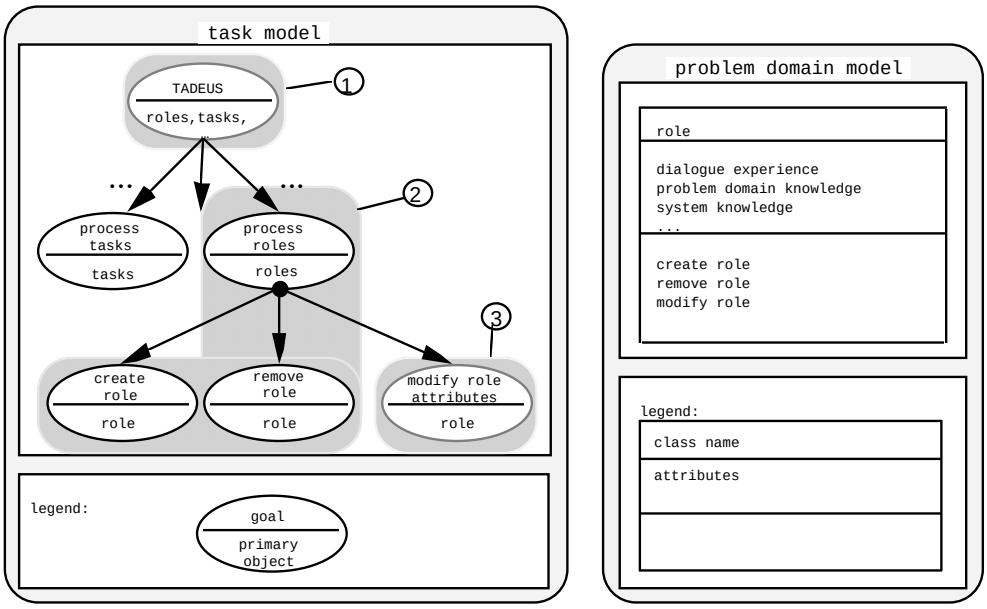

Figure 3. Example: task and problem domain model

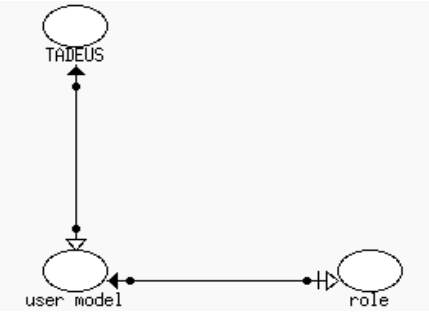

Figure 4. Example: Dialogue graph

The dialogue designer defines for the view *user model* the interaction table (see table 4). There are some special features which should be explained. First, the rows of the interaction table were created automatically. The sequence of the first three transitions confirms to the task model (e.g., the order of sub-tasks from left to right). The *help* and *quit* transition are added by using styleguide information.

Second, the dialogue designer changed the dialogue form of the *create role* transition to data input. And third, the dialogue designer had to change the positions in the second group to achieve suitable sequence of the buttons. The generation result is

shown in figure 5; figure 6 shows the corresponding generation result of the view *role*. The next example explains how the generation results will change, if the dialogue designer uses the generated interaction table (see table 5). Now there are only two groups, the transitions derived from the task model and the transitions added by using styleguide information. The generation result is shown in figure 7.

transition	dialogue form	type	group	position in group
create role	data input	free	1	1
remove role	function call		2	2
modify role	function call		2	1
help	function call		3	1
quit	function call		3	2

Table 4. Example: interaction table of the view ***user model***

transition	dialogue form	type	group	position in group
create role	function call		1	1
remove role	function call		1	3
modify role	function call		1	2
help	function call		2	1
quit	function call		2	2

