An Iterative and Incremental Process for Interaction Design through Automated GUI Generation

David Raneburger, Roman Popp, Hermann Kaindl,
Alexander Armbruster, and Vedran Šajatović

Institute of Computer Technology, Vienna University of Technology
Gusshausstrasse 27-29, 1040 Vienna, Austria
{raneburger,popp,kaindl,armbruster,sajatovic}@ict.tuwien.ac.at

Abstract. Model-driven generation of graphical user interfaces (GUIs) for multiple devices requires a model representing an interaction design. High-quality interaction models are a prerequisite for achieving a good level of usability for the corresponding applications. Our tool-supported process facilitates the exploration and evaluation of interaction design alternatives in an iterative and incremental manner, using automated GUI generation to achieve a running application more quickly and with reduced effort in comparison to manual (prototype) development. This allows the designer to quickly find a suitable alternative and to build more complex applications incrementally.

Keywords: Interaction design, automated GUI generation, iterative and incremental process.

1 Introduction

According to Preece et al. [1], creating an interaction design typically involves designing alternatives, prototyping and evaluating, and these activities are to be repeated for informing each other. We recently presented an iterative process for facilitating interaction design through automated GUI generation in [2]. While Preece et al. have strong emphasis on iterations, they do not even mention incremental development, which is widely used in software development, however (see, e.g., [3]).

Our objective in the current paper is to facilitate interaction design in a defined iterative and incremental process, utilizing automated GUI generation tool-support for fast, inexpensive, and still high-fidelity prototyping. So, we present a defined process for this kind of development based on automated GUI generation, and experience from its application during the interaction design of a larger trial application. This process is iterative and incremental and, in principle, facilitates both improvements and extensions to the interaction design in each iteration. It starts with the creation of an initial design using certain heuristics. This initial interaction design is *not* supposed to be complete in our proposed

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2014, LNCS 8510, pp. 373–384, 2014.

process in the sense of covering all the interaction possibilities yet. Especially for relatively large applications, providing an already complete initial model of the interaction design is too difficult. So, like in usual approaches to software development in general, it is preferable to start with an essential part first, to get cyclic feed-back to the current version, to adapt accordingly, and to add an increment in a defined iteration.

The remainder of this paper is organized in the following manner. First, we relate this new process to existing work and provide some background material in order to make our paper self-contained. Then we present this tool-supported process for iterative and incremental interaction design in detail. After that, we describe an informal evaluation in the course of developing a trial application. Finally, we discuss our approach more generally and conclude.

2 Related Work and Background

A thorough discussion of related work on academic and industrial state-of-the-art design-time GUI generation approaches can already be found in [2]. So, we restrict our discussion on major work on life-cycle models for iterative and incremental development. Subsequently, we provide background information on the design-time GUI generation approach that we used to evaluate our new process, the Unified Communication Platform.[1]

2.1 Related Work

Software development has generally changed from a 'waterfall' approach to an iterative and incremental development (IID) approach. The basic approach is much older than usually acknowledged, see, e.g., [4]. However, IID became much more wide-spread in use with the Rational Unified Process (RUP) [5], or 'agile' approaches such as Extreme Programming, see, e.g., [6], where the most popular one today is *Scrum* [7].

Recently, more and more literature suggests to use some kind of iterative and incremental process also for the development of general systems. In particular, RUP has been extended for the engineering of general "large-scale systems composed of software, hardware, workers and information components" [8], and is supported in a process management platform as a plug-in. One of the authors of the current paper proposed a life-cycle model for iterative and incremental systems engineering in [9].

