Automatic Generation of the Behavior of a User Interface from a High-level Discourse Model

Roman Popp, Jürgen Falb, Edin Arnautovic, Hermann Kaindl, Sevan Kavaldjian, Dominik Ertl, Helmut Horacek
Vienna University of Technology, Institute of Computer Technology
A–1040 Vienna, Austria
{popp, falb, arnautovic, kaindl, kavaldjian, ertl, horacek}@ict.tuwien.ac.at

Cristian Bogdan
Royal Institute of Technology, School of Computer Science and Communication
S-100 44 Stockholm, Sweden
cristi@csc.kth.se

Abstract

In addition to the structure and “look” of a user interface (UI), its behavior needs to be defined. For a fully-automated UI generation, of course, it will have to be generated fully automatically as well. We avoid that finite-state machines or similar would have to be created manually by a UI designer. Instead, we start from a largely declarative high-level discourse model including a few procedural constructs. Based on our definitions of the procedural semantics of all parts of such a discourse model, we are able to automatically generate a finite-state machine that fully defines the behavior of the generated UI. In this way, we show how automatic generation of the behavior of a user interface is possible from a high-level discourse model.

1 Introduction and Background

Since manual creation of user interfaces (UIs) is hard and expensive, we strive for automatic generation. Instead of generating them from simple abstractions, we let an interaction designer model discourses in the sense of dialogues (supported by a tool). From such a high-level declarative discourse model, we have been able to automatically generate the structure and even the “look” of a UI [1], but the generation of the behavior of the UI turned out to be more difficult than expected.

The key ingredients of our declarative discourse models are communicative acts as derived from speech acts [9], adjacency pairs adopted from Conversation Analysis [6], and RST relations inherited from Rhetorical Structure Theory (RST) [7]. Communicative acts represent basic units of language communication. Thus, any communication can be seen as enacting of communicative acts, acts such as making statements, giving commands, asking questions and so on. Communicative acts carry the intention of the interaction (e.g., asking a question) or issuing a request. Adjacency pairs are sequences of talk “turns” that are specific to human (oral) communication, e.g., a question should have a related answer. RST relations specify relationships among text portions and associated constraints and effects. The relationships in a text are organized in a tree structure, where the rhetorical relations are associated with non-leaf nodes, and text portions with leaf nodes. In our work we make use of RST for linking communicative acts and further structures made up of RST relations.

While the declarative models made up from these ingredients are useful for specifying discourses, automatic generation of the behavior of a related UI turned out to be difficult. Therefore, we recently included a few procedural constructs explicitly into our discourse models, rather than overloading declarative constructs with complex procedural semantics. Still, we had to define procedural semantics for all the constructs potentially included in a discourse model according to our approach. We did this using dialogue statecharts.

Based on these definitions, we had to figure out how to combine all these statecharts into a composite one integrating the behavior of all the constructs involved. For graphical user interfaces (GUIs), we had previously defined partitioning of a discourse tree according to dependencies between variables. Finally, we have integrated handling such dependencies with dialogue statecharts to achieve a finite-
state machine (FSM) that defines the behavior of a graphical user interface.

We also strive for a wide range of multimodal interfaces, including speech input and output. Our discourse models do not include any modality-related information about physical input and output, so it should be generated in the course of rendering for a specific user interface. This has consequences for generating the behavior of a user interface, however, since speech must be fully sequential, while a graphical UI on a reasonably large screen allows for some interaction being quasi-parallel.

We explain all this using a simple online shop as a running example. The simplified scenario starts with customer authentication. Once it is successful, the customer is asked about a product category. The answer enables the software to present the information about the products of this category for selection by the customer. Finally, there is checkout with a credit card.

In summary, this paper shows how the behavior of a user interface can be generated automatically from a largely declarative high-level discourse model including a few procedural constructs. The automatic generation of the FSMs for defining this behavior is implemented. So, this means another major step on the way to fully-automatic generation of user interfaces from our high-level discourse models.

The remainder of this paper is organized in the following manner. First, we present a metamodel of our discourse models that also includes procedural constructs. After defining the procedural semantics of the elements of such discourse models in terms of statecharts, we show how these machines can be combined with each other and with handling dependencies between variables. This leads to an automatic generation of the overall behavior specification of the generated UI. Finally, we discuss this approach in a larger context and compare it with related work.

