Lecture Notes in Computer Science 5002

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Transforming Discourse Models to Structural User Interface Models

Sevan Kavaldjian, Cristian Bogdan, Jürgen Falb, and Hermann Kaindl

Institute of Computer Technology
Vienna University of Technology, Vienna, Austria
{kavaldjian,bogdan,falb,kaindl}@ict.tuwien.ac.at

Abstract. User-interface design is still a time consuming and expensive task to do, but recent advances allow generating them from interaction design models. We present a model-driven approach for generating user interfaces out of interaction design models. Our interaction design models are discourse models, more precisely models of classes of dialogues. They are based on theories of human communication and should, therefore, be more understandable to humans than programs implementing user interfaces. Our discourse models also contain enough semantics to transform them automatically into user interfaces for multiple devices and modalities. This paper presents a two-step transformation approach with an intermediate user interface model. By showing specific transformation rules, we concentrate on a major part of the first step, transforming discourse models to structural user interface models.

1 Introduction

In previous work [7], we have already been able to automatically generate usable user interfaces (UIs), even for multiple devices and for real-world applications. We generated such UIs from models, but since these models included finite-state machinery they were more in the spirit of abstract UIs (abstracting from the interaction modality) rather than high-level interaction design.

More recently, in the OntoUCP[1] project, we wanted to work with models that are more understandable to humans and possibly more easily to build. Therefore, we studied several theories of human communication from various fields. Based on insights from some of these theories, we focus on high-level specifications of discourse in the form of models. These models specify discourse in the sense of dialogues, where monologues are embedded and connected. According to the reference framework [3], these discourse models are located at the "task and concepts" level.

From our previous work, we inherit the use of *communicative acts* (and references to domain knowledge). Communicative acts are derived from Speech

[1] OntoUCP (A Unified Communication Platform both for Machine-Machine and Human-Machine Interaction based on Ontologies), partially funded by the FIT-IT Program of the Austrian FFG as project number 809254/9312. We also acknowledge the (financial) support of the PSE division of Siemens AG Österreich.

H. Giese (Ed.): MoDELS 2007 Workshops, LNCS 5002, pp. 77–88, 2008.

Act Theory [13] and express intentions in the sense of desired effects on the environment.

By integrating communicative acts with some results from *Rhetorical Structure Theory* (RST) [10] and *Conversation Analysis* [9], we developed a new discourse metamodel [1,5]. The metamodel defines what the discourse models should look like in our approach.

So, we strive for high-level modeling of discourse, including dialogues. Such a discourse model is inspired by human communication and serves as an interaction design for a traditional information system. Currently we do not support the generation of UIs with direct manipulation.

From such an interaction design, user interfaces for several devices are to be generated automatically. Since we knew already how to generate them from a kind of abstract UI model, we strived for generating UI models from our new interaction design models. In general, this involves partitioning of a given discourse tree, which is described in [1].

The remainder of this paper is organized in the following manner. First, we explain our model-driven transformation approach on the basis of self-defined metamodels. Then we focus on concrete transformation rules for mapping a discourse model to a structural UI model. Finally, we briefly discuss our approach as compared to related work.

2 Overall Approach

Our approach to fully automated UI generation is a two-step process. Model-to-model and model-to-code transformations are necessary to transform a discourse model to structural UI models, and further to multiple UIs for diverse platforms. In the following, we present the first step (see Figure 1) and explain the input (discourse model), the output (structural UI models) and the transformation rules for the model-to-model transformations.

Our discourse models use a self-defined Domain Specific Language (DSL) for specifying the classes of possible dialogues or interactions between the human and the machine. The abstract syntax of the DSL is based on the conceptual