2.2 Background on the Unified Communication Platform

The Unified Communication Platform (UCP) uses a Discourse-based Communication Model [10] to model the communicative interaction between two parties, which can be assigned to the Tasks & Concepts Level of the CRF [11]. In the

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

context of UI Generation, one communication party is the human user interacting with the application via a UI. A Discourse-based Communication Model specifies interaction based on discourses in the sense of dialogues. It consists of a Discourse Model, which models all the possible dialogues, a so-called Domain-of-Discourse (DoD) Model, which models the concepts that can be exchanged between the user and the application, and an Action-Notification Model (ANM), which models Actions and Notifications that can be performed by one of the communicating parties (either the user or the application). UCP already provides a so-called 'basic ANM' that specifies basic Actions like `get`, `set`, or `selecting`, and basic Notifications like `presenting`.

The basic interaction units in such Discourse Models are so-called Adjacency Pairs, like question–answer or offer–accept/reject. Each such Adjacency Pair consists of one or two Communicative Acts (e.g., a question and an answer). These units are connected via so-called Discourse Relations, such as *Alternative*, *Title*, *Switch* or *IfUntil*, in a hierarchical way. An *Alternative* relation specifies that the connected sub-branches can, in principle, be executed concurrently, but that only one sub-branch can be finished, in contrast to a *Switch* relation, where each sub-branch specifies a condition and only one sub-branch can be executed (which sub-branch is executed is determined through evaluating the corresponding condition). A more complex relation is *IfUntil*, which is a combination of a loop with a condition. A specified sub-branch is repeated until a condition is fulfilled. After the condition is fulfilled, another sub-branch is executed. All three relations are assigned to one of the interacting parties, which evaluates the corresponding conditions. A *Title* relation, in contrast, specifies that its Nucleus and its Satellite branch can be executed concurrently without any conditions, and does not depend on a specific interaction party. For more details see [10].

UCP supports modeling such Discourse-based Communication Models with a dedicated editor [12]. Most importantly, UCP can fully-automatically generate GUIs from such models [13]. UCP also provides a message-based run-time environment [13], which allows deploying the generated GUIs and application backends. This run-time environment uses the Communicative Acts as messages and provides a generic function-based interface for the back-end integration. The functions are implicitly defined in the Discourse-based Communication Model through the *propositional content* of the Communicative Acts and the interface can be generated automatically. UCP uses a run-time architecture that is based on the Model-View-Controller pattern [14], but does not impose any constraints on the technology to be used to implement the functionality.

3 Our Tool-supported Process for Iterative and Incremental Interaction Design

Larger and more complex applications typically involve more complex interaction. Such applications are typically developed in an iterative and incremental manner. Therefore, we extended our process for iterative interaction design presented in [2] to support incremental development as well. Figure 1 shows