Table 5. Example: modified interaction table of the view ***user model***

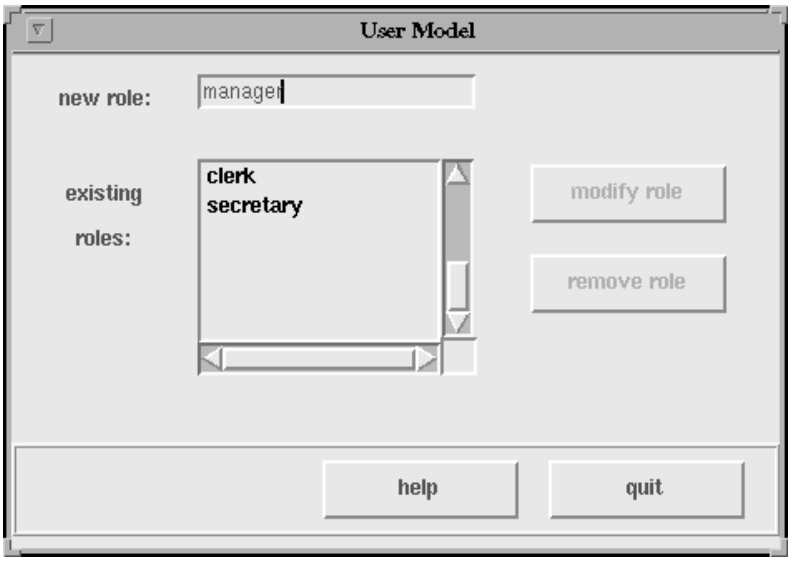

Figure 5. Generation result of view ***user model***

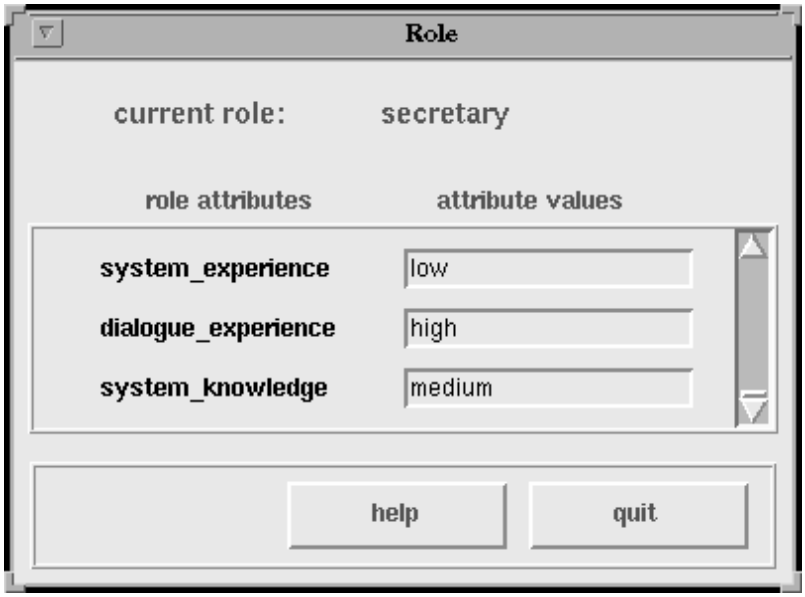

Figure 6. Generation result of the view ***role***

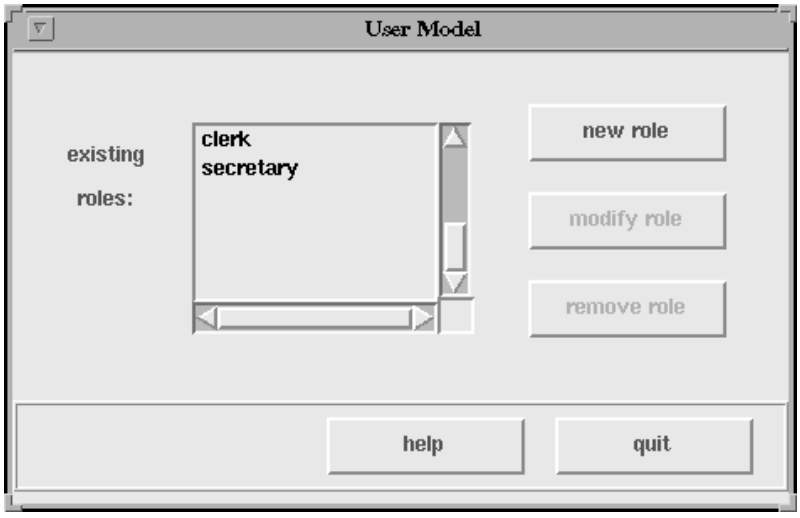

Figure 7. Generation result of the modified view ***user model***

Conclusion

In this paper we summarised the work in the area of model-based user interface software tools in order to come to a basis for automatic user interface generation. In a lot of various model-based user interface tools some common declarative

models are used to specify the necessary information for automatic user interface generation.

