Automated Generation of User Interfaces
Based on Use Case or Interaction Design Specifications?

Hermann Kaindl, Roman Popp and David Raneburger
Institute of Computer Technology, Vienna University of Technology, Gusshausstrasse 27-29, 1040 Vienna, Austria

Keywords: Modeling Languages, Model Transformation.

Abstract: Instead of manually creating (graphical) user interfaces (UIs), automatically generating them is desirable, especially since UIs are needed today for diverse devices like PCs and smartphones. The basis for such automated generation can be a UI specification, but most of the related research takes task models as input, which are on a higher level of abstraction. More recently, another modeling language employing discourse-based models for specifying communicative interaction has been proposed, which completely abstracts from specifics of a particular UI and even its type. From such models, UIs can be generated automatically through model transformations. Some research, however, claims that UIs can be generated from use cases. While it would be desirable to utilize such a form of requirements definition without having to create another specification, we found that these approaches actually use additional information attached to the use cases, usually UI-related attachments. In addition to contrasting different kinds of specifications, we propose a synthesis through a combination. In fact, we found that discourse-based models can be also viewed as specifying classes of scenarios, i.e., use cases more precisely than the main-stream approach using UML and use-case reports.

1 INTRODUCTION

Automated generation of user interfaces (UIs) needs to be based on some input specification. This can be a UI specification that abstracts from the peculiarities of a real UI, but specifies already the kind of UI (e.g., a graphical UI), and possibly also the kinds of widgets. A higher abstraction can be achieved with so-called task models, e.g., see (Costa et al., 2007; van den Bergh and Coninx, 2007; Pastor et al., 2008; Mori et al., 2002; Paternò et al., 2009; Vanderdonckt, 2008). A more recent approach employs discourse-based models, e.g., see (Falb et al., 2006; Falb et al., 2009). From higher-level models, usually model transformations lead to lower-level models and, with the help of templates to final code implementing a UI. Alternatively, use cases are considered as the basis for automated UI generation, see (Elkoutbi et al., 2006; Fatolahi et al., 2008; Hennicker and Koch, 2001; da Silva and Paton, 2003).

In this paper, we discuss these possibilities using a simplified version of flight-booking as a running example. First, we elaborate on the question, whether use cases are enough for automated generation of UIs. Then we consider discourse-based models. Finally, we propose a combination.

2 ARE USE CASES ENOUGH?

First, let us figure out whether the information available in use case specifications is sufficient for automated generation of user interfaces. According to the *Unified Process* (UP) and mainstream practice, use case diagrams such as the one in Figure 1 are employed for providing an overview of the use cases of a given application, their relations among each other and with actors (e.g., human users of the application). For our simple running example, this amounts to the use case *book flight*, which *includes* the use cases *select flight* and *buy ticket*. *Customer* is the actor of these use cases.

Such a use case often consists of a class of scenarios, in the sense of sequences of actions involving

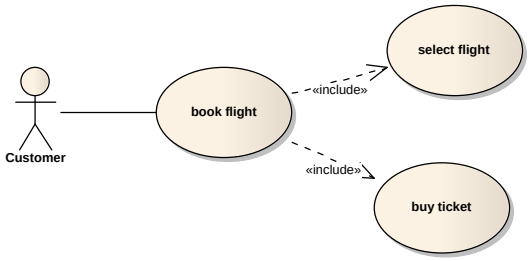

Figure 1: Flight-booking Use Case Diagram.

Kaindl H., Popp R. and Raneburger D..
Automated Generation of User Interfaces - Based on Use Case or Interaction Design Specifications?.
DOI: 10.5220/0004136203030308
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 303-308
ISBN: 978-989-8565-19-8

the actor and the system to be built. Such scenarios are either described in (stylized) natural language text or modeled in sequence diagrams such as the one in Figure 2, or both. This example sequence diagram specifies the main scenario of the use case *book flight*. Note, that this is typically just an example of use and not a general specification of all the possible interactions of a use case.