2 Our Discourse Metamodel Including Procedural Elements

We have developed a metamodel which defines what the structure of the discourse models should look like in our approach. We explain it here using the UML class diagram\(^1\) in Figure 1 and through the example shown in Figure 2.

[Figure 1. Conceptual discourse metamodel.]

The conceptual metamodel in Figure 1 shows that a Discourse contains communicative acts, adjacency pairs and discourse relations. A single interaction is represented with a Communicative Act where its type specifies the intention of the utterance like asking a question about user identification data as shown by the top left Open Question in our example in Figure 2. Further, each communicative act carries propositional content referring to the domain of discourse. The propositional content is shown in our discourse model within each communicative act below its type.

Adjacency pairs model typical sequences of communicative acts within a dialogue that include turn-taking like question – answer or offer – accept. In our discourse models, communicative acts belonging to an adjacency pair are connected by a diamond as shown in Figure 2. In the special case of just informing the other dialogue partner without making a dialogue turn, the adjacency pair will degrade to an adjacency pair with only an opening communicative act as shown in the satellite branch of the Background relation in Figure 2. Adjacency pairs may contain embedded dialogues like clarification dialogues, which may become necessary before a communication party is able to answer a question, for example. In our metamodel, adjacency pairs are modeled as association classes.

Discourse Relations relate Nodes which can be communicative acts or other discourse relations, thus building up the hierarchical structure of the discourse. Discourse relations are further specialized into RST Relations (from Rhetorical Structure Theory) and Procedural Constructs.

Most RST relations, represented by rectangular boxes in our discourse models, describe a subject-matter relationship between the branches they relate, e.g., an Elaboration states that the satellite branch elaborates the dialogue executed in the nucleus branch. RST relations do not imply any particular execution order but eventually may suggest one. For example, the Elaboration in our online shop example allows the presentation of the question on selecting a product category and the question on selecting a product of the currently active product category at the same time on a graphical user

\(^1\)At the time of this writing, the specification of UML is available at http://www.omg.org.
Figure 2. Online shop discourse model.
interface. This presentation should be in some hierarchical manner to emphasize their elaboration relationship (category – instance). If there is not enough real estate available on the screen, the Elaboration relation may suggest to serialize the dialogue in a way that the dialogue about the instances is executed after execution of the dialogue about the category.

If there is a formal dependency between category and instance in the sense of variable binding, such a sequence can be derived along the lines of [1]. More generally, it turned out to be useful to be able to prescribe particular sequences and repetitions eventually based on the evaluation of some conditions. RST relations are not sufficient for this purpose and, therefore, we have introduced Procedural Constructs into our tree structure.

Procedural constructs, represented by hexagonal boxes in our discourse models, provide means to express a particular order between branches of the discourse tree, to specify repetition of a branch and to specify execution of different branches. Thus, our procedural constructs add control structures to our discourse trees that are more complex than usual if-then-else or repeat-until constructs in typical procedural programming languages. When operationalizing the discourse tree, these procedural constructs also determine which information cannot be presented together on one screen of a graphical user interface.

While a discourse model using discourse relations as described above represents a set of possible dialogues, the concrete flow of communication in a concrete dialogue will depend on many factors like the application logic, user interactions, used modality (e.g., graphical user interface or speech) and further constraints, like real estate on a screen. Additionally, a given modality determines how many dialogues can be dealt with in a quasi-parallel manner. A completely serialized interface corresponds to an interface without any parallel options, e.g., speech input and output.

To support the explanation of procedural semantics and the generation of the behavior specification below, let us briefly explain our simplified discourse model for the online shop shown in Figure 2. Communicative acts belonging to the online shop are shown by dark (green) rounded boxes and communicative acts to be uttered by the customer are shown by light (yellow) rounded boxes. This online shop example, first, requires a login by asking the customer for username and password. After checking username and password, the customer can select a product category or, alternatively, exit by accepting the checkout offer. If the customer has selected a product category, she is able to elaborate the selected category and select a product of this product category to add it to her shopping cart. The customer has also the possibility to change the product category or to check out at any time. If the customer decides to check out, she has to enter her payment information (credit card information), if she has some products in her shopping cart. Otherwise, the discourse is finished after accepting the checkout. The online shop discourse model shown in Figure 2 covers also the scenario when a user tries to login too often. In this case locked() becomes true, e.g., the application logic locks the account after three failed tries. The dialogue ends here by informing the user about the login failure.