The user interface generation process in the TADEUS system was described and demonstrated by an example. The development of the tool supporting this generation process is not finished yet. In order to improve the quality of the generation result, we plan to implement a tool which generates different suggestions of the layout of a view and then the dialogue designer can select the best one. Furthermore, it is necessary to extend the generation tool by a possibility for the creation of the system pull-down menu of the desired user interface in order to fulfil styleguide requirements.

In our opinion, one point of discussion during the CADUI workshop should be the relation between modelling effort and quality of generation result. As it is obvious, on the one hand, the modelling effort using the TADEUS environment is high, but on the other hand, the generation result of user interfaces in the area of information systems is acceptable.

Furthermore, there are a lot of other points which could be discussed. Important ones are which steps of user interface generation can be done in a full automatic way, how many it will cost (e.g., realisation of the tool, required time of the generation procedure), and what kind of quality we will get as result of this generation process. Or there are any steps which the dialogue designer must execute (these steps are unable for automatisation) or should execute (these steps are carried out by the dialogue designer better than automatisation) in order to achieve an acceptable quality per acceptable costs.

Acknowledgements

Many thanks to Peter Forbrig for his remarks and proof reading and the anonymous referees for their helpful comments.

References

[Balzert95a] Balzert, H., *From OOA to GUI - The JANUS-System*, in Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction INTERACT'95, Lillehammer, 25-29 June 1995, K. Nordbyn, P.H. Helmersen, D.J. Gilmore and S.A. Arnesen (Eds.), Chapman & Hall, London, 1995, pp. 319-324. http://www.swt.ruhr-uni-bochum.de/forschung/janus/lillehammer. html

[Balzert95b] Balzert, H., Hofmann, F., Niemann, C., *Vom Programmieren zum Generieren - Auf dem Weg zur automatischen Anwendungsentwicklung*, in Proceedings of GI-Fachtagung Software-technik'95 (Braunschweig, October 1995), 1995, pp. 126-136. http://www.swt.ruhr-uni-bochum.de/forschung/swt95/artikel.htm

[Bodart94b] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Vanderdonckt, J., *A Model-based Approach to Presentation: A Continuum from Task Analysis to*

Prototype, in Proceedings of 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94 (Bocca di Magra, 8-10 June 1994), F. Paternó (Ed.), Focus on Computer Graphics Series, Springer-Verlag, Berlin, 1995, pp. 77-94.

[Bodart95a] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacré, B., Vanderdonckt, J., *Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide*, in Proceedings of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95 (Château de Bonas, 7-9 June 1995), R. Bastide and Ph. Palanque (Eds.), Eurographics Series, Springer-Verlag, Vienna, 1995, pp. 262-278. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-019

[Bodart95b] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Vanderdonckt, J., *Computer-Aided Window Identification in TRIDENT*, in Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction INTERACT'95, Lillehammer, 25-29 June 1995, K. Nordbyn, P.H. Helmersen, D.J. Gilmore and S.A. Arnesen (Eds.), Chapman & Hall, London, 1995, pp. 331-336. http: //www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-021

[Coutaz94] Coutaz, J., Taylor, R.N., *Introduction to the Workshop on Software Engineering and Human-Computer Interaction: Joint Research Issues*, in Proceedings of the Software Engineering and Human-Computer Interaction ICSE'94 Workshop (Sorrento, 16-17 May 1995), J. Coutaz, R.N Taylor, (Eds.), Lecture Notes In Computer Science, Vol. 896, Springer-Verlag, Berlin, 1995, pp. 1-3.

[Curtis94] Curtis, B., Hefley, B., *A WIMP No More - The Maturing of User Interface Engineering*, ACM Interactions, Vol. 1, No. 1, 1994, pp. 22-34.

[EHCI95] « Engineering for Human-Computer Interaction », Proceedings of the 6th IFIP TC 2/WG 2.7 Working Conference on Engineering for Human-Com

[Elwert95] Elwert, T., Schlungbaum, E., *Modelling and Generation of Graphical User Interfaces in the TADEUS Approach*, in Proceedings of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95 (Château de Bonas, 7-9 June 1995), R. Bastide and Ph. Palanque (Eds.), Eurographics Series, Springer-Verlag, Vienna, 1995, pp. 193-208. http://www. informatik.uni-rostock.de/~schlung/TADEUS/paper/DSV-IS95.html

[Foley94] Foley, J.D., *History, Results and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based Systems for User Interface Design and Implementation*, in Proceedings of 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94 (Bocca di Magra, 8-10 June 1994), F. Paternó (Ed.), Focus on Computer Graphics Series, Springer-Verlag, Berlin, 1995, pp. 3-14.