For a more comprehensive documentation of use cases, the UP defines so-called use-case reports, which consist of the following information:

1. Brief Description
2. Flow of Events
3. Special Requirements
4. Pre-conditions
5. Post-conditions
6. Extension Points
7. Relationships
8. Use-Case Diagrams
9. Other Diagrams

While a few variations of the main scenario are usually given under Flow of Events, rarely a comprehensive specification of all the possible interactions of a use case is available. Even if, it is given informally in natural language (with a few encodings), much as the use-case report overall.

Is this sufficient information for automated generation of a user interface?

We think that a major problem involved is that the given scenarios are just examples. In early work (Kaindl and Jezek, 2002), a systematic decomposition of the overall task into tasks assigned to the steps in a given scenario was proposed, their composition to *interaction tasks* etc., and finally the assignment of widget classes to such interaction tasks. In this way, all the possible variations of performing the interaction tasks are covered, but this approach has to be done manually by a human instead of automated generation.

However, a few papers claim that user interfaces can be generated automatically from use cases, see (Elkoutbi et al., 2006; Fatolahi et al., 2008; Hennicker and Koch, 2001; da Silva and Paton, 2003). We found that use cases in these approaches actually have additional information attached, usually through annotations with extra information on user interface aspects.

So, we conclude that use case specifications without extra information are not enough for generating user interfaces automatically.

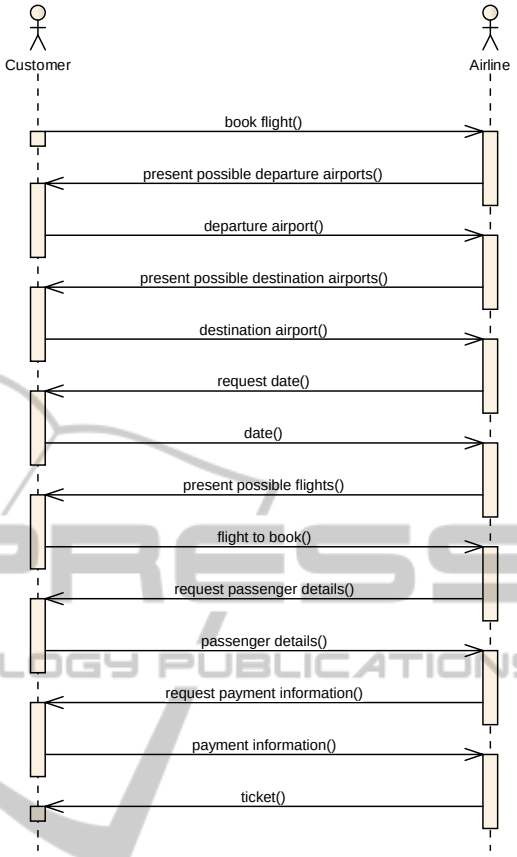

Figure 2: Flight-booking Sequence Diagram.

3 HOW ABOUT AN INTERACTION DESIGN SPECIFICATION?

Let us contrast this with our high-level specification of communicative interaction specifically developed for automated generation of UIs, e.g., see (Falb et al., 2006; Falb et al., 2009; Raneburger et al., 2011a). This kind of specification is on a high level of abstraction in the sense, that it is not (yet) a UI specification, as it does not contain any concrete clues for a real UI. It does not even specify whether a graphical or, e.g., a speech UI will be generated based on it.

The main part of such an interaction specification is a so-called Discourse Model, see the example in Figure 3. It specifies all possible (communicative) interactions through discourses in the sense of dialogues in terms of *Communicative Acts*, *Adjacency Pairs* (Luff et al., 1990), *Rhetorical Structure Theory (RST)* (Mann and Thompson, 1988) relations and procedural constructs. The green and yellow rounded boxes represent Communicative Acts (a generaliza-

