2014 47th Hawaii International Conference on System Science

Towards Fully Declarative High-level Interaction Models: An Approach Facilitating Automated GUI Generation

Filip Kis, Cristian Bogdan
Media Technology and Interaction Design
KTH Royal Institute of Technology, CSC
Stockholm, Sweden
{*fkis, cristi*}*@kth.se*

Hermann Kaindl, Jürgen Falb
Institute of Computer Technology
Vienna University of Technology
Vienna, Austria
{*kaindl, falb*}*@ict.tuwien.ac.at*

***Abstract*—Models of high-level interaction design are usually based on *procedural* representation. For knowledge representation and reasoning, however, *declarative* representations are preferred. In this paper, we define purely declarative high-level interaction models based on theories of human communication. In contrast, earlier attempts to define purely declarative models resulted for pragmatic reasons in a mixed representation including procedural constructs within the overall declarative model structure. We show how the declarative models can be operationalized into behavioral (abstract) UI models corresponding to those generated from the mixed representation. Based on an implementation integrated with an existing framework for GUI generation, we show that and how it is possible to automatically generate GUIs from purely declarative models as well.**

***Keywords*-interaction design; discourse model; declarative representation; GUI generation;**

I. Introduction

High-level interaction models in the context of automated generation of user interfaces, on the level of concepts and tasks of Cameleon Reference Framework [1], are today primarily *procedural* models. That is, they encode behavior (e.g., a sequence of tasks) directly and explicitly.

In this paper, we propose *declarative* representation instead. We take the definition of declarative from Knowledge Representation and Reasoning (KR) in Artificial Intelligence (AI) where it means to (only) represent *what* is needed, and not an explicit control flow [2]. Mathematical logic is an example of declarative representation.

The high-level interaction models employed in this paper are discourse-based models [3], [4], [5]. In such models, the high-level interaction between the user and the interactive system is represented, according to certain principles of human communication, as a discourse. For instance, the system provides some information (presents data) to the user and asks questions (asks for input). The user provides answers (gives values for the input) and, depending on the answers, the system provides some other information or asks more questions. Discourse modeling applies such concepts (informing, questions, answers, etc.) for specifying Discourse Models. From such models, automated generation of (G)UIs is already possible.

In the area of discourse-based models there was an attempt to come up with declarative models, but after a while certain constructs were needed in order to achieve a complete operationalization of the model that turned out to be Procedural Constructs [4]. In effect, discourse-based models are currently a combination of procedural and declarative constructs (a bit like the programming language PROLOG).

In the context of interaction modeling, the following example should motivate a certain advantage of a declarative representation. Assume a sequence of interactions, where a product category needs to be selected before a product can even be chosen (using a GUI). Of course, any task-based approach and the current discourse-based approach with Procedural Constructs can represent such a sequence explicitly. However, it is hard to automatically check whether such a sequence is correctly specified. Our proposal for a fully declarative approach determines the sequence automatically if and only if the variable of product category is bound. In this sense, the sequence is also automatically correct.

So, there was a challenge to devise a purely declarative approach to discourse-based modeling. This meant to explore discarding the Procedural Constructs for this endeavor, and to be complete and precise in terms of logic conditions assigned to the Discourse Model. This is necessary for enabling operationalization of such models in the course of generating user interfaces from them.

This paper presents the declarative models, inspired by human communication, and an algorithm for their operationalization that results in the same finite-state machinery as the one operationalized from models that include Procedural Constructs. Furthermore, we use the existing tools to generate the same graphical user interfaces (GUIs) based on these finite-state machines.

The remainder of this paper is organized in the following manner. First we provide some background material on discourse modeling for GUI generation. Next, we sketch the essence of declarative vs. procedural Discourse Models. After that, we define the purely declarative models and

978-1-4799-2504-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HICSS.2014.59

demonstrate, on an example, how they can be operationalized. Based on that, we formalize the algorithm used for the operationalization. Next, we present how these operationalized models can be used to generate GUIs. Finally, we reflect on our results and compare them with related work.