[Gorny95] Gorny, P., *EXPOSE - An HCI-Counseling for User Interface Design*, in Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction IN-

TERACT'95, Lillehammer, 25-29 June 1995, K. Nordbyn, P.H. Helmersen, D.J. Gilmore and S.A. Arnesen (Eds.), Chapman & Hall, London, 1995, pp. 297-304.

[Hayes85] Hayes, P.J., Szekely, P.A., Lerner, R.A., *Design Alternatives for User Interface Management Systems Based on Expierence with COUSIN*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'85 (San Francisco, 14-18 April 1985), L. Borman, B. Curtis (Eds.), ACM Press, New York, 1985, pp. 169-175.

[Janssen93] Janssen, C., Weisbecker, A., Ziegler, J. , *Generating User Interfaces from Data Models and Dialogue Net Specifications*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993, pp. 418-423.

[Janssen96] Janssen, C., *Dialogentwicklung für objektorientierte, graphische Benutzungsschnittstellen*, Springer, Berlin, 1996. Also Ph.D. thesis, University of Stuttgart, 1996.

[Johnson92a] Johnson, J.A., *Selectors: Going Beyond User Interface Widgets*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'92 « Striking a balance » (Monterey, 3-7 May 1992), P. Bauersfeld, J. Bennett, G. Lynch (Eds.), ACM Press, New York, 1992, pp. 273-279.

[Johnson95] Johnson, P., Johnson, H., Wilson, S., *Rapid Prototyping of User Interfaces Driven by Task Models*, in « Scenario-Based Design: Envisioning Work and Technology in System Development », J. Carroll (Ed.), John Wiley & Sons, London, 1995, pp. 209-246.

[Kelly92] Kelly, C., Colgan, L., *User Modelling and User Interface Design*, in Proceedings of British Conference on Human-Computer Interaction HCI'92 « People and Computers VII », A. Monk, D. Diaper, M.D. Harrison (Eds.), Cambridge University Press, Cambridge, 1992, pp. 227-239.

[Lauridsen95] Lauridsen, O., *Generation of user interfaces using formal specification*, in Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction INTERACT'95, Lillehammer, 25-29 June 1995, K. Nordbyn, P.H. Helmersen, D.J. Gilmore and S.A. Arnesen (Eds.), Chapman & Hall, London, 1995, pp. 325-330.

[Märtin96a] Märtin, Ch., *Modellierung, Entwurf und automatische Konstruktion interaktiver Softwaresysteme*, Entwurf der modellbasierten Entwicklungsumgebung Application Modeling Environment (AME), Ph.D. thesis, University of Rostock, 1996.

[Märtin96b] Märtin, C., *Software Life Cycle Automation for Interactive Applications: The AME Design Environment*, in this volume, pp. 57-74.

[Morin90] Morin, D., *Working Group Discussion: Current Practice*, in Proceedings of Eurographics Workshop on User Interface Management Systems and Environments (Lisbon, June 1990), Duce, D.A., Gomes, M.R., Hopgood, F.R.A., Lee, J.R. (Eds.), Eurographics Seminars, Tutorial and perspectives in computer graphics, Springer-Verlag, 1990, pp. 51-56.

[Myers92] Myers, B.A., Rosson, M.B., *Survey on User Interface Programming*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'92 « Striking a balance » (Monterey, 3-7 May 1992), P. Bauersfeld, J. Bennett, G. Lynch (Eds.), ACM Press, New York, 1992, pp. 195-202.

[Myers94] Myers, B.A., *Challenges of HCI Design and Implementation*, Interactions, Vol. 1, No. 1, pp. 73-83.

[Myers95] Myers, B.A., *User Interface Software Tools*, ACM Transactions on Computer-human Interaction, Vol. 2, No. 1, March 1995, pp. 64-103.

[Olsen86] Olsen, D.R., *MIKE: The Menu Interaction Kontrol Environment*, In: ACM Transactions on Information Systems, Vol. 5, No. 4, pp. 318-344.

[Olsen93] Olsen, D.R., Foley, J.D., Hudson, S.E., Miller, J., Myers, B.A, *Research directions for user interface software tools*, Behaviour & Technology, Vol. 12, No. 2, 1993, pp. 81-97.

[Puerta94a] Puerta, A.R., Szekely, P., *Model-based Interface Development*, CHI'94 Tutorial Notes, 1994.