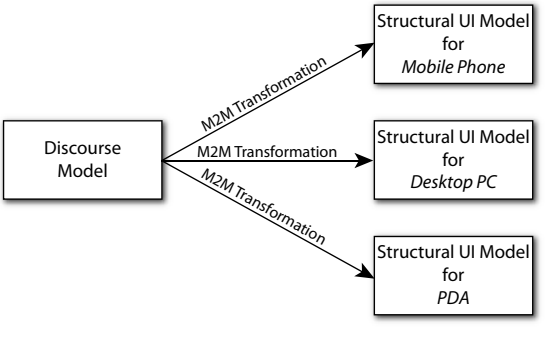


Fig. 1. The model-to-model transformation step

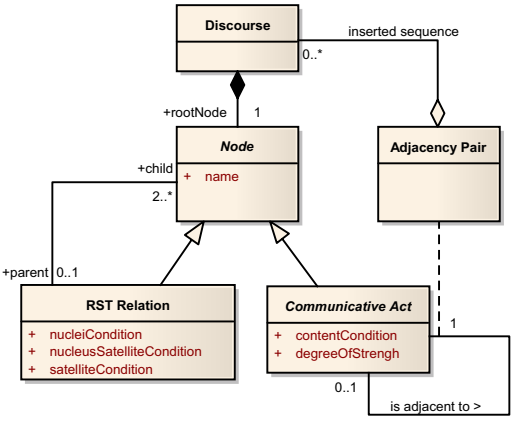


Fig. 2. Conceptual discourse metamodel

metamodel shown in Figure 2, which illustrates the concepts used. Every discourse is composed of a tree, where leaf nodes are *Communicative Acts* and inner nodes are *Rhetorical Relations* based on RST. The UML class diagram shown in Figure 2 is not as restrictive as our interpretation, since it would allow the creation of more general kinds of graphs than tree structures. The association class *Adjacency Pair* is needed to model Inserted Sequences.

The Communicative Acts are used to model the intention of a communication and refer to elements of the domain of discourse. Figure 3 shows a selection of the most important Communicative Acts used in our approach. Two corresponding Communicative Acts, like *Offer* and *Accept*, form a sequence, which is called *Adjacency Pair*. The Adjacency Pairs build up the dialogue structure.

The Rhetorical Relations are used to connect Communicative Acts or, again, Rhetorical Relations with each other. They represent the dependencies between single interactions of dialogues. Examples for Rhetorical Relations are *Condition*, *Joint* and *Background*. The *Condition* relation is used to model dependencies between adjacency pairs in the way that one branch (*satellite*) has to be executed and its Boolean expression fulfilled before the other (*nucleus*) can start. The *Joint* Relation is used to group Communicative Acts of the same type. No presentation order is presumed. The *Background* Relation is used to express that the *satellite* branch contains background information related to the *nucleus* branch.

Figure 4a shows a small part of an online shop discourse model, which we use as a running example throughout the paper. The example describes an interaction between the user and the online shop with the purpose of demanding the customer to select one product category and supporting her with background information to ease her choice. The *nucleus* branch N of the *Background* relation conveys the main interaction sequence. The online shop system *offers* a list of *product categories* to the user. The user *accept*s one of them. During the offering process the *satellite* branch S provides background information about

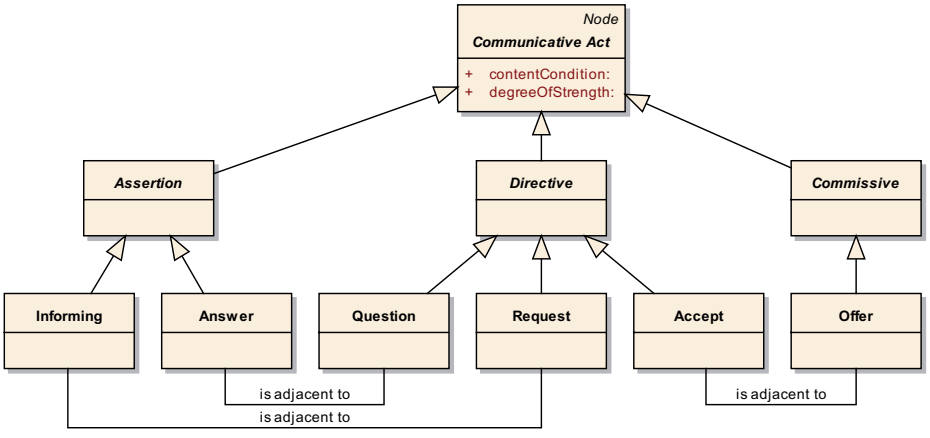
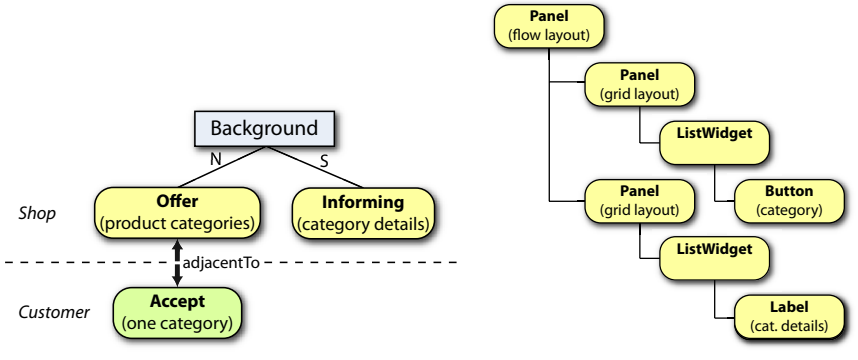