3 Procedural Semantics

An important issue is the operationalization of our discourse models for making them executable. In order to achieve this operationalization, we specify the procedural semantics of our discourse relations using statecharts. As specified above, we distinguish two different kinds of Discourse Relations: Procedural Constructs and RST Relations. We select here examples of each and present their procedural semantics using statecharts.

IfUntil is a procedural construct that we found useful for defining a certain control structure in our discourse models. It is more complex than the usual procedural statement in a typical procedural programming language. In fact, it may be thought of as some combination of an if-statement and a conditional loop. A statechart defining its procedural semantics can be found in Figure 3. If Condition is fulfilled, the discourse continues in the Then branch. Otherwise, there are two possibilities:

1. the Tree branch can be performed again and again until Condition becomes fulfilled, or
2. the discourse can continue in the optional Else branch, if the RepeatCondition is not fulfilled. If the Else branch is missing, this subtree with the IfUntil as root
node is considered as completely executed and the dialogue continues on the next higher level in the discourse or stops if the IfUntil node is the top-level root node.

Checking whether Condition and RepeatCondition are fulfilled or not is performed by one of the parties involved in the dialogue. In our example, the online shop and not the user checks both Condition and RepeatCondition.

Alternative is an RST Relation which represents that the discourse can be performed in either branch of the Alternative relation. Thus, both branches can be started to present alternatives as indicated by the two concurrent paths (separated by the dashed line) in the statechart in Figure 4. However, only one branch can be finished, specified in Figure 4 by transitions from the inner nucleus sub states to the outside final state. That is, if a transition triggers, the overall state and, therefore, all of its sub states will terminate.

Background is an RST relation which states that the communication performed in the satellite increases the ability to understand the communication performed in the nucleus. From the procedural point of view, this basically means that the communication sequence is not defined, and the nucleus and satellite could even be performed in parallel (e.g., presented at the same time if the communication is through a graphical user interface) as shown in the statechart in Figure 5. However, if the communication in the nucleus is completed, the operationalization of the whole relation is considered completed as well, specified by the transition from the nucleus sub state to the outside final state.

4 Generating Composite Statecharts

Based on the statecharts defining procedural semantics for discourse relations, we can generate composite statecharts for whole discourse trees to specify the set of possible dialogue sequences. The main idea is to build a higher-level statechart by combining the statecharts in a hierarchical manner forming a hierarchical statechart. The composition is done by traversing the tree and recursively applying statechart mappings of the corresponding discourse relations. The statechart of one discourse relation is included as a submachine state in the statechart of the higher-level statechart. Therefore, the hierarchy of the overall statechart corresponds to the hierarchy of the discourse tree.

Let us explain the composition process by combining the statechart mappings of the second IfUntil (the lower one in Figure 2), the Alternative and the Elaboration discourse relations in our online shop example. For specifying each discourse relation’s procedural semantics, we use a statechart like the ones defined in the previous section. In addition, we need a statechart for adjacency pairs to build a complete hierarchical statechart. Since an adjacency pair describes a typical sequence of communicative acts, e.g., a question followed by an answer, this statechart simply consists of a sequence of two states representing the corresponding dialogue states after the utterance of the corresponding communicative act. Thus, communicative acts model transitions between dialogue states.

Figure 6 shows the part of the overall composite statechart that results from the composition of these three discourse relations and the adjacency pairs. Explaining the statechart in Figure 6 in a top-down manner, we start with the IfUntil procedural construct, which will be mapped to the statechart in Figure 3. In the resulting composite statechart, the large composite states correspond to the Tree and Then states in Figure 3. The Else state in Figure 3 is missing in the resulting composite statechart, since there is no else branch in the discourse model. The Condition and RepeatCondition decision points in Figure 3 are mapped to the corresponding Condition and RepeatCondition decision points in Figure 6.

Going one level down in the online shop’s discourse model, we fill the composite states with the corresponding statecharts. Thus, the Then state of the IfUntil is filled with the statechart of the Checkout Data adjacency pair consisting of two states describing the sequence of the Checkout
Data question and answer. The Tree state of the *IfUntil* is replaced with the statechart of the *Alternative* RST relation shown in Figure 4. It is shown in Figure 6 by the two concurrent submachines in the left composite state labeled Alternative.