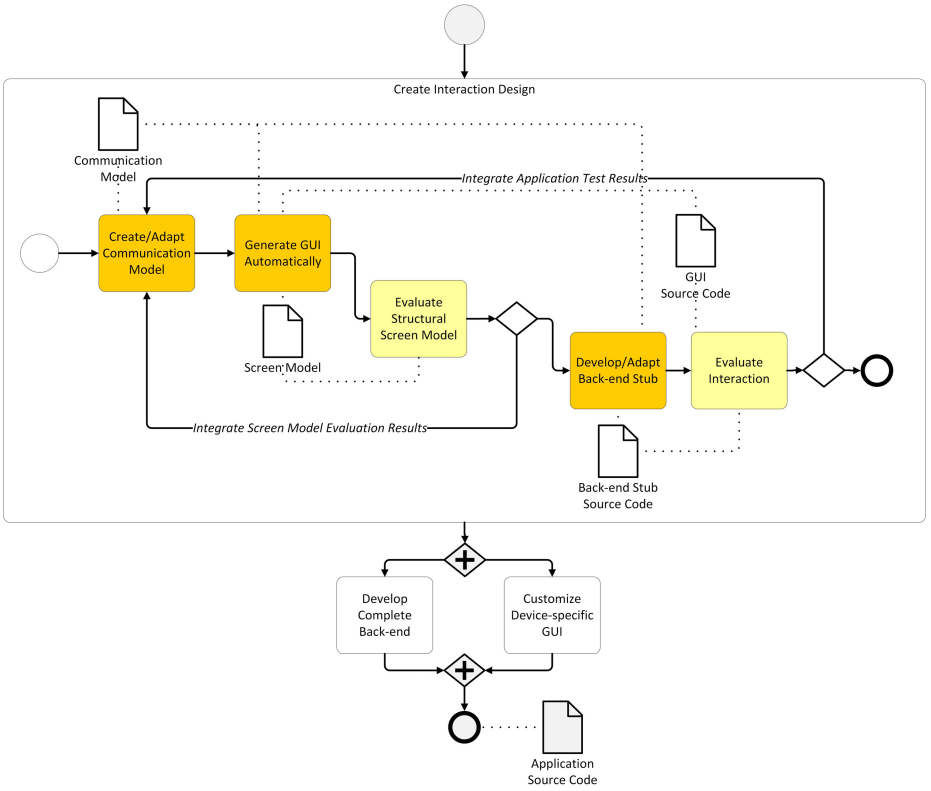


Fig. 1. Our Tool-supported Process for Iterative and Incremental Interaction Design Embedded in our Communication Model-based Application Development Process

our extended Communication Model-based application development process in BPMN.[2] Analogously to our iterative development process, it includes and starts with the tool-supported process named *Create Interaction Design*. Once a stable interaction design is available, the activities *Develop Complete Back-end* and *Customize Device-specific GUI* can again be performed concurrently. These activities focus on GUI customization and manual back-end development and are out of the scope for this paper.

In Figure 1, all activities shown in orange (dark) result in new or customized artifacts, while all activities shown in yellow (light) are evaluation activities. Our iterative and incremental process *Create Interaction Design* starts with the *Create/Adapt Communication Model* activity. This activity creates the initial `Communication Model` and adapts it in subsequent iterations. Such adaptations can either be modifications due to the results of an evaluation activity, or incremental extensions, or both. Incremental extensions of the interaction design

[2] `http://www.omg.org/spec/BPMN/2.0`

may concern any of the three models that constitute the Communication Model (i.e., the Domain-of-Discourse, the Action-Notification or the Discourse Model).

Once a `Communication Model` is available, it can be transformed in the activity *Generate GUI Automatically* to a `Screen Model` and the `GUI Source Code` fully automatically using UCP. The Structural Screen Model can be used for a first evaluation in the *Evaluate Structural Screen Model* activity. These evaluation results can be immediately used to customize the `Communication Model` by following the path labeled *Integrate Screen Model Evaluation Results* in Figure 1. This allows iterating on design alternatives in micro-iterations, much as in our previous purely iterative process [2].

Once the results from the evaluation of the screen model are satisfactory, our process continues with the *Develop/Adapt Back-end Stub* activity. Developing a back-end stub is the same activity as in our purely iterative process and means its initial creation. Adapting the back-end stub can be modifications due to the results of an evaluation activity, or incremental extensions, or both. Incremental extensions implement the new functionality introduced through an extension of the Communication Model. This activity results in a prototypical application back-end stub, as required to achieve a running application prototype. It can be used in the *Evaluate Interaction* activity, where the focus is on its external behavior in the course of interacting with a user. The results of this evaluation can be used to customize the `Communication Model`. In effect, this allows iterating on design alternatives in macro-iterations.

Overall, this process is similar to the iterative design process presented in [2], but additionally includes incremental extensions of various artifacts. To make this paper self-contained, we include as short description of the *Generate GUI Automatically*, the *Evaluate Structural Screen Model* and the *Evaluate Interaction* activities here, based on [2].

The *Generate GUI Automatically* activity is essentially a machine task performed fully-automatically by UCP as presented in [13]. It generates the `Screen Model` (as an intermediate result) and the `GUI Source Code`. The Screen Model specifies the GUI's behavior through a Behavioral Screen Model and the GUI's structure through the Structural Screen Model. The Structural Screen Model is a screen-based graphical representation on Concrete UI Level [11] comparable to GUI mock-ups.