Fig. 3. Selection of communicative act taxonomy

the product categories to the user. This part of an online shop discourse model gets transformed to the structural UI model shown in Figure 4b by applying the rules *Light Background*, *Adjacency Pair*, *Offer-Accept* and *Informing* to the corresponding discourse model elements in the listed order. Details on each rule are described in section 3.

The structural UI model is basically a tree representing the UI structure independently of any toolkit (e.g., Web, Java Swing, etc.). It is not completely independent of the target device, however, since the device's real-estate is taken into account for building up the UI structure. Still, our structural user interface model is completely independent of the considered UI toolkit. This tree structure will be transformed to a toolkit-specific (final) UI. The concepts used in such a structural UI model are specified in the structural UI metamodel shown



(a) Subtree of an online shop discourse. (b) Resulting structural UI model.

Fig. 4. Online shop example showing a part of its discourse (a) and the resulting structural UI model (b) of its model-to-model transformation

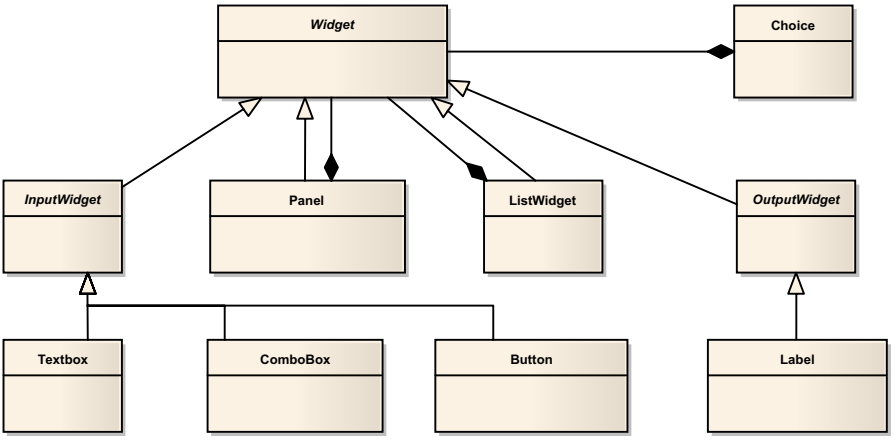


Fig. 5. Part of conceptual metamodel of structural UI models

conceptually in Figure 5. It only shows the parts that are important to our running example. The most important concept of the metamodel is the *Widget* class. It is specialized into four functional categories: *OutputWidget*s, *InputWidget*s, *ListWidget*s and *Panel*s. The *OutputWidget*s present information to the user in different ways like text and images and the *InputWidget*s gather information from the user. Nevertheless, *InputWidget*s also convey information to the user like defaults, current values and type and quantity of required information.

The main issue that we address in this paper is how to transfer models as exemplified in Figure 4a to structural user interface models at the abstract widget level as in Figure 4b. In particular, it means a transformation from a mainly declarative model of a discourse to the toolkit-independent structure of a user interface.