On the next level, one of the nuclei in the Alternative statechart in Figure 4 is replaced with a statechart for the Checkout adjacency pair, whereas the other nucleus is replaced with the statechart of the Elaboration RST relation. This process is applied recursively until it terminates at the leaves of the discourse tree. Thus, in Figure 6 the Background state represents the whole satellite branch of the Elaboration RST relation. The choice of elaborating in this example is up to the online shop system, but there may be necessary conditions that have to be fulfilled to be able to elaborate. These “necessary conditions” depend on the information (bound variables) required by the communicative acts in the optional branch of the RST relation and is similar to the tree dependency analysis for user interface generation described in [1]. To specify these “necessary conditions” for these decision points in the composite statechart, the discourse tree has to be analyzed as follows:

1. First we search for communicative acts which need a variable to be bound for getting uttered and depends on other variables, i.e., we have to make sure the required information is available. In our running example, this is (only) the case for the productCategory.products variable in the Add Product closed question. This means that the list of products to select from is only available after a product category is bound.

2. Afterwards, we search recursively for an RST relation in an upper level of the discourse tree which does not force the execution of the subtree we are currently in. The matching relation in our running example is the RST relation Elaboration. The variable to be bound (in this example the productCategory variable) defines
the necessary condition of the decision point of the relation, which is responsible for executing the subtree.

5 Integrating the Overall Behavior Specification

Up to this point, we have described how to build a dialogue state machine which is capable to represent the set of possible flows of communicative act utterances (set of possible dialogues). These dialogue state machines contain sequences of states and decision points allowing the system and the user to influence the advancement of the dialogue, but they also include some freedom of choosing dialogue paths through order-independent sequences modeled in the form of concurrent sub-state machines. Nevertheless, these dialogue state machines do not make any statements about the behavior of user interfaces per se. The actual behavior depends on the modality we use for the user interface. In case of speech input and output, these dialogue state machines can serve immediately the implementation of a dialogue system. In case of a graphical user interface, we need additional mechanisms and algorithms to derive presentation units and their behavior, i.e., transitions between the presentation units. A presentation unit represents a set with a maximum number of communicative acts that can go in parallel. If there is enough real estate on a given screen, a presentation unit can be equivalent to an application window.

The following subsections describe how to derive presentation units from the composite statechart by aggregating information, and how to derive a GUI statemachine from the composite statechart. The GUI statemachine models the transitions between the presentation units and thus the behavior of the graphical user interface.

5.1 Presentation unit generation

By analyzing the composite statechart we are able to derive the set of possible presentation units containing the maximum possible information that can go in parallel. The generation of the presentation units is achieved by traversing the statechart for each possible set of truth values of the decision point conditions until reaching statecharts corresponding to adjacency pairs (leaves of our discourse trees). During this traversal, all the dialogue states are collected that can be represented together, in principle, on a screen of a graphical user interface. The algorithm for generating the structure of each presentation unit (e.g., widget hierarchy) is based on rendering heuristics and is described in [4].

For instance, in our online shop example we can infer the product category selection presentation unit shown as an automatically generated window in Figure 7, starting from the situation where a user is logged in, the user did not request to checkout (checkout = false) and a product category is not set yet. When traversing the composite statechart in Figure 6 starting with this set of truth values at the Initial state, we arrive at the alternative composite state in a first step. Since this state contains two parallel submachines, we go to the “Checkout offer and accept” adjacency pair at the top and to the Elaboration composite state below. Continuing in the Elaboration composite state, the Background state at the bottom is bypassed due to the fact that no product category is set. In the upper part of the Elaboration composite state we go to the Select Product Category adjacency pair statechart. Since we stop at each adjacency pair statechart, we get a presentation unit consisting of the closed question Select Product Category and the offer Checkout as shown in Figure 7.

Once we set a product category, we can also explore the Background state within the Elaboration composite state and get a presentation unit that contains in addition Informing on volume discount and the closed question Add Product. This presentation unit is shown in Figure 8. When setting checkout to true and the shopping cart is not empty, we get a presentation unit operationalizing only the Checkout Data adjacency pair shown in Figure 9.