II. Discourse modeling for UI generation

In discourse modeling the fundamental unit of communication (information, question, answer, etc.) is a *Communicative Act*, derived from Speech Act Theory [6]. The Communicative Acts are represented as leafs in the model trees of Figures 1 and 2. They are hierarchically organized according to *Rhetorical Structure Theory* (RST) [3], [7]. RST proposes a way to structure the parts of a discourse, with a focus on the rhetorical value of these parts. For example one sentence in the discourse can be an $Elaboration$ of another, and a third question can be an $Elaboration$ of the first $Elaboration$, thereby forming a hierarchical tree structure (RST relations represented as rectangles in Figures 1 and 2). Other such RST Relations are: $Background$ (a sort of reversed $Elaboration$), $Alternative$, $Joint$ etc. In addition to RST Relations, the approach presented in [4] introduces Procedural Constructs such as the $IfUntil$ shown in Figure 1. Figure 2 shows a similar model with only standard RST Relations, as the declarative approach presented here does not need to extend RST.

Along with RST, several other theories of human communication are at the foundation of UI generation based on discourse modeling. RST defines the relations between *Adjacency Pairs* derived from Conversation Analysis [8]. This research field proposes ways of conceptualizing naturally-occurring human communication. For example, answers are usually *adjacent* to questions that were posed. In discourse modeling, Adjacency Pairs are represented as diamonds, as illustrated in Figures 1 and 2. In other cases, a question may be followed by a whole conversation, before it is finally answered, and Conversation Analysis calls that interim conversation an *Inserted Sequence* which was adopted in the discourse modeling approach to UI generation. Inserted Sequences can be used to split bigger models (of complex UIs) into smaller, more manageable elements.

We illustrate how Communicative Acts are used by way of an example, shown in two versions in Figure 1 (with Procedural Constructs) and Figure 2 (only standard RST Relations), where a dialog between a user ($Customer$) and an interactive system ($OnlineShop$) is modeled. Communicative Acts shown in green can be *uttered* by the $OnlineShop$ system. In yellow we show the Communicative Acts that can be uttered by the $Customer$. The types of Communicative Acts used in the example are $OpenQuestion$ (a question accepting many kinds of answers), $ClosedQuestion$ (a question that requires the answer to be part of a known set), $Answer$ (to an open or closed question), etc.

Annotations of the RST tree in the Discourse Model represent constraints under which certain communication takes place. For example, in Figure 1 the right branch of the $Elaboration$ node (annotated with $productCategory.products > 0$) in the product category selection takes effect only if the selected category contains at least one product.

Using these modeling elements, the discourse modeling approach can generate GUIs including their behavior [5], [9]. To achieve that, a number of pragmatic decisions have been made, including the introduction of Procedural Constructs. We propose an alternative approach below.

III. Declarative vs. procedural discourse modeling

Generating behavior from the model representation (*operationalization*) is an important aspect for most UI modeling approaches. Operationalization of Discourse Models involves determining a state machine from the Discourse Model. A first attempt to build such a state machine was proposed in [10]. Authors of [4] found that this approach poses challenges in modeling certain behaviors like specific sequences and repetitions. To address this issue they introduced *Procedural Constructs*.

The introduction of the Procedural Constructs was a matter of discussion because it added procedural elements to an essentially declarative model, and mixing discursive with behavioral elements may be viewed as a non-coherent design. On the other side, the Procedural Constructs are easily recognizable by the users of the modeling approach (designers) who are trained, like many people are nowadays, in procedural programming. Since this discussion is not unusual in research or in design (of user interface modeling languages in this case), both sides agreed to aim for testing their approaches with modelers and comparing the results. For that, the first step was to devise a working operationalization of purely declarative Discourse Models, thus showing that it is possible to generate behavior (state machine) from Discourse Models without involving procedural constructs.

The declarative model operationalization, proposed in this paper, makes use of logical constraints that annotate the tree edges. These constraints are used as logical pre-conditions for enabling the respective parts of the communication. Therefore, besides focusing on the declarative nature of RST, the logical constraints are also used in a declarative manner.

Below we demonstrate by example an operationalization of a declarative Discourse Model and present an algorithm for operationalization of such models. The resulting state machine from the example model presented is compared to the results of the operationalization according to the approach described in [4]. We show that the state machines resulting from the two approaches (and therefore also as the GUIs generated based on them) are the same.