The general principle behind our approach is that the structural UI model is made up of "presentation" units that are made visible when the logic of the interaction with the user so requires. Once this principle is established, our problem can be specified as follows:

- Given a discourse tree with communicative acts as leafs, generate the possible set of presentation units, and the transitions between these presentation units. Since each presentation unit has to be a coherent discourse itself, it corresponds to a subtree of the overall discourse tree. As such, we call this problem *the discourse tree partitioning problem.* This problem and a solution to it is described in [1].
- Given a presentation unit as a discourse subtree, generate a structural UI model based on heuristic rules. Since this effectively "pre-renders" a discourse tree into a structural UI model, we call this problem *the pre-rendering problem.*

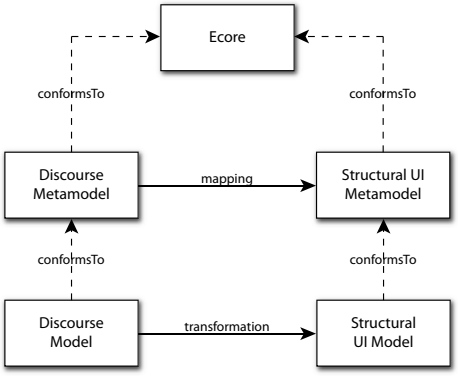


Fig. 6. The transformation process

In the structural UI model, a complete tree or subtree with a *Panel* as its root element represents a presentation unit. Hence, a complete structural UI model can, in general, be a forest consisting of possibly several trees. Trees in the structural UI model that are alternatives, i.e., trees on the same level resulting from discourse partitioning or Joint relations, will be linked in the structural UI model via a Choice element as shown in the metamodel in Figure 5. The Choice element is used to specify alternative presentation units, which can be used to fill in the same space. Our example in Figure 4a represents exactly one presentation unit that corresponds to the tree shown in Figure 4b.

Figure 6 illustrates that the transformation of one presentation unit is fulfilled by mapping elements of the discourse metamodel to elements of the structural UI metamodel. Both metamodels are based on the Ecore[2] meta-metamodel. Transformation languages like ATL[3] (ATLAS Transformation Language) or MOLA[4] (MOdel transformation LAnguage), which is used in this paper to graphically specify the transformation rules, support this transformation approach. At the same time a state machine is generated from the discourse model which controls the sending and receiving of Communicative Acts. This latter generation is beyond the scope of this paper, however.

3 Pre-rendering Rules for Transforming to a Structural UI Model

After having introduced the general transformation principles, we concentrate on the pre-rendering problem and introduce concrete rules for mapping a discourse model to a structural UI model. They are specific to certain structural patterns

[2] Essential MOF like core meta model of the Eclipse Modeling Framework (`http://www.eclipse.org/emf/`)
[3] `http://www.eclipse.org/m2m/atl/`
[4] `http://mola.mii.lu.lv/`

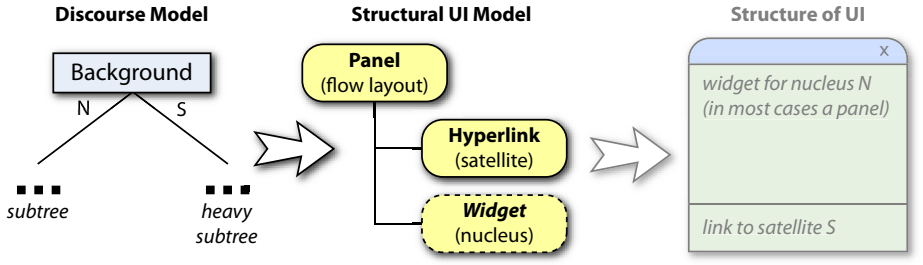


Fig. 7. Heavy Background Rule

occurring in the discourse models. We have found many such patterns during our modeling experience, and we continue to find new ones. Due to limited space, we only exemplify five rules which we believe illustrate the principle.

Heavy Background Rule: Figure 7 shows a rule for a "Heavy Background" relation, which associates a nucleus subtree with a "large" satellite subtree, i.e. a subtree that has a large number of nodes compared to the nucleus subtree and can lead to clutter if rendered in its entirety. The "nuclear" part is rendered directly since, as per the definition of RST nucleus, it is the most important part to convey, but if there is no space for its background information, the background information is rendered in a separate presentation unit. A link to the latter is presented together with the "nuclear" part, so an appropriate widget for a link (e.g., hyperlink, button) will be generated by the transformation process. Also, the action of the respective button can be generated to activate the separate presentation unit corresponding to the background.