For generating transitions between the presentation units in the next section, we have to save the discovered mapping between the set of truth values of the conditions and the corresponding presentation unit, e.g., in a lookup table.

5.2 Defining transitions between presentation units

To complete the behavior specification of user interfaces, we have to specify the transitions between presentation units. To specify them, one has to figure out which decision point condition in the dialogue state machine depends on which event. Events can be either the utterance of communicative acts or external events (e.g., timer events). In the
online shop example of Figure 2, the truth values of all conditions depend only on the outcome of uttering communicative acts and they do not change without communication by external means.

The transitions from one presentation unit to the other can be derived by applying the algorithm sketched below and will result in a user interface state machine like the one shown in Figure 10 for our online shop example.

1. For specifying an initial state, use default truth values for the conditions of each decision point (ideally, derived from a standard situation in the domain of discourse). E.g., in our online shop example this could be to unset checkout, username, password and product-Category.

2. Analyze dependencies between the conditions of the decision points and the communicative acts, to figure out which condition is influenced by which communicative acts. For example, the two conditions of the IfUntil relation depend on the checkout variable which can be only changed by the communicative acts of the “Checkout offer – accept” adjacency pair.

3. Select one of the generated presentation units and get all conditions whose truth values can be changed by the outcome of uttering any communicative act allowed in the selected presentation unit. E.g., within the Login presentation unit, there is only an answer communicative act allowed which may change the locked() or check(username, password) conditions.

4. Change the truth value of each selected condition and look up the presentation units that correspond to the new sets of condition values in the association table created during the presentation set generation process described in the previous section. Afterwards, add transitions to the presentation unit state machine between its original state and its looked-up states.

5. Repeat steps 3 und 4 until all presentation units are processed, resulting in a complete finite-state machine.

5.3 Behavior of the resulting UI

The presentation unit state machine in Figure 10 specifies the resulting behavior of a user interface. In our online shop example, each presentation unit is directly mapped to a window for a graphical user interface shown in Figures 7, 8 and 9, corresponding to the lower three states in Figure 10, in that order.

Starting with the login presentation unit, the answering of the Login closed question modifies the variables username and password. This in turn has effects on the conditions check(username, password) and locked(). If the check of username and password evaluates to true, the dialogue continues in the presentation unit for selecting the product category. The conditions evaluated for this presentation unit transition result directly from the IfUntil procedural construct in the discourse model.

Looking at the transition between the PU Product Category (corresponding to Figure 7) and the PU Product (corresponding to Figure 8), the transition condition bound(productcategory) does not result from procedural constructs, but from variable binding. This is analyzed during the process of composing the overall dialogue statechart. Thus, selecting a product results in displaying additional information on volume discount and a closed question about products to add to the shopping cart. Once a transition leads to the final state, the dialogue and thus the application ends.

This generated presentation unit state machine describes what can go in parallel and be presented together at maximum. Thus it is the other end of a spectrum of dialogues and user interfaces ranging from completely serialized to this maximum of parallel dialogues. This allows for different modalities to be used for input and output, independently developed from the discourses. There are modalities which allow parallel (multi-dimensional) dialogues like graphical user interfaces. In contrast, modalities like speech or console text are serial interfaces which lead to serialized inter-
action. Thus, interaction follows a step-by-step fashion, as no parallelism of input or output is possible.

A multi-dimensional interface may be transformed to a serialized form in case there is not enough real estate. As mentioned in [5], serialized interaction has two problems: First, when the interaction path splits, there is no simple choice which interaction object to activate next. Second, backward references in a dialogue may lead to confusion when using the interface. To avoid this, a clean composition of the interface is necessary. The interface elements have to be sorted topologically. This ordering fulfills all dependencies, as long as there are no cycles. The rendering process of a concrete implementation of speech or a GUI based on Figure 6 must take the possible parallelization of the physical representation of the modality into account. It usually (partly) serializes all parallel behavior to a modality-specific amount. Different heuristics and the semantics of RST relations with respect to content support this serialization. Interactions for a speech interface always have to be serialized. When two or more dialogues are in parallel and are not serializable because they are RST alternatives, a menu presents all possible alternatives to the conversation partner in a serialized form.