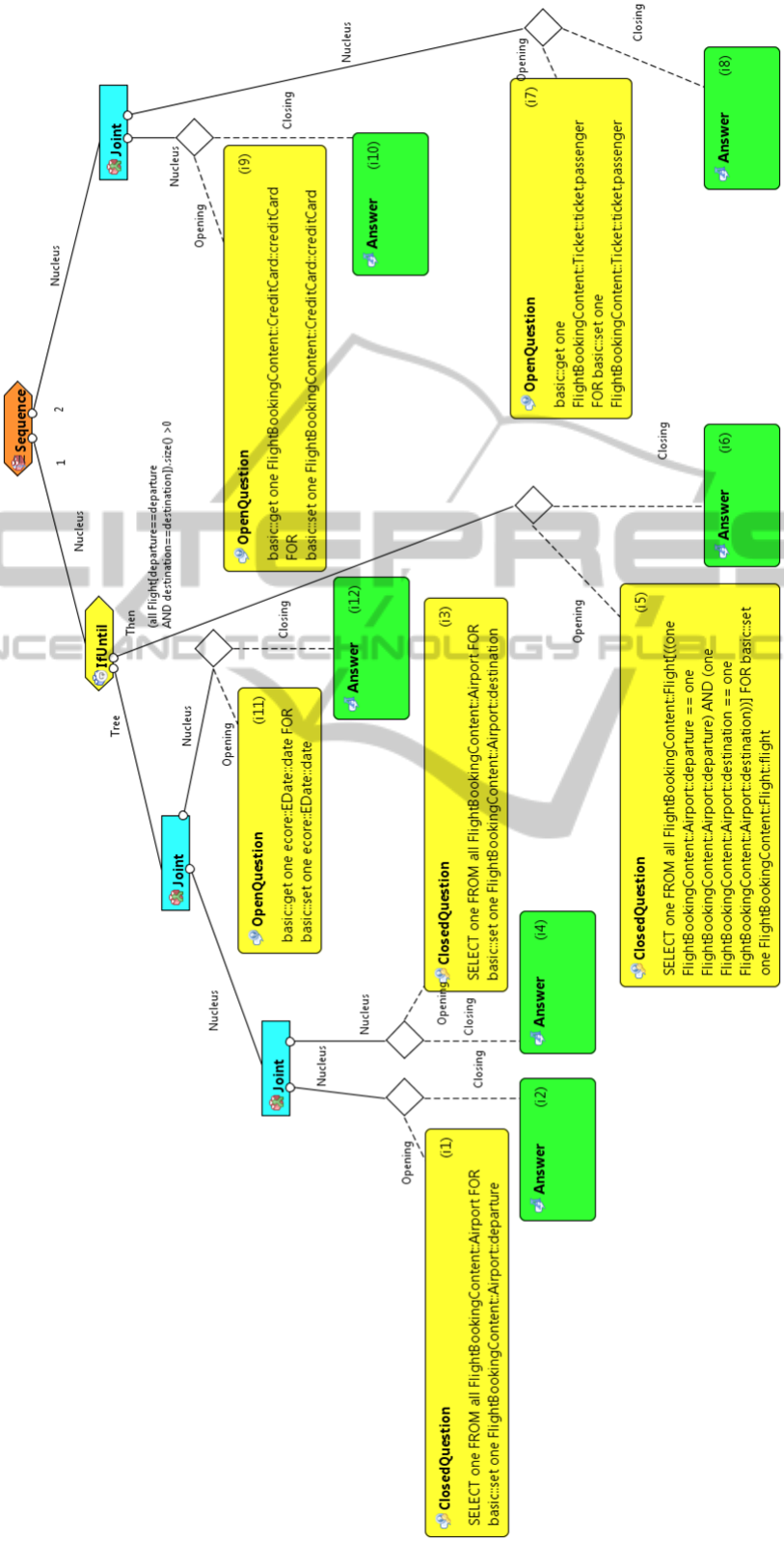

Figure 3: Flight-booking Discourse Model.

tion of so-called Speech Acts (Searle, 1969)). There are two colors for two dialogue partners. A question is related with an answer through an Adjacency Pair (represented through a diamond symbol). A tree results from recursively connecting Adjacency Pairs with RST relations (such as *Joint* in our example) or procedural constructs (such as *Sequence*). The leftmost *Joint* in Figure 3 means that the related questions may be asked concurrently and that both of them will have to be answered. *Sequence* means that the connected trees will have to be elaborated in this given sequence.