Light Background Rule: Figure 8 shows another rule specific to the Background RST relation. The satellite is rendered on the right side of the presentation unit, while the nucleus occupies the left area, as an "aside". In accordance to the rule above, the "most nuclear part" takes the interface space that is of highest surface and most central to the user focus. Following this principle further, the layout management of the presentation unit will always give precedence to the "nuclear" side, e.g., when the window is resized by the user. This rule is used in our example to generate the basic tree structure of Figure 4b, i.e., this rules generates the root panel and places the transformation results of the pre-rendering of the nucleus and satellite subtrees next to each other by a flow layout manager. The Light Background Rule can also be localized (adapted), e.g., for cultures that write from right to left, where it may be more suitable to place the satellite at the left side. A system-wise style configuration can also render light backgrounds to the top and to the left, like it is, e.g., customary when the concrete user interface will be an HTML page. However, even there the space allocation and re-allocation in case of resizing will prioritize the nuclear part.

Figure 9 illustrates the formalized Light Background Rule in MOLA, composed of the following elements. The outer bold rectangle symbolizes a *for-each* loop. The rounded rectangle inside represents the actual rule that will be repeated for

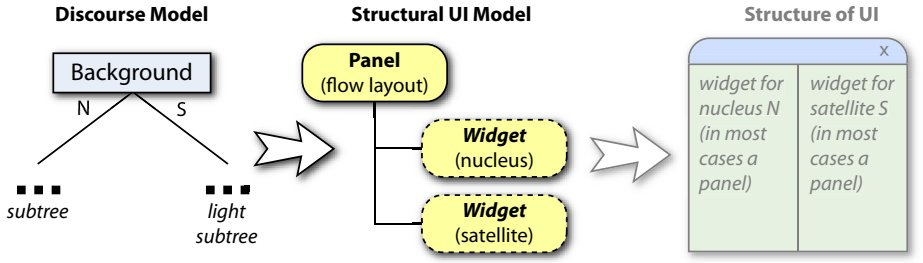


Fig. 8. Light Background Rule

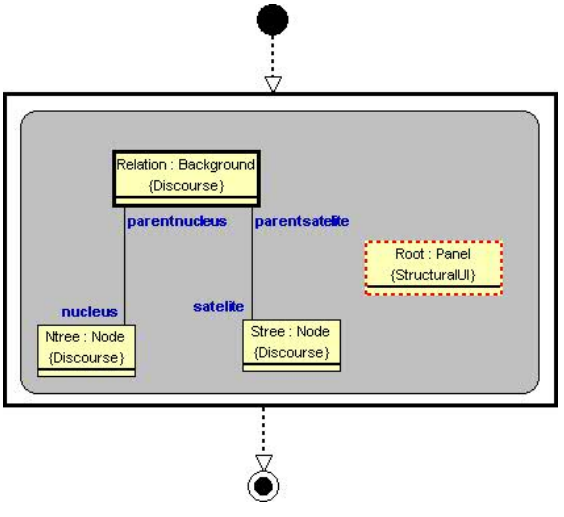


Fig. 9. Light Background Rule

each matched element. The small boxes inside the rule represent different kinds of classes, depending on their borderline style. When the thickness of the border line is regular, they represent a "normal" class. A bold border lined box represents a loop variable. A dashed lined box border represents a class that will be created by the transformation rule. The small black circle represents the starting point of the rule. The double rounded circle represents the end point of the rule. In particular, the Light Background rule iterates over all Background RST relations. Whenever a Background relation connects a nucleus tree and a satellite tree, the rule matches. In this case, a root *Panel* element of the structural UI model is generated. Next the execution is handed over to the rule responsible for the nucleus and subsequently to the rule transforming the satellite. Both rules get the panel as a parameter, so that they can add their results to the panel.