The *Evaluate Structural Screen Model* activity evaluates the screens of the Structural Screen Model. It allows identifying missing interaction elements and thus shortcomings of the interaction model through comparing the generated GUI model to the functional requirements (e.g., given through use cases) that the resulting application needs to satisfy. Importantly, this is feasible even without the availability of an application back-end (prototype or final). Therefore, it allows for micro-iterations in our process. UCP provides a dedicated graphical editor for the Structural Screen Model, which supports this evaluation activity through visualization of the generated screens.

The *Evaluate Interaction* activity uses the running application prototype to evaluate the interaction. This may be done informally by the developer, again

through comparing the interaction to the functional requirements for the application, but is typically achieved through *heuristic evaluation* performed by usability experts, or possibly even through *usability tests*, but always with a focus on the interaction rather than the GUI screens per se. Only problems or violations of heuristics concerning the external behavior are relevant, because layout and style are not reflected in the interaction design.

4 Evaluation Using Vacation Planning

We built a vacation planning application according to our process, in effect evaluating it. In particular, we use this trial application in this paper to illustrate all activities of our process and present how we developed the corresponding interaction design incrementally in several micro- and macro-iteration, using UCP tool support.

Our vacation planning application is based on a commercial accommodation booking Web-site of an Austrian province and implements a subset of this Web-site's functionality. In particular, our application supports searching for an accommodation either through text search or through more specific search masks. In addition, it provides information on events, articles that report on different topics, and information on how to get to this Austrian province. Finally, it allows a user to send a booking request to a specific accommodation or to book an accommodation directly. Below we use excerpts of the corresponding Communication Model to illustrate the enactment of our new process.

4.1 Initial Iteration

The basis for our interaction development was a set of tasks that should be supported through the vacation planning application to build, and an existing commercial Web-site that supported them. Such tasks were, for example, get information on events, book a specific accommodation or find out how to get there. These tasks were already supported by the commercial Web-site, and we basically re-engineered its interaction. Our aim was to allow for a comparative usability evaluation of our application through a user study in the end. This user study is, however, out of scope for this paper.

The interaction required to support the tasks could be triggered on the commercial Web-site through links labeled *Home*, *Search Accomodation*, *Plan Vacation*, *Get There* and *Events*. The Web-site allowed for switching between these categories at any time, which decouples the interaction attached to each category and facilitates incremental development of the interaction model.

The *Home* link, for example, led to the start-page that displayed a welcome message and presented a list of events and a list of articles. Selecting a specific event or article led to a Web-page with more detailed information on the selected event/article.

When developing our interaction design, we started with modeling these alternative selections and the interaction provided by the start-page. Figure 2