6 Discussion

In this paper we have improved upon the variable binding approach presented in [1], which we subsequently found to be unnatural to model with in specific modeling situations such as modeling login. We have recognized that one of the root causes of this modeling problem lies in the variables (and their binding) being present only in the leaf nodes of the discourse model trees. This does not make sufficient use of the tree structure. For example, the whole “shopping” (post-login) subtree can only be instantiated if there is a user logged in, and the only way to express this with the binding approach would have been to use the “user” variable in each and every leaf of the shopping subtree. However, few of these nodes actually need the user identity variable directly.

Introducing procedural constructs and assigning procedural semantics to RST relation nodes, as illustrated in this paper, allows the modeler to gain more powerful and generic control of the behavior of the resulting interface. In the example above, modeling in the root node of the shopping subtree has an impact on the whole subtree, so the additional mentioning of a variable in each subtree leaf node can be avoided.

We also pursue a more generic approach than leaf binding annotations and relational node semantic assignment. This approach involves assigning information to the tree branches in the form of conditional expressions. We have began using such expressions already in this paper, since procedural semantics of relations in themselves are not enough (e.g., a Condition relation needs a conditional expression) and we are currently investigating more generic use of conditional expressions in the form of constraints on any tree branch. Working with such conditional expressions is also more flexible both because it allows for more complex conditions than the “is bound” vs. “is not bound” distinction from the binding approach and because it needs fewer procedural constructs and less procedural interpretation of the RST relations.

As mentioned above, we allow for an arbitrary degree of dialogue parallelization. This includes completely (speech) or partially (GUI) serialized interfaces for input and output. When a system based on our discourse engine is coupled with a number of rendering schemes for input or output, serialized interfaces may be interpreted together to allow parallel behavior to a higher degree. Thus, a more flexible communication flow is provided.

7 Related Work

A main part of any user interface is its behavior. Many approaches have recognized that it is advantageous to go beyond programming of the user interface behavior and to use some kind of dynamic models for its specification and
execution. Most of these approaches utilize some kind of state machinery for this.

Book and Gruhn [2] present an approach to describe the dialogue by dialogue graphs that are enhanced state machines. They present the formal semantics of the Dialogue Flow Notation (DFN), that provides constructs for the design of modular navigation models. These dialogue graphs are on the same level as our composite statecharts describing all possible dialogue paths. The dialogue graphs have to be specified by the UI designer, whereas our composite statecharts are automatically generated from the high-level discourse model.

ConcurTaskTrees [8] are a way of hierarchical task modeling where different temporal relations among tasks can be specified (e.g., enabling, disabling, concurrency, order independence, and suspend-resume). Out of such task models, the user interface can be created through several steps of model transformations. CTT also allows the derivation of presentation units consisting of a set of tasks and their transitions, but their transition graph is more abstract than our presentation unit state machine, skipping detailed trigger events for the transitions.

Elkoutbi et al. [3] present an approach that generates a user interface prototype from scenarios. Scenarios are enriched with UI information and are automatically transformed into UML statecharts, which are then used for UI prototype generation. In contrast to this approach, we model classes of dialogues supporting a set of scenarios.

We are not aware of any approach that would generate the behavior of user interfaces automatically from a largely declarative high-level discourse model. In addition to taking a communication-centric approach and modeling human-machine communication in the form of discourses, we generate user interfaces automatically, including their behavior.

8 Conclusion

In this paper, we address the issue of automatic generation of the behavior of a user interface from a high-level discourse model. While the generation seems to be possible from a purely declarative model, the complete behavioral operationalization is facilitated by introducing behavioral constructs into the otherwise declarative model.

We defined procedural semantics for both the procedural and the declarative constructs of the discourse model using statecharts. Based on these building blocks for operationalization, we showed how composite statecharts for whole discourse trees can be generated. These directly define all possible behaviors of a fully serialized user interface, e.g., using speech input and output. In contrast, for a graphical user interface with potentially infinite screen size, we showed a derivation of a finite-state machine from a composite statechart for defining its behavior. The behavior of multimodal interfaces can be defined in the spectrum between these extremes.

In this way, we have defined how the behavior of user interfaces can be automatically generated from a largely declarative high-level discourse model. Taking this together with generating structural UI models enables the automatic generation of user interfaces from such discourse models.

9 Acknowledgements

This research has been carried out in the CommRob project (http://www.commrob.eu) and is partially funded by the EU (contract number IST-045441 under the 6th framework programme).

References