Adjacency Pair Rule: Each adjacency pair is transformed to a *Panel* element of the structural UI model containing widgets according to the related commu-

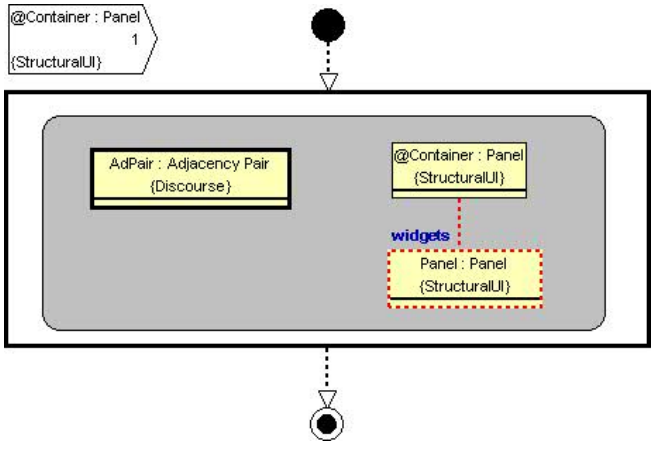


Fig. 10. Adjacency Pair Rule

nicative acts. In our example, the first panel on the second level in Figure 4b results from the application of the Adjacency Pair Rule to the Offer-Accept adjacency pair.

Figure 10 illustrates the formalized Adjacency Pair Rule in MOLA. It iterates over all Adjacency Pairs of the Discourse Model and creates a *Panel* element each. It also creates an association to the parent panel which the rule gets as a parameter. In our example, the Adjacency Pair Rule adds the created panel to the root panel of Figure 4b.

Offer-Accept Rule: Each *Offer-Accept* adjacency pair is transformed either to a *Button* or to a *ListWidget* element containing *Button*s, depending on the cardinality of the content offered. Because our example online shop can offer more than one product category, the *ListWidget* element is needed to model an undefined number of categories. Since the acceptance of an *Offer* requires a user action, a *Button* element is embedded in the *ListWidget* in Figure 4b. As a result, the subtree of the *ListWidget* is repeated according to the actual number of product categories in the final UI.

Figure 11 illustrates the formalized Offer-Accept Rule in MOLA. It iterates over all *Offer*s that are associated with an *Accept* via an *Adjacency Pair*. The parameter container of type panel represents the parent panel and allows the rule to add the generated widgets to the panel. Depending on the type of the communicative act's content, one of two alternative rules is selected. If the content is a kind of List (consisting of more than one element) a *ListWidget* and a *Button* are generated. In all other cases only a *Button* is generated in the structural UI model.

Informing Rule: Each *Informing* communicative act is transformed either to a *Label* element or to a *ListWidget* element containing a *Label* element, depending on the cardinality of the content. This rule assumes that the information will be

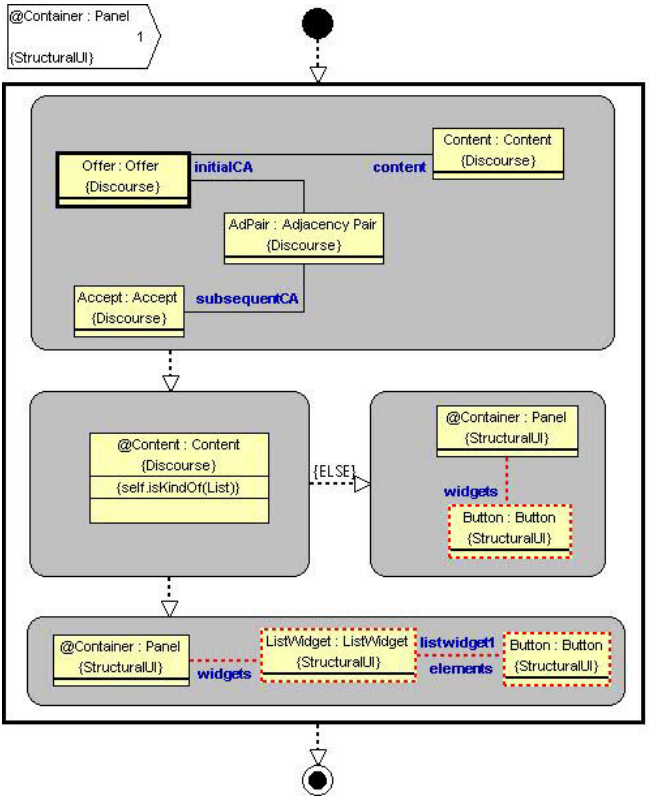


Fig. 11. Offer-Accept Rule

forwarded in textual form, otherwise, e.g., a *PictureBox* or *AudioPlayer* element will be used. In the online shop example, information is conveyed for each product category and, therefore, a *ListWidget* containing *Label*s is generated.