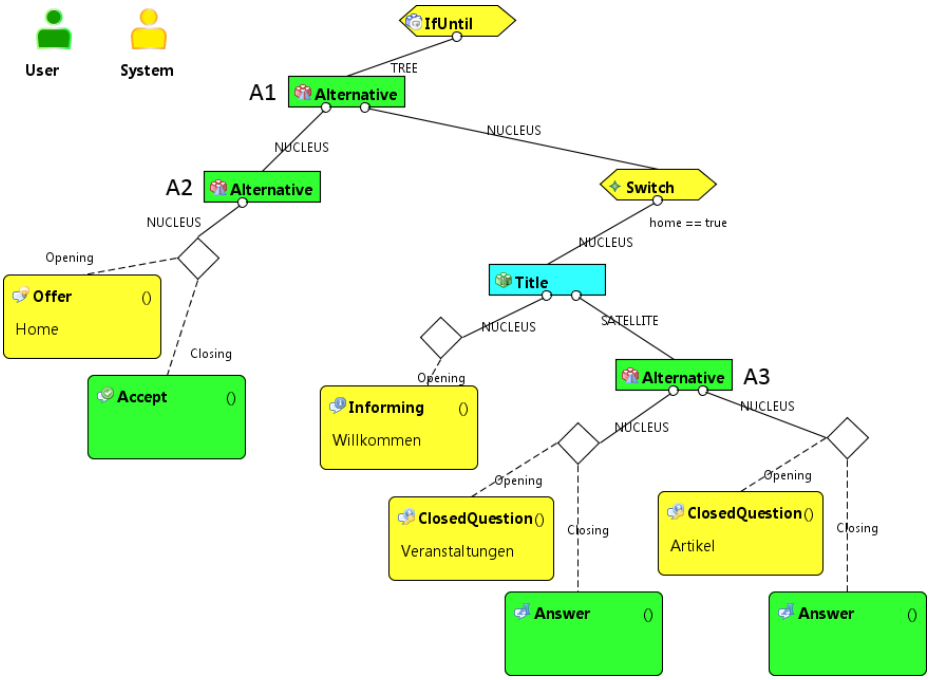


Fig. 2. Excerpt of Initial Vacation Planning Discourse Model

shows an excerpt of the corresponding Discourse Model. This model has German content descriptions for the Communicative Acts, because the language of the application was German and these descriptions are used as default labels by our GUI generation framework. We will use English translations in this paper, providing the original German words in italics next to the translation.

The interacting parties in our Communication Model are a User (green/dark fill-color) and the System (yellow/light fill-color), depicted in the upper left corner of Figure 2. The top-level `IfUntil` relation is assigned to the System (indicated through its yellow/light fill-color) and specifies a so-called `Tree` branch without a condition only and no `Then` or `Else` branch. This models an endless loop used for restarting the application (i.e., to display the home screen again) after the interaction has been finished.

The Alternative relation labeled A1 has been assigned to the User, which means that the User can alternatively perform any interaction specified in its `Nucleus` branches. Its left Nucleus contains another Alternative relation (A2), which we used to model the selection between the five links that should be available at any time for the user during the run-time of the application. We built our interaction model incrementally starting with the Home link, which is modeled through the `Offer-Accept` Adjacency Pair. The corresponding interaction is modeled in the second Nucleus branch of Alternative A1. This nucleus contains a `Switch` relation that has been assigned to the System. This means that the

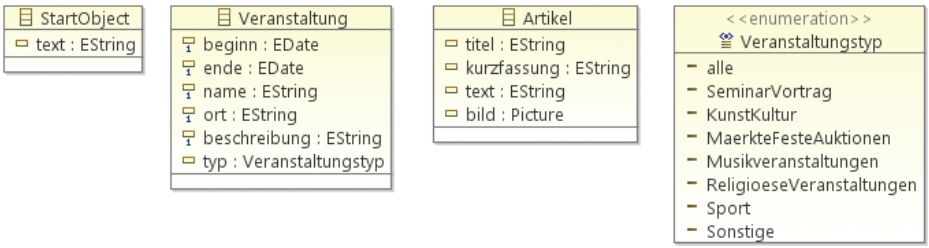


Fig. 3. Initial Vacation Planning DoD Model

condition assigned to the Switch's Nucleus branch (i.e., `home==true`) is evaluated by the System. This condition is true when the application is started and can be set to true at any time by the User through accepting the Home Offer (modeled through the corresponding Adjacency Pair as described above).

The interaction of the start-page has been modeled in the Nucleus branch of the Switch relation with the condition `home==true`. It contains a `Title` relation, whose Nucleus contains the `Informing` Communicative Act presenting the welcome message, and whose Satellite contains another Alternative relation (A3). This Alternative relation links two `ClosedQuestion-Answer` Adjacency Pairs, one for the events (*Veranstaltungen*) and one for the articles (*Artikel*).

In addition to the Discourse Model, we also created the initial Domain-of-Discourse and the Action-Notification Model in an iterative way. The initial Domain-of-Discourse Model is shown in Figure 3. It defines the classes `StartObject`, `Event` (*Veranstaltung*) and `Article` (*Artikel*) with their respective Attributes. Moreover, it defines `EventType` (*Veranstaltungstyp*) enumeration, which is used by the `type` (*typ*) attribute of the event class.

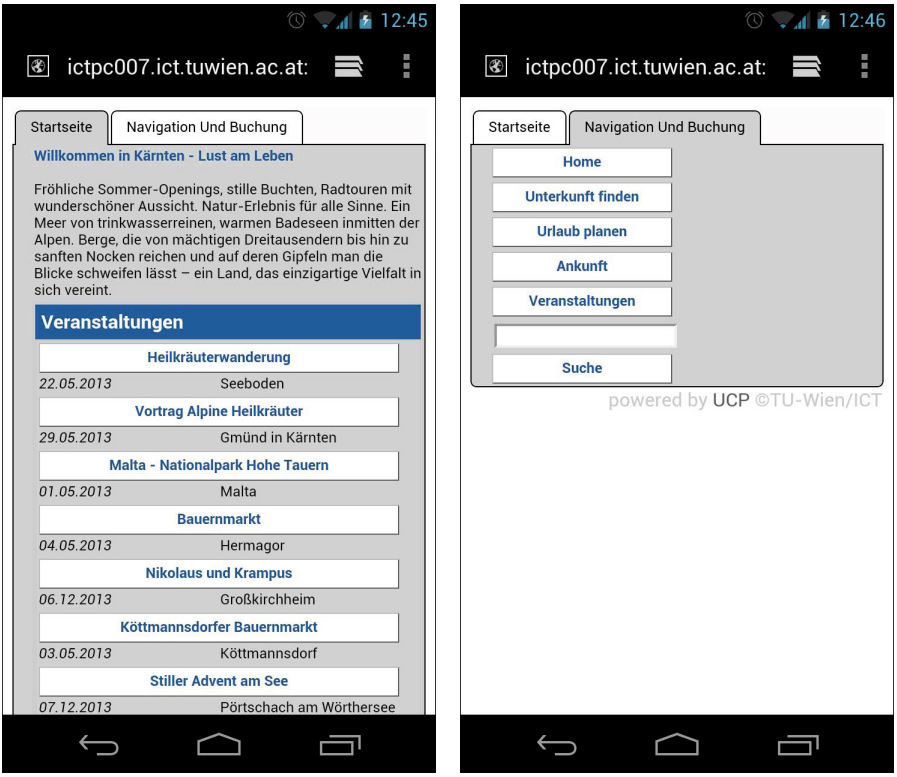
The initial ANM model contained a `home` Action only, as the remaining actions used in our initial Communication Model were already specified in the basic ANM model. Both, the concepts specified in the DoD Model and in the ANM were referenced through the propositional content of the Communicative Acts in our Communication Model.

After having created these three models, we generated the corresponding GUI, evaluated the Screen Model and implemented the back-end stub. Then we iteratively refined the interaction.

4.2 Generated GUIs and Incremental Extensions

The generated GUI of the start-page of our application is shown in Figure 4 on a smartphone device. Our generation framework split the Alternative relation A1, rendering it as a tabbed pane, to avoid horizontal scrolling, and to keep the vertical scroll-limit of five times the screen length (as defined through the smartphone platform model). Figure 4(a) shows the GUI for the Home branch of the Switch relation (as sketched in Figure 2).

We modeled the interaction following the selection of a specific event or article on the start-page in our first incremental iteration. This interaction intuitively



(a) Vacation Planning Smartphone Home Screen.

(b) Vacation Planning Smartphone Menu Screen.

Fig. 4. Vacation Planning GUI Displayed on a Samsung Galaxy Nexus Device

belongs to the home Sub-Discourse presented above, but we modeled it in a new branch of the Switch relation instead, as it should be reachable from different points of the final interaction model, and subsequently again evaluated and refined this new branch iteratively. Another example for interaction that should be reachable from different points is the payment interaction that finalizes the booking process for a specific accommodation, which we added in another incremental iteration. In general, we modeled all interactions that should be reachable from different points in the interaction model as sub-branches of the Switch relation, extending the existing interaction model and developing the final interaction model in increments.