Communicative Acts refer to a specification of what they "talk about" — the domain of discourse. The Domain-of-Discourse Model for our running example can be found in Figure 4. It contains, e.g., classes *Airport*, *Flight*, etc. Note the ontological difference to a class model of a use-case realization, which actually contains software design classes. In contrast, a Domain-of-Discourse Model specifies those classes of objects in the real-world domain that the application can communicate about in its discourses with the human user through the UI. A software design model may contain these or similar classes, of course, but the exact model for the implementation in software may also be different for various reasons.

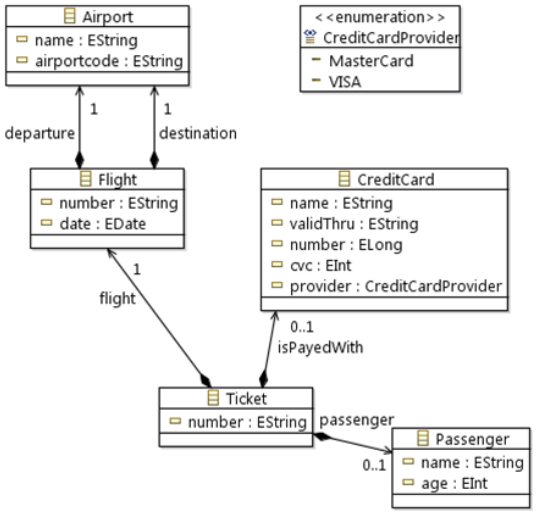

Figure 4: Flight-booking Domain-of-Discourse Model.

Finally, there needs to be a connection to the functionality provided by the software application. So, there is yet another model involved that specifies the interfaces of methods of the application logic, see (Popp and Raneburger, 2011). Note the relation of such a specification with use cases.

From such models, it is possible to automatically generate user interfaces through model transformations (Falb et al., 2009; Falb et al., 2011). They are not necessarily always usable enough for practical application when fully automatically generated for large screens. For relatively small screens of todays smartphones, however, they can be specifically optimized, see (Raneburger et al., 2011b). Figure 5 shows how such automatically generated UIs look on an iPhone.

So, these models, e.g., contain enough information for generating user interfaces based on them.

4 HOW ABOUT A COMBINATION?

At least on the surface, such models look quite different from common use case / scenario models. So, there may be a danger of separation of requirements specification from interaction design. Is there a useful way of combining such models?

In fact, we found that the discourse-based models sketched above specify all the possible flows of communicative interaction in the course of such dialogues. This is missing in the common use case approach of a main scenario example with a few variations. We can view it as a specification of the class of all possible dialogues. These possible flows are well defined and understandable in the sense that they are specified using concepts common in human communication (e.g., questions and answers). Moreover, there is additional information in RST relations and procedural constructs. So, combining such models with common use-case reports results in a more complete specification of the use cases, and it facilitates automated generation of user interfaces.

Application of this combined approach in a project together with a company provided some empirical evidence of its usefulness. This company provided requirements in a specification based on the common use-case approach. According to these, we developed related discourse-based models, which specified all possible communicative interactions / scenarios. Our tools supporting this approach facilitated the editing of the models and their preliminary verification, and they generated optimized graphical user interfaces for smartphones.