If a communicative act is not part of an adjacency pair, as it is the case with the Informing in Figure 4a, a *Panel* element is also created as a container for the communicative act.

More detailed information about the automatic generation that is used as a basis for this approach can be found in [6,7].

4 Related Work

Model-based UI design methods developed and published in the nineties including OVID [12], and Idiom [14] focus on creating different kinds of models, like user's conceptual models, task models and interaction models. Unlike our approach, which is *model-driven*, all the mentioned approaches above are *model-based*. That is, they allow expressing an interactive system by task, concept

and/or abstract models in a first step and use them in an informal process or in a sequence of systematic steps to construct a user interface.

In contrast, UI Frameworks like XUL[5] (XML User Interface Language) are able to generate UIs automatically but they rely on UI models at the abstract widget level, which is on a lower level than our discourse models.

An advanced approach to specifying multi-device user interfaces based on task models instead of discourse models is presented in [11]. Its basic approach is to start modeling tasks and to generate user interfaces for diverse devices according to specific device characteristics. In contrast to our approach, some of the transformations between models are done semi-automatically or manually. A major difference between task models and our discourse models is that task models express richer temporal dependencies whereas our discourse models specify causal dependencies, too. The semantic mapping of task models to dialog models based on UML State Machines is explained in [4]. A similar approach is used to generate the UI behavior from our discourse models.

Florins *et al.* describe in [8] transformation rules for pagination of UIs on different levels. Partitioning our discourse models into presentation units in the first transformation step provides important guidance for pagination [1].

Botterweck shows in [2] a model-driven approach that starts on the abstract UI level, but contains rich procedural UI descriptions together with UI elements. Thus, it requires UI modeling as well as dialogue modeling.

5 Conclusion

In this paper, we present a new approach to generating structural user interface models by applying model-driven transformations to discourse models. Our discourse models are derived from results of human communication theories, cognitive science and sociology and are used for specifying interaction design of human-computer interaction of information systems. Thus, they contain additional metainformation, like the intention of an interaction, which allows us to define sophisticated pre-rendering rules to transform the discourse models to structural UI models. Our transformation takes already device constraints into account to generate a UI structure well suited for the target device, but the resulting UI models are still independent of UI toolkits. Taking this together with our previous work on automatically generating UIs from abstract models, this paves the way for automatic generation of UIs from our new interaction design models.

References

1. Bogdan, C., Falb, J., Kaindl, H., Kavaldjian, S., Popp, R., Horacek, H., Arnautovic, E., Szep, A.: Generating an abstract user interface from a discourse model inspired by human communication. In: Proceedings of the 41th Annual Hawaii International Conference on System Sciences (HICSS-41), Piscataway, NJ, USA, IEEE Computer Society Press, Los Alamitos (2008)