[Puerta94b] Puerta, A.R., Eriksson, H., Gennari, J.H., Musen, M.A., *Beyond Data Models for Automated User Interface Generation*, in Proceedings of British Conference on Human-Computer Interaction HCI'94 « People and Computers IX » (Glasgow, 23-26 August 1994), G. Cockton, S.W. Draper, G.R.S. Weir (Eds.), Cambridge University Press, Cambridge, 1994, pp. 353-366. http://www-ksl.stanford.edu/KSL_Abstracts/KSL-93-62.html

[Puerta96b] Puerta, A., *The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development*, in this volume, pp. 19-35.

[Reiterer94] Reiterer, H., *User Interface Evaluation and Design*, GMD-Report No. 237, Oldenbourg, 1994.

[Schreiber94b] Schreiber, S., *Specification and Generation of User Interfaces with the BOSS-System*, in Proceedings of the East-West International Conference on Human-Computer Interaction EWHCI'94 (St. Petersburgh, 1994), B. Blumenthal, J. Gornostaev, C. Unger (Eds.), Lecture Notes in Computer Sciences, Vol. 876, Springer-Verlag, Berlin, 1994, pp. 107-120. ftp://hpeick7.informatik.tu-muenchen.de/pub/papers/sis/ewhci94.ps.Z

[Sukaviriya93] Sukaviriya, P., Foley, J.D., Griffith, T., *A Second Generation User Interface Design Environment: The Model and the Runtime Architecture*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993, pp. 375-382

[Sukaviriya94] Sukaviriya, P., Muthukumarasamy, J., Frank, M., Foley, J.D., *A Model-based User Interface Architecture: Enhancing a Runtime Environment with Declarative Knowledge*, in Proceedings of 1st Eurographics Workshop on Design, Specification,

Verification of Interactive Systems DSV-IS'94 (Bocca di Magra, 8-10 June 1994), F. Paternó (Ed.), Focus on Computer Graphics Series, Springer-Verlag, Berlin, 1995, pp. 181-197.

[Szekely92] Szekely, P., Luo, P., Neches, R, *Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'92 « Striking a balance » (Monterey, 3-7 May 1992), P. Bauersfeld, J. Bennett, G. Lynch (Eds.), ACM Press, New York, 1992, pp. 507-514. http://www.isi.edu/isd/CHI92.ps

[Szekely93] Szekely, P., Luo, P., Neches, R., *Beyond Interface Builders: Model-Based Interface Tools*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993, pp. 383-390. http://www.isi.edu/isd/Interchi-be-yond.ps

[Szekely95] Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., Salcher, E., *Declarative interface models for user interface construction tools: the MASTERMIND approach*, in « Engineering for Human-Computer Interaction », Proceedings of the 6th IFIP TC 2/WG 2.7 Working Conference on Engineering for Human-Computer Interaction EHCI'95 (Grand Targhee Resort, 14-18 August 1995), L. Bass, C. Unger (Eds.), Chapman & Hall, London, 1995, pp. 120-150. http://www.isi.edu/isd/Mastermind/Papers/ ehci95.ps

[Vanderdonckt93] Vanderdonckt, J., Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993, pp. 424-429. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-93-005

[Vanderdonckt94d] Vanderdonckt, J., Ouedraogo, M., Yguietengar, B., *A Comparison of Placement Strategies for Effective Visual Design*, in Proceedings of British Conference on Human-Computer Interaction HCI'94 « People and Computers IX » (Glasgow, 23-26 August 1994), G. Cockton, S.W. Draper, G.R.S. Weir (Eds.), Cambridge University Press, Cambridge, 1994, pp. 125-143. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-94-019

[Vanderdonckt95b] Vanderdonckt, J., *Knowledge-Based Systems for Automated User Interface Generation: the TRIDENT Expierence*, Technical Report RP-95-010, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1995. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-010

[Weisbecker95] Weisbecker, A., *Ein Verfahren zur automatischen Generierung von software-ergonomisch gestalteten Benutzungsoberfläachen*, Springer, Berlin, 1995, Also Ph.D. thesis, University of Stuttgart, 1995.

[Wiecha90] Wiecha, C., Bennett, W., Boies, S., Gould, J., Green, S., *ITS: A Tool for Rapidly Developing Interactive Applications*, ACM Transactions on Information Systems, Vol. 8, No. 3, July 1990, pp. 204-236.

[Wilson96] Wilson, S., Johnson, P., *Bridging the Generation Gap: From Work Tasks to User Interface Designs*, in this volume, pp. 77-94.