In future work, we plan to extend our tool support and to investigate typical interaction patterns that can be provided as templates for the interaction designer. We also plan to work on the scalability of our Communication Models. They capture all possible flows of events and, therefore, potentially encompass several use cases. Sub-communication Models may facilitate a decomposition analogously to the one of use cases. We plan to link these sub-models resulting in

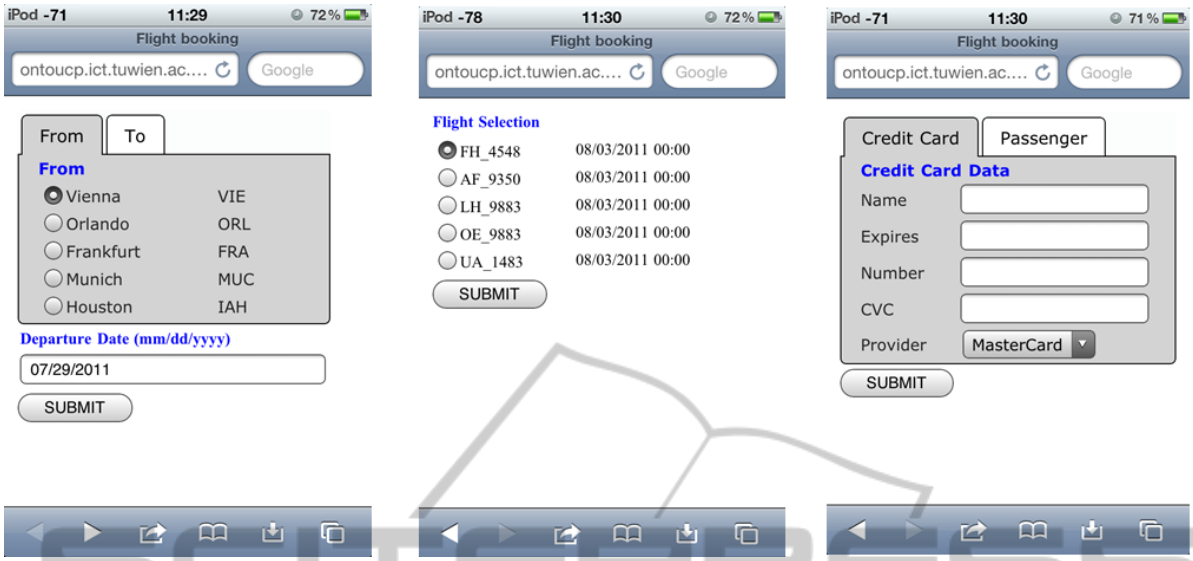

Figure 5: Final UI for iPod Touch and iPhone.

the overall Communication Model. A hierarchically structured model will be easier to manage and the sub-models will also facilitate reuse.

Our Discourse-based Communication Models provide a specification of the communicative interaction between two dialogue partners. We will investigate Habermas' Communicative Action Theory (Habermas, 1984; Habermas, 1987), especially its application to understanding Information Systems (Cecez-Kecmanovic and Janson, 1999), to develop a workflow for the creation of our Communication Models. Another interesting point based on Habermas' work may be the negotiated combination of (Sub-)Communication Models to new Communication Models according to our modeling approach.

5 CONCLUSIONS

So, should automated generation of UIs be based on use case or interaction design specifications? If this was an alternative, it can realistically only be based on interaction design specifications, such as task models or discourse-based models.

The latter models can be also viewed as specifying classes of scenarios, i.e., use cases. Therefore, these models can be usefully combined with more common use-case reports, in order to achieve a more complete specification of use cases, in particular of the possible flows of actions. This combination has the potential to make applications both more useful and usable.

ACKNOWLEDGEMENTS

We thank Points Management GmbH for sponsoring part of this research in the context of a project partially funded by the Austrian FFG.

REFERENCES

Cecez-Kecmanovic, D. and Janson, M. (1999). Communicative action theory: An approach to understanding the application of information systems. In *Proceedings of the Tenth Australasian Conference on Information Systems (ACIS99)*, pages 183–195.

Costa, D., Nóbrega, L., and Nunes, N. (2007). An MDA approach for generating web interfaces with UML ConcurTaskTrees and canonical abstract prototypes. In Coninx, K., Luyten, K., and Schneider, K., editors, *Task Models and Diagrams for Users Interface Design*, volume 4385 of *Lecture Notes in Computer Science*, pages 137–152. Springer Berlin / Heidelberg. 10.1007/978-3-540-70816-2_11.

da Silva, P. and Paton, N. (2003). User interface modeling in UML*i*. *IEEE Software*, 20(4):62–69.

Elkoutbi, M., Khriss, I., and Keller, R. K. (2006). Automated prototyping of user interfaces based on UML scenarios. *Automated Software Engineering*, 13(1):5–40.

Falb, J., Kaindl, H., Horacek, H., Bogdan, C., Popp, R., and Arnautovic, E. (2006). A discourse model for interaction design based on theories of human communication. In *Extended Abstracts on Human Factors in Computing Systems (CHI '06)*, pages 754–759. ACM Press: New York, NY.

Falb, J., Kaindl, H., Popp, R., and Raneburger, D. (2011). Automated WIMP-UI generation based on communication models. *i-com*, 10(3):48–55.

Falb, J., Kavaldjian, S., Popp, R., Raneburger, D., Arnautovic, E., and Kaindl, H. (2009). Fully automatic user interface generation from discourse models. In *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI '09)*, pages 475–476. ACM Press: New York, NY.

Fatolahi, A., Som, S. S., and Lethbridge, T. C. (2008). Towards a semi-automated model-driven method for the generation of web-based applications from use cases. In *Proceedings of the workshop on Speech and Natural Language*.

Habermas, J. (1984). *The Theory of Communicative Action - Reason and the Rationalisation of Society*, volume I. Boston, MA: Beacon Press.

Habermas, J. (1987). *The Theory of Communicative Action - the Critique of a Functionalist Reason*, volume II. Boston, MA: Beacon Press.

Hennicker, R. and Koch, N. (2001). Modeling the user interface of web applications with UML. In *Workshop of the pUML-Group held together with the «UML»2001 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists*, pages 158–172. GI.

Kaindl, H. and Jezek, R. (2002). From usage scenarios to user interface elements in a few steps. In Kolski, C. and Vanderdonckt, J., editors, *Proceedings of CADUI'02*, pages 91–102. Kluwer.

Luff, P., Frohlich, D., and Gilbert, N. (1990). *Computers and Conversation*. Academic Press, London, UK.

Mann, W. C. and Thompson, S. (1988). Rhetorical Structure Theory: Toward a functional theory of text organization. *Text*, 8(3):243–281.

Mori, G., Paternò, F., and Santoro, C. (2002). CTTE: Support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28:797–813.

Pastor, O., España, S., Panach, J. I., and Aquino, N. (2008). Model-driven development. *Informatik Spektrum*, 31(5):394–407.

Paternò, F., Santoro, C., and Spano, L. D. (2009). Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16:19:1–19:30.

Popp, R. and Raneburger, D. (2011). A High-Level Agent Interaction Protocol Based on a Communication Ontology. In Huemer, C., Setzer, T., Aalst, W., Mylopoulos, J., Sadeh, N. M., Shaw, M. J., and Szyperski, C., editors, *E-Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, pages 233–245. Springer Berlin Heidelberg. 10.1007/978-3-642-23014-1_20.

Raneburger, D., Popp, R., Kaindl, H., Falb, J., and Ertl, D. (2011a). Automated Generation of Device-Specific WIMP UIs: Weaving of Structural and Behavioral Models. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '11, pages 41–46, New York, NY, USA. ACM.

Raneburger, D., Popp, R., Kavaldjian, S., Kaindl, H., and Falb, J. (2011b). Optimized GUI generation for small screens. In Hussmann, H., Meixner, G., and Zuehlke, D., editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 107–122. Springer Berlin / Heidelberg.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, England.

van den Bergh, J. and Coninx, K. (2007). From task to dialog model in the UML. In *Proceedings of the 6th International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA 2007), LNCS 4849*, pages 98–111, Toulouse, France. Springer.

Vanderdonckt, J. M. (2008). Model-driven engineering of user interfaces: Promises, successes, and failures. In *Proceedings of 5th Annual Romanian Conf. on Human-Computer Interaction*, pages 1–10. Matrix ROM.