[5] `http://www.mozilla.org/projects/xul/`

2. Botterweck, G.: A model-driven approach to the engineering of multiple user interfaces. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, Springer, Heidelberg (2006)
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting With Computers 15/3, 289–308 (2003)
4. den Bergh, J.V., Coninx, K.: From task to dialog model in the UML. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 98–111. Springer, Heidelberg (2007)
5. Falb, J., Kaindl, H., Horacek, H., Bogdan, C., Popp, R., Arnautovic, E.: A discourse model for interaction design based on theories of human communication. In: CHI 2006 extended abstracts on Human factors in computing systems CHI 2006 (2006)
6. Falb, J., Popp, R., Röck, T., Jelinek, H., Arnautovic, E., Kaindl, H.: Using communicative acts in interaction design specifications for automated synthesis of user interfaces. In: Proceedings of the 21th IEEE/ACM International Conference on Automated Software Engineering (ASE 2006), Piscataway, NJ, USA, pp. 261–264. IEEE Computer Society Press, Los Alamitos (2006)
7. Falb, J., Popp, R., Röck, T., Jelinek, H., Arnautovic, E., Kaindl, H.: Fully-automatic generation of user interfaces for multiple devices from a high-level model based on communicative acts. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS-40), Piscataway, NJ, USA, January 2007, IEEE Computer Society Press, Los Alamitos (2007)
8. Florins, M., Simarro, F.M., Vanderdonckt, J., Michotte, B., Michotto, B.: Splitting rules for graceful degradation of user interfaces. In: AVI 2006: Proceedings of the working conference on Advanced visual interfaces, pp. 59–66. ACM Press, New York (2006)
9. Luff, P., Gilbert, N., Frohlich, D.: Computers and Conversation. Academic Press, London (1990)
10. Mann, W.C., Thompson, S.: Rhetorical Structure Theory: Toward a functional theory of text organization. In: Text, pp. 243–281 (1988)
11. Mori, G., Paterno, F., Santoro, C.: Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Transactions on Software Engineering 30(8), 507–520 (2004)
12. Roberts, D., Berry, D., Isensee, S., Mullaly, J.: Developing software using OVID. IEEE Software 14(4), 51–57 (1997)
13. Searle, J.R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, Cambridge (1969)
14. van Harmelen, M.: Object oriented modelling and specification for user interface design. In: Interactive Systems: Design, Specification and Verification (1994)

Holger Giese (Ed.)

Models in Software Engineering

Workshops and Symposia at MoDELS 2007
Nashville, TN, USA, September 30 - October 5, 2007
Reports and Revised Selected Papers

Springer

Volume Editor

Holger Giese
Hasso-Plattner-Institut
für Softwaresystemtechnik
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
E-mail: holger.giese@hpi.uni-potsdam.de

Library of Congress Control Number: 2008930116

CR Subject Classification (1998): D.2, D.3, I.6, K.6

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-69069-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-69069-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12282005 06/3180 5 4 3 2 1 0

Preface

Following the tradition of previous instances of the MoDELS conference series, 11 workshops and two symposia were hosted in 2007. These satellite events complemented the main conference by providing room for important subject areas and enabling a high degree of interactivity.

The selection of the workshops was organized like in former instances of the MoDELS conference series by a Workshop Selection Committee. The following well-known experts agreed to serve on this committee:

- Gabor Karsai, Vanderbilt University, USA
- Thomas Kühne, Darmstadt University of Technology, Germany
- Jochen Küster, IBM Research Zurich, Switzerland
- Henry Muccini, University of L'Aquila, Italy
- Sebastian Uchitel, Imperial College London, UK

The workshops provided collaborative forums for particular topics. They enabled a group of participants to exchange recent and/or preliminary results, to conduct intensive discussions, or to coordinate efforts between representatives of a technical community. They served as forums for lively discussion of innovative ideas, recent progress, or practical experience on model-driven engineering for specific aspects, specific problems, or domain-specific needs.

As in previous editions, there were a Doctoral Symposium and an Educators' Symposium. The Doctoral Symposium provided specific support for PhD students to discuss their work and receive useful guidance for the completion of their dissertation research. The Educators' Symposium addressed how to educate students as well as practitioners to move from traditional thinking to an engineering approach based on models.

These satellite-event proceedings published after the conference include summaries as well as revised versions of up to two best papers from the workshops, the Doctoral Symposium, and the Educators' Symposium.

I am grateful to the members of the Selection Committee who did a great job in reviewing the workshop proposals and selecting the best workshops. In particular Thomas Kühne (member of the Selection Committee and my predecessor as Workshop Chair) was of great help to me and eased my work by generously sharing his experiences from the former year with me.

February 2008 Holger Giese

Organization

Sponsoring Institutions

ACM Special Interest Group on Software Engineering
(`www.sigsoft.org`)

IEEE Computer Society
(`www.computer.org`)

Table of Contents

Aspect-Oriented Modeling

Language Engineering

Model Driven Development of Advanced User Interfaces

Model Size Metrics

Model-Based Design of Trustworthy Health Information Systems

Model-Driven Engineering, Verification and Validation

Ocl4All: Modelling Systems with OCL

Models@run.time

Multi-Paradigm Modeling: Concepts and Tools

Quality in Modeling

Doctoral Symposium

Educators' Symposium