Figure 5 illustrates the structure of the final Discourse Model. This model offers six options in the navigation (i.e., Alternative) branch of Alternative relation A1. Five of these options were Offer-Accept Adjacency Pairs, which are sketched in Figure 5 through the Home and the Events (*Veranstaltungen*) Adjacency Pairs with dots in between. The sixth branch contains the text search (*VeranstaltungenSuche*), modeled as OpenQuestion-Answer Adjacency Pair.

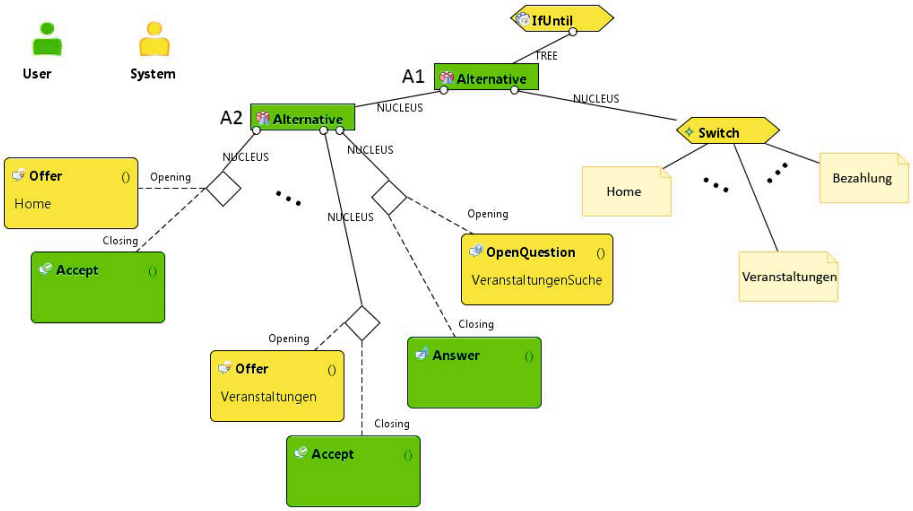


Fig. 5. Excerpt of Final Vacation Planning Discourse Model

Each Offer-Accept Adjacency Pair triggers the corresponding interaction, modeled as a branch of the Switch relation. The Switch branch of Alternative A1 finally contains eleven branches due to the additional branches for interaction that was reachable from different points of the final interaction model. The remaining Switch branches are sketched through notes in Figure 5.

The five branches corresponding to the Offer-Accept Adjacency Pairs are sketched through the Home and the Events (*Veranstaltungen*) notes with dots in between. The remaining six branches, that model interaction that is reachable from different points in the final interaction model, are sketched through the dots between the Events (*Veranstaltungen*) and the Payment (*Bezahlung*) notes. Such interaction is the display of text search results, which typically contains a list of accommodations, events and articles. All items in these three lists can be selected to provide further details, which triggers the interaction in the corresponding branch of the Switch relation. We provided three branches for this interaction, one for Accommodation details, which also allowed for booking a specific accommodation, one for Event and one for Article details. The remaining two branches model the interaction for getting information on how to get there by a specific means of transport (i.e., plane, train or car) and the interaction required for completing the booking process for a specific accommodation through payment.

Overall, we performed 11 incremental iterations while developing the Communication Model. After each increment, we evaluated and refined the Communication Model iteratively. Extending and refining the Discourse Model also included extensions and modifications of the DoD Model and the ANM. The final DoD Model specifies 18 classes and 8 enumerations, and the final ANM specifies 11 Actions and 2 Notifications.

Figure 4(b) shows the GUI for the final Alternative branch A2, as sketched in Figure 5. The complete GUIs of our vacation planning application are accessible in the Web for Smartphone[3] and Desktop[4].

5 Discussion

As an empirical evaluation, we presented the application of our iterative and incremental interaction development process during the development of a more complex vacation planning application. We found, however, that even a high-quality interaction model requires further GUI customization to achieve a good level of usability and the desired "look & feel". We will, therefore, extend our iterative and incremental process to support GUI customization as well, concurrently to interaction design. This will facilitate the evaluation of the interaction design through heuristic evaluations and user studies, and become a synthesis of the approach in the current paper and our previous approach to perfect-fidelity prototyping [15].

6 Conclusion

In this paper, we present a new iterative and incremental process for creating a (high-level) interaction design in the context of model-driven generation of GUIs, and its enactment in the course of a trial application. For such an automated generation, a (good) interaction design (represented as a corresponding model) is a prerequisite. We show, that such generation can actually be utilized for the creation of an improved interaction design. This defined and concrete process is consistent with the usual approach to interaction design in general. It involves designing alternatives, prototyping and evaluating, and how these activities are to be repeated for informing each other. In terms of process execution, it adopts from software development, where the artifact (here an interaction design) grows incrementally, iteration by iteration.

The significance of this work is that GUIs for multiple devices are typically derived from the same interaction model, which in case of UCP also specifies the interface between GUI and application back-end. Achieving a stable interaction model allows developing the GUIs and the back-end concurrently, which (potentially) shortens the development time of the overall application.

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

[3] `http://ucp.ict.tuwien.ac.at/UI/accomodationBookingSmartphone`
[4] `http://ucp.ict.tuwien.ac.at/UI/accomodationBookingDesktop`

References

1. Preece, J., Rogers, Y., Sharp, H.: Interaction design: beyond human-computer interaction, 3rd edn. John Wiley & Sons (2011)
2. Raneburger, D., Kaindl, H., Popp, R., Šajatović, V., Armbruster, A.: A process for facilitating interaction design through automated GUI generation. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing (2014)
3. Larman, C.: Applying UML and Patterns, 3rd edn. Prentice Hall (2005)
4. Larman, C., Basili, V.: Iterative and incremental development: a brief history. Computer 36(6), 47–56 (2003)
5. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
6. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley (2004)
7. Deemer, P., Benefield, G., Larman, C., Vodde, B.: The Scrum primer, Version 1.2 (2010)
8. Cantor, M.: Rational unified process for systems engineering. Rational Edge, IBM (August 2003)
9. Kaindl, H., Falb, J., Arnautovic, E., Ertl, D.: Increments in an Iterative Systems Engineering Life Cycle. In: Proceedings of the 7th European Systems Engineering Conference (EuSEC 2010), Stockholm, Sweden (April 2010)
10. Popp, R., Raneburger, D.: 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., Szyperski, C. (eds.) E-Commerce and Web Technologies. LNBIP, vol. 85, pp. 233–245. Springer, Heidelberg (2011)
11. 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)
12. Falb, J., Kavaldjian, S., Popp, R., Raneburger, D., Arnautovic, E., Kaindl, H.: Fully automatic user interface generation from discourse models. In: Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI 2009), pp. 475–476. ACM Press, New York (2009)
13. Popp, R., Raneburger, D., Kaindl, H.: 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 2013. ACM, New York (2013)
14. Popp, R., Kaindl, H., Raneburger, D.: Connecting interaction models and application logic for model-driven generation of Web-based graphical user interfaces. In: Proceedings of the 20th Asia-Pacific Software Engineering Conference, APSEC 2013 (2013)
15. Falb, J., Popp, R., Röck, T., Jelinek, H., Arnautovic, E., Kaindl, H.: UI prototyping for multiple devices through specifying interaction design. In: Baranauskas, C., Abascal, J., Barbosa, S.D.J. (eds.) INTERACT 2007. LNCS, vol. 4662, pp. 136–149. Springer, Heidelberg (2007)