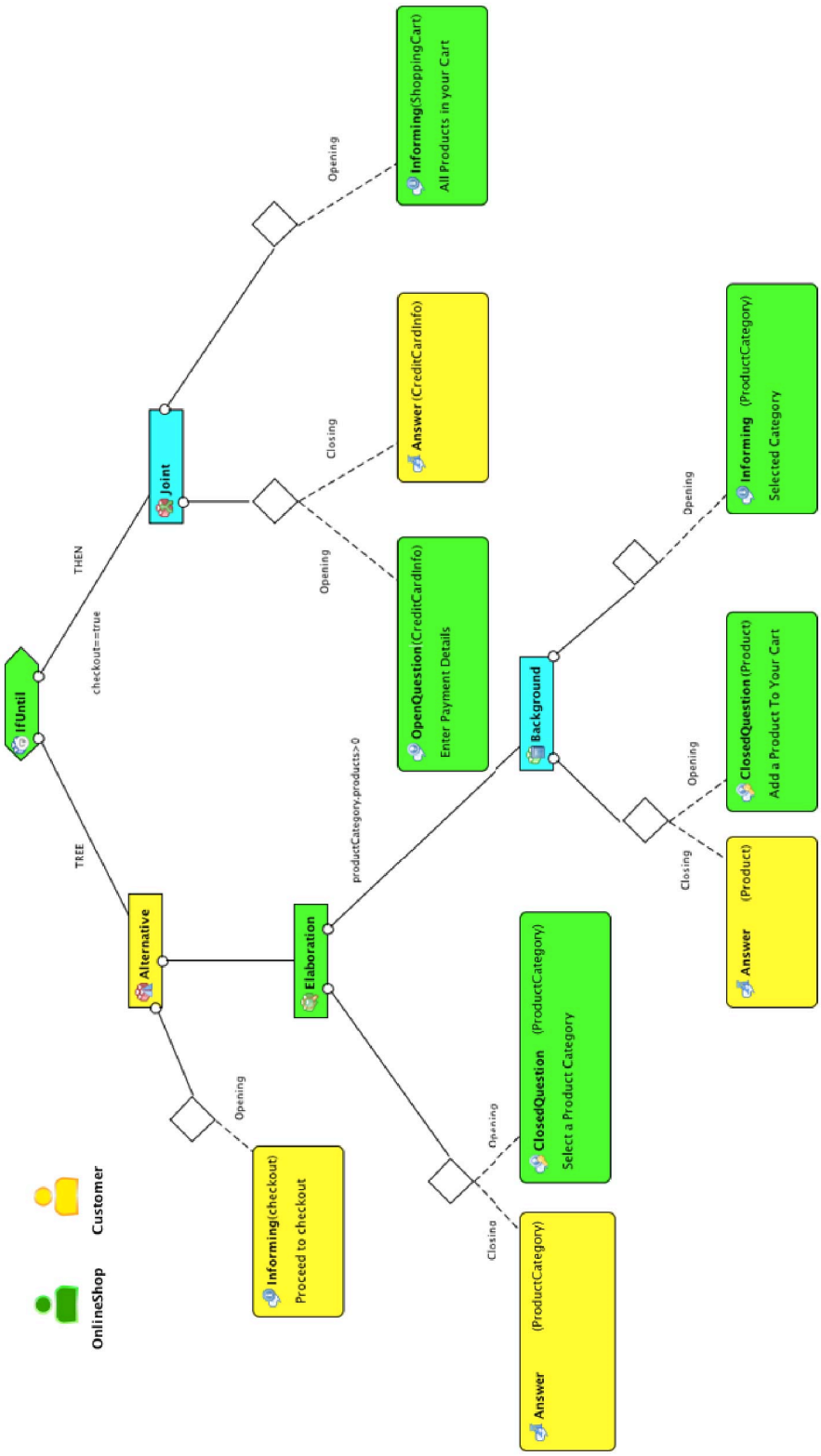

Figure 1. Online shop Discourse Model, procedural version

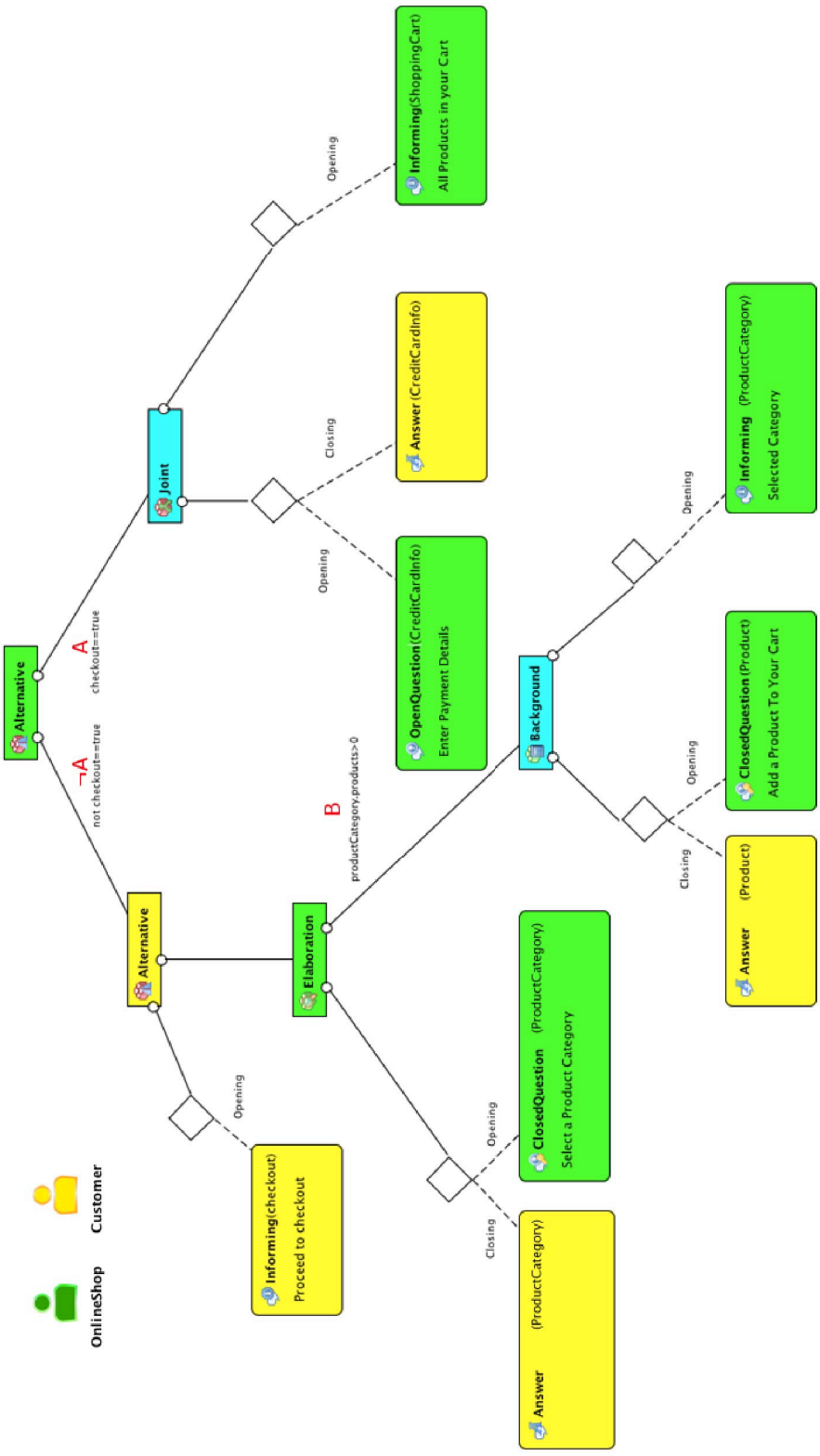

Figure 2. Online shop Discourse Model, declarative version

IV. Declarative model operationalization

Now let us explain how it is possible to automatically generate a state machine, representing the GUI behavior, based on a purely declarative model. A *Partitioning State Machine* (PSTM), used to represent the behavior in discourse modeling, comprises various states that the communication can be in. In principle, each state in the PSTM corresponds to a screen (or dialog box) in the GUI runtime of