Design Alternatives for GUI Development with Discourse-based Communication Models

David Raneburger, Roman Popp and Hermann Kaindl
Institute of Computer Technology, Vienna University of Technology
Gusshausstrasse 27-29, A–1040 Vienna, Austria
{raneburger, kaindl, popp}@ict.tuwien.ac.at

Abstract: In general, design has to take many alternatives into account. In particular, for a given interaction design many potential implementations of graphical user interfaces (GUIs) exist. Additionally, for a given problem many different design alternatives are possible. We address this latter issue in this paper and investigate different ways for achieving certain information exchange between two interacting parties. Our concrete approach to communicative interaction design in this context deals with Discourse-based Communication Models of such design alternatives. We show that for more or less the same problem, different alternatives may be preferable, even depending on the required information exchange between the GUI and the application logic. We also sketch how typical GUIs for such design alternatives may look like, but without elaborating on alternatives in the course of such rendering.

1 Introduction

In the context of this paper, design alternatives define a design space of different ways for achieving similar information exchange between two interacting parties. However, we focus here on alternatives on a high level of abstraction and refer the reader to our approach for dealing with alternative GUIs in the course of generating them [RPK+11, RPK13].

Even on the high level of abstraction, design alternatives differ in how certain information is exchanged and can thus be more or less suitable for a given context (e.g., application scenario). We explain the different pros and cons of the presented alternatives and, in effect, provide design rationale for them. As a running example, we use (simplified) flight booking through the Web. It is intentionally kept simple, in order to focus on the essence of each design alternative. The various design structures can certainly be part of larger interaction designs in practice and may be viewed as a step towards design patterns.

In particular, we investigate three issues:

1. Which design alternatives can be modeled with Discourse-based Communication Models for a certain given problem?
2. Which impact do these alternatives have on a resulting GUI?
3. For which context is a certain alternative better suited than the others?

In principle, design alternatives on this level of abstraction are independent of a certain GUI generation approach. Since we use the modeling approach based on Discourse-based Communication Models [FKPR11, PRK13], we assume the Discourse-based GUI Generation Framework[1] (UCP:UI) for rendering GUIs. Our illustrations of different GUI structures, however, are only sketches here, in order to avoid potential bias to the current rendering of Final User Interfaces by this Framework.

The remainder of this paper is organized in the following manner. First, we provide some background material and discuss related work. Then we present several design alternatives in the form of Discourse-based Communication Models for (simplified) flight booking: concurrent, sequential and incremental discourse design. Finally, we discuss this approach more generally and indicate future work.

2 Background and Related Work

In order to make this paper self-contained, we present some background information on Discourse-based Communication Models. After that, we provide a short overview on state-of-the-art interaction modeling approaches for automated generation GUI generation, which could be used alternatively to Discourse-based Communication Models.

2.1 Discourse-based Communication Models

Discourse-based Communication Models specify high-level communicative interaction of the user with the application, primarily based on discourses in the sense of dialogues. The interacting agents (User and System) are depicted in Figure 1, which are the same for all presented Discourse-based Communication Models in this paper. A small excerpt of such a Communication Model of a simple flight booking application is shown in Figure 2. This excerpt models the interaction between the user and the system for selecting an outgoing flight and a return flight.

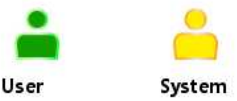

Figure 1: Interaction Parties

The basic building-blocks of Discourse-based Communication Models are Communicative Acts, depicted as rounded rectangles in Figure 2. Each Communicative Act is assigned to an agent, represented through its fill color (green/dark for User, and yellow/light for System). *Adjacency Pairs* model typical turn-takings in a conversation (e.g., Question–Answer or Request–Accept/Reject). Such Adjacency Pairs are represented through dia-

[1] http://ucp.ict.tuwien.ac.at

monds as shown in Figure 2 and relate one opening and zero to two closing Communicative Acts. An Adjacency Pair like the ones in Figure 2 relates one Question with one Answer. An opening Request Communicative Act, e.g., is related to two closing Communicative Acts, Accept and Reject, which are alternatives. An Informing Communicative Act, however, may also have no closing one in an Adjacency Pair, when no response is foreseen in the model.

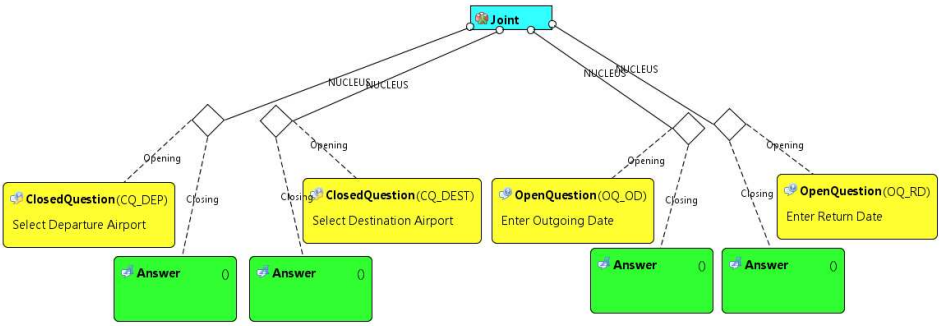

Figure 2: Discourse Model with Joint Relation

Additionally, Discourse Relations, like the *Joint* relation, can be used to link such Adjacency Pairs and to model more complex discourse structures. The Joint relation, as depicted in Figure 2 links two or more Adjacency Pairs and specifies that all Adjacency Pairs may be executed concurrently. Another Discourse Relation is the so-called *Elaboration*, based on the Elaboration relation defined in the Rhetorical Structure Theory (RST) [MT88]. According to this theory, all such relations have a main part called *Nucleus*, some have additionally a "helper" part called *Satellite*. In addition to relations based on RST, there are Procedural Constructs such as *Sequence* and *IfUntil* [PFA+09]. Such Procedural Constructs allow for modeling a more complex interaction that is not supported by RST (e.g., conditional loops).

A Discourse-based Communication Model refers to a Domain-of-Discourse (DoD) Model, which specifies the concepts that the two interacting agents can "talk about". A Domain-of-Discourse Model can be specified by an Ecore or UML class diagram. For the Flight Booking Excerpt it needs to specify an `Airport` and a `Flight` concept (shown in Figure 3). The connection between the Discourse and the Domain-of-Discourse Model is established through the propositional content specified for a Communicative Act (for details see [PR11]).

Figure 3: Excerpt of Flight Booking Domain-of-Discourse Model

Figure 2 uses a single Joint relation to connect all the Adjacency Pairs for inquiring all flight information. This is the straight-forward way for modeling such an interaction, because there is no dependency between the different information parts.

Such models can be automatically transformed into GUIs through the UCP:UI Framework [PRK13].

2.2 Design Alternatives in other GUI Generation Approaches

The most established way to model high-level interaction for model-based UI generation are *task models* [MPV11]. Such models, just like Discourse-based Communication Models, are situated on the Tasks & Concepts Level of the Cameleon Reference Framework [CCT+03]. A widely used notation for such task models are ConcurTaskTrees (CTT) [PMM97]. They can be transformed to multi-modal service front-ends with the MARIA Environment [PSS09]. UsiXML [Van08] is another framework that uses task models to specify high-level interaction for transformation to UIs. UseML models are task models that have been designed to model high-level interaction in the automation domain and can be transformed to GUIs automatically though the Model-based Useware Engineering approach [MBS11]. These three task modeling approaches form the basis for a W3C standardization effort currently under way.[2]

An approach that supports the model-driven development GUIs for enterprise applications is CEDAR Studio [ABY13]. This approach relies on task models in CTT notation and UsiXML compliant Abstract UI (AUI) and Concrete UI (CUI) models and thus supports the exploration of design-alternatives on these three levels.

UsiComp is another approach that relies on the UsiXML meta-models [GFCDC+12]. UsiComp supports automated GUI generation at design-time and run-time. Run-time generation facilitates the exploration of design alternatives as the resulting UI is immediately available.

Run-time GUI generation for information systems is presented in [PEPA08]. The OO-Method uses a conceptual model that consists of 4 sub-models as input and compiles them into a running application. It also relies on task models on the highest level of abstraction.

A UML based approach for interaction modeling is presented in [HSW11]. This approach uses UML class diagrams and statemachines on the Tasks&Concepts Level and supports the transformation of these models to GUIs.

Model-driven development and evolution of customized UIs that facilitates the creation of UI families is presented in [PWB13]. In contrast to the approaches presented above, this approach applies design alternatives on AUI Level to allow for the automated derivation of product families.

Any of these approaches can be used to model design alternatives, of course. We have not yet seen any systematic investigation of this important topic, however.

[2] http://www.w3.org/2011/01/mbui-wg-charter.html

3 Concurrent Discourse Design

A concurrent discourse in our context is one that allows, in principle, concurrent execution of all its parts, which may be rendered on a GUI in such a way that all these parts are reflected on a single screen. In fact, the model in Figure 2 above specifies such a concurrent discourse design. If all its discourse parts can actually be rendered on a single GUI screen, no particular order is pre-determined for the user to provide all this flight information. Note, however, that it also does not specify any order for placing the related widgets on the GUI screen.

Information that is concurrently available is typically not completely independent as modeled through the Joint relation in Figure 2. For example, it would be counter-intuitive or at least unusual for users to find the question for the destination airport above the one for the departure airport.

Such dependencies can be captured in high-level interaction models through structuring such information hierarchically, even though there are no strong dependencies. Figure 4 shows how the information of our running example can be structured using our modeling approach through additional grouping of the parts. In addition, this Discourse Model uses the *OrderedJoint* relation instead of the Joint relation. The procedural semantics of both relations are identical, but the OrderedJoint allows defining a weak ordering between the parts. This additional information is beneficial for creating the layout of a GUI, or for determining the sequence of presenting the parts on several screens not large enough for the whole structure. This sequence can also be used, e.g., for rendering a speech UI. In summary, this concurrent discourse design differs from the straight-forward one shown in Figure 2 in two ways:

- Additional information is provided through grouping the information hierarchically.
- Additional information is provided through ordering the parts (Nuclei).

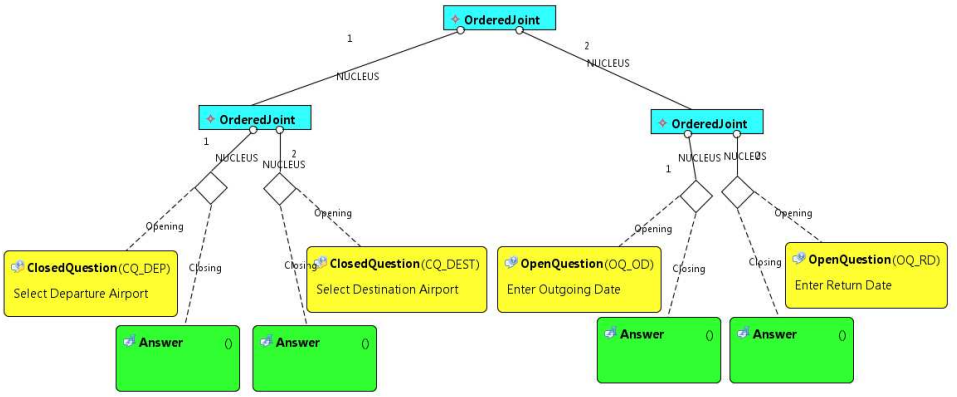

Figure 4: Concurrent Discourse Design with OrderedJoint

Figure 5 sketches the structure of a potential GUI for the OrderedJoint Discourse Model. The information is grouped according to the hierarchical structure of the Discourse in two sub-panels (i.e., Airport Selection and Date Selection). The layout of the container panel (Flight Booking – Ordered Joint) has been created based on the ordering of the Nuclei of the top-most OrderedJoint Relation. Thus the Airport Selection panel is placed above the Date Selection panel.

Figure 5: Sketch of GUI Structure for Concurrent Discourse Design with OrderedJoint

Concurrent discourse design allows capturing weak dependencies between parts that do not have strong dependencies requiring to render different screens for them. It is particularly beneficial for multi-device GUI generation, as it facilitates placing these parts together on large screens as well as assigning them to several small screens.

4 Sequential Discourse Design

A sequential discourse in our context is one that defines strictly sequential execution of all its parts in separate steps, which may be rendered in a GUI as one screen for each step. The model in Figure 6 shows such a design for our running example. Instead of the Ordered-Joint relation of the concurrent design presented above, at the top of the model diagram shown in Figure 6 is a *Sequence* Procedural Construct. It explicitly specifies the sequence in which its Nuclei are to be executed through numbering them. This Discourse Model specifies that the information about departure and destination airport is to be exchanged before the information of the flight dates.

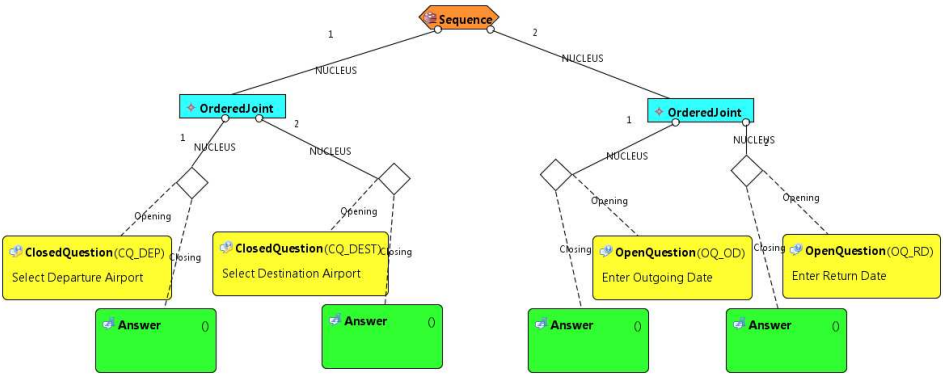

Figure 6: Sequential Discourse Design

This interaction design is strictly linear and thus results in two screens as illustrated in Figure 7, instead of (potentially) one shown in Figure 5.

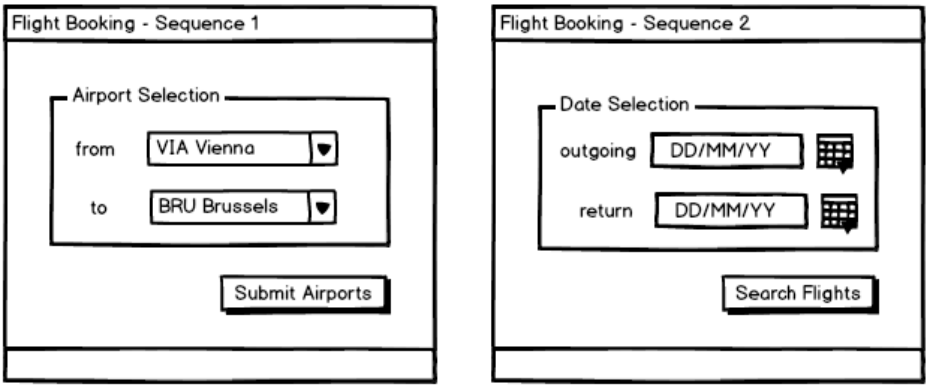

Figure 7: Sketch of GUI Structure for Sequential Discourse Design

Modeling the flow of interaction in well-defined steps is beneficial in the context of process-oriented applications (e.g., booking applications) as it facilitates user orientation. An additional benefit is that information from previous screens can already be exploited to refine the information on subsequent screens. For example, the dates that are offered for selection in our running example could be adjusted according to availability of flight connections for the given airports.

5 Incremental Discourse Design

An incremental discourse in our context is one that allows for incremental execution of its parts in two (or more) steps. In the basic case, the first part is initially executed and triggers the subsequent execution of the second part, like the sequential design presented above. However, the discourse part of the second step is an increment of the one in the first in the sense, that the first discourse part can be repeated, in contrast to the sequential design. In a related GUI, there a two resulting screens on a GUI, one that renders the first part and a subsequent one that renders the first *and* the second part.

Let us illustrate also this design alternative with our running example. The Discourse Model shown in Figure 8 uses an *Elaboration* relation instead of OrderedJoint or Sequence in the previous design alternatives. An Elaboration specifies exactly one Nucleus and one Satellite branch and is assigned to one of the two interacting parties. This assignment is obligatory as the Satellite branch additionally specifies a condition that has to be evaluated by this party. The Satellite branch is executed in addition to the Nucleus branch if this condition is true. More precisely, the Elaboration in Figure 8 is assigned to the System, illustrated through its yellow/light fill color. Therefore, the condition of the Satellite branch (i.e., *departureAirport != null && outgoingDate != null*) is evaluated by the System.

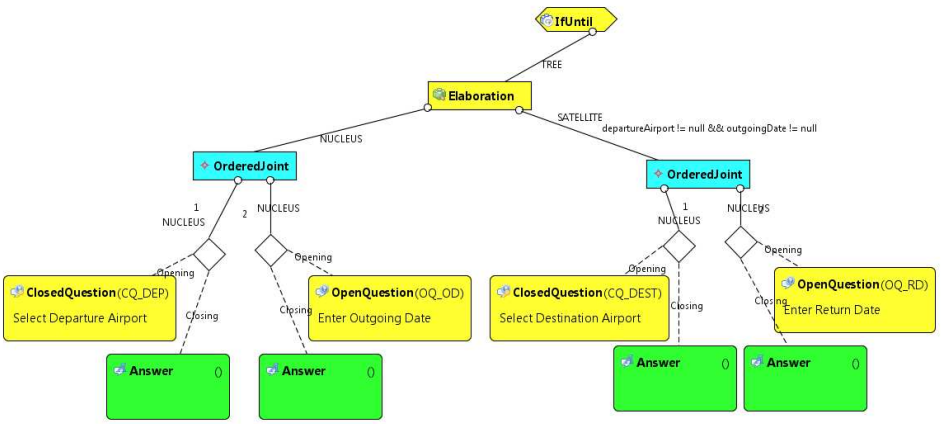

Figure 8: Incremental Discourse Design

Modeling the interaction in such an incremental way allows for additional interplay with the business logic for collecting information in the Nucleus branch that can be used to calculate the information provided in the Satellite branch. For this reason, we modeled the flow of interaction in the incremental discourse model shown in Figure 8 slightly differently than in the concurrent and the sequential design. In particular, we grouped the ClosedQuestion for the departure airport together with the OpenQuestion for the departure date in the Nucleus branch of the Elaboration, and the ClosedQuestion for the destination airport together with the OpenQuestion for the return date in its Satellite branch. This allows the System to calculate all possible destination airports for a given departure airport

and date, which can already be presented in the second screen.

At the top of the diagram shown in Figure 8 is an *IfUntil* Procedural Construct, which is also assigned to the System. It is needed to trigger the evaluation of the Satellite condition after the Nucleus branch has been completed.

The sketch of a GUI structure rendered for this incremental discourse design is illustrated in Figure 9 and contains two screens. The screen on the left renders the Nucleus branch collecting the departure information only. The screen on the right renders the Nucleus and the Satellite branches together (i.e., departure and destination information). Note, that the interaction of the first step of the incremental design is still possible in this second screen, in contrast to the sequential design.

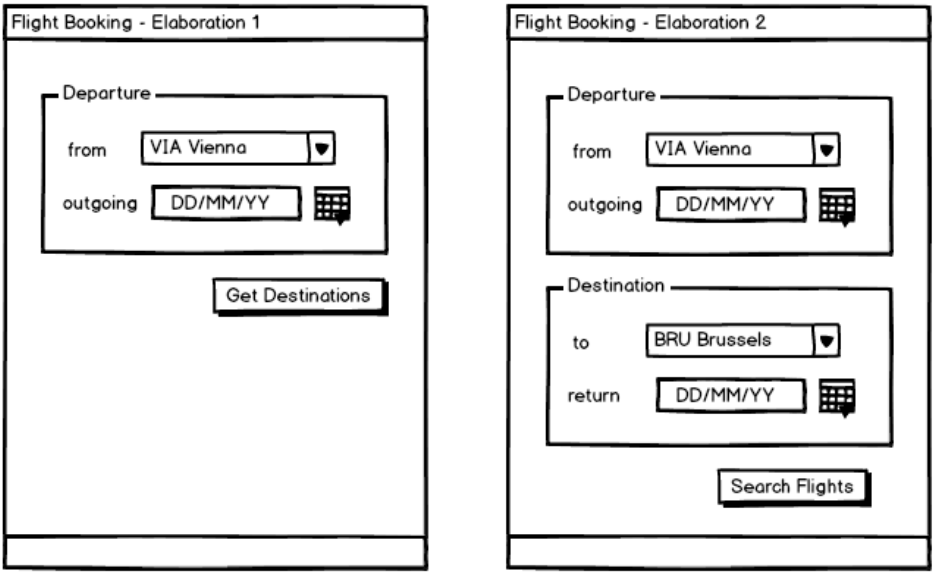

Figure 9: Sketch of GUI Structure for Incremental Discourse Design

So, an Elaboration allows updating the information provided in the satellite according to information collected in the nucleus, while the nucleus information is still available and can be modified. This additionally requires the application logic to evaluate the condition of the satellite branch, in contrast to the previously presented design alternatives. Therefore, it makes sense to include additional functionality of the application logic that exploits the information of the Nucleus branch to adjust the information in the satellite (e.g., calculating the destination airports for a given departure airport) in incremental discourse design. Note, that such additional functionality may also be included in sequential design, but not in an incremental manner. Thus, incremental discourse design is beneficial in a context where the user shall be given the opportunity to explore different alternatives through re-adjusting previously entered data (e.g., departure and destination airport) in well-defined steps.

Actually, incremental discourse design is not limited to *two* steps. Elaborations can be cascaded to model incremental design in more than two steps.

6 Discussion and Future Work

This paper presents three discourse design alternatives together with possible sketches of corresponding GUI structures. However, model-driven GUI development typically involves some model of the domain as well, in our approach a DoD Model. There is actually a strong dependency between these models.

In particular, all design alternatives presented above use the same DoD Model (shown in Figure 3). Figure 10 shows an alternative DoD model for our running example that models all relevant data as attributes of a single class.

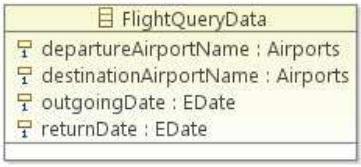

Figure 10: Excerpt of Flight Booking DoD with Composite FlightQueryData

The interaction required to exchange this data can be modeled with one single Adjacency Pair as depicted in Figure 11.

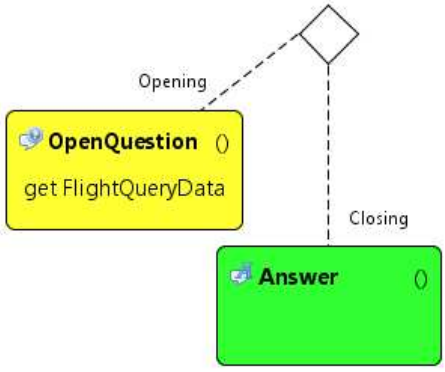

Figure 11: Discourse Model Corresponding to Composite DoD

So, the granularity of the DoD model is strongly related to the complexity of the Discourse Model and vice versa. The composite DoD Model, for example, does not fit to the discourse alternatives presented above, but it reduces the complexity of the Discourse Model. However, this complexity reduction constrains the flexibility in modeling the interaction.

The sketch of a corresponding GUI structure is presented in Figure 12 and shows that all widgets are part of the same panel. The transformation Framework creates such a GUI through the application of one single transformation rule that transforms an OpenQuestion–Answer Adjacency Pair. This increases the predictability for the designer what the resulting GUI will look like as no automated generation of layout is required. However, such a model does not allow for splitting the information on different screens as required for tailoring a GUI to a small device, for example. Thus, the interplay between Discourse and DoD Model can be viewed as a trade-off between predictability of the resulting GUI and flexibility in terms of GUI tailoring. Such a composite design is well applicable in a context where predictability of the resulting GUI is important, because it does not allow for splitting the information and it facilitates GUI customization as only one single customized rule would be required.

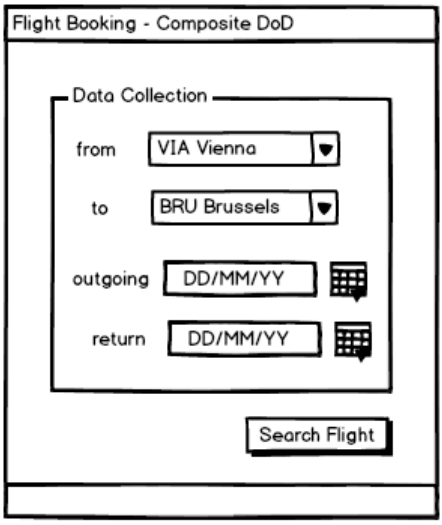

Figure 12: Sketch of GUI Structure for the Composite DoD

We conjecture that such design alternatives also exist between other models that are used to capture the different aspects involved in UI development (e.g., device, user or environment models).

In future work, we plan to create interaction *patterns* and to develop solutions based on our Communication Models on two different levels of abstraction. In particular we plan to develop:

- *Discourse Model patterns* that define interaction design independently of a specific application domain, based on the design alternatives presented above; and
- *Communication Model patterns* that provide a concrete solution for a specific scenario (e.g., authentication) with a fixed part of a DoD Model.

7 Conclusion

This paper presents alternatives for communicative interaction design on a high level of abstraction, and sketches of corresponding GUI structures. According to our best knowledge, this is new in the field of automated GUI generation, while it is well known that design usually involves many alternatives. Of course, for any such interaction design many GUIs rendering it are possible as well, but these alternatives are out of the scope of this paper.

The presented design alternatives have different properties, which we discuss in the spirit of design rationale. An especially interesting property is, whether such a design alternative allows for additional interplay with the application logic, which may provide further information for subsequent screens of the rendered design.

Acknowledgments

Part of this research has been carried out in the GENUINE project (No. 830831) funded by the Austrian FFG.

References

[ABY13] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. Cedar Studio: An IDE Supporting Adaptive Model-Driven User Interfaces for Enterprise Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive computing systems*, EICS '13, New York, NY, USA, 2013. ACM.

[CCT+03] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.

[FKPR11] Jürgen Falb, Hermann Kaindl, Roman Popp, and David Raneburger. Automated WIMP-UI Generation Based on Communication Models. *i-com*, 10(3):48–55, November 2011.

[GFCDC+12] Alfonso García Frey, Eric Céret, Sophie Dupuy-Chessa, Gaëlle Calvary, and Yoann Gabillon. UsiComp: an extensible model-driven composer. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '12, pages 263–268, New York, NY, USA, 2012. ACM.

[HSW11] Frank Honold, Felix Schüssel, and Michael Weber. A UML-Based Approach for Model Driven GUI Generation. *i-com*, 10(3):26–32, November 2011.

[MBS11] Gerrit Meixner, Kai Breiner, and Marc Seissler. *Model-Driven Useware Engineering*, chapter 1, pages 1–26. Studies in Computational Intelligence, SCI. Springer, Heidelberg, 1 2011.

[MPV11] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. Past, Present, and Future of Model-Based User Interface Development. *i-com*, 10(3):2–10, November 2011.

[MT88] W. C. Mann and S.A. Thompson. Rhetorical Structure Theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

[PEPA08] Oscar Pastor, Sergio España, José Ignacio Panach, and Nathalie Aquino. Model-Driven Development. *Informatik Spektrum*, 31(5):394–407, 2008.

[PFA+09] Roman Popp, Jürgen Falb, Edin Arnautovic, Hermann Kaindl, Sevan Kavaldjian, Dominik Ertl, Helmut Horacek, and Cristian Bogdan. Automatic Generation of the Behavior of a User Interface from a High-level Discourse Model. In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences (HICSS-42)*, Piscataway, NJ, USA, 2009. IEEE Computer Society Press.

[PMM97] Fabio Paternò, Cristian Mancini, and Silvia Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the IFIP TC13 Sixth International Conference on Human-Computer Interaction*, pages 362–369, 1997.

[PR11] Roman Popp and David Raneburger. A High-Level Agent Interaction Protocol Based on a Communication Ontology. In Christian Huemer, Thomas Setzer, Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, and Clemens Szyperski, editors, *E-Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, pages 233–245. Springer Berlin Heidelberg, 2011. 10.1007/978-3-642-23014-1_20.

[PRK13] Roman Popp, David Raneburger, and Hermann Kaindl. Tool Support for Automated Multi-device GUI Generation from Discourse-based Communication Models. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive computing systems*, EICS '13, New York, NY, USA, 2013. ACM.

[PSS09] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. 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, November 2009.

[PWB13] Andreas Pleuss, Stefan Wollny, and Goetz Botterweck. Model-driven Development and Evolution of Customized User Interfaces. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive computing systems*, EICS '13, New York, NY, USA, 2013. ACM.

[RPK+11] David Raneburger, Roman Popp, Sevan Kavaldjian, Hermann Kaindl, and Jürgen Falb. Optimized GUI Generation for Small Screens. In Heinrich Hussmann, Gerrit Meixner, and Detlef Zuehlke, editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 107–122. Springer Berlin / Heidelberg, 2011.

[RPK13] David Raneburger, Roman Popp, and Hermann Kaindl. Model-driven Transformation for Optimizing PSMs: A Case Study of Rule Design for Multi-device GUI Generation. In *Proceedings of the 8th International Joint Conference on Software Technologies (ICSOFT'13)*. SciTePress, July 2013.

[Van08] Jean M. Vanderdonckt. 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, Sept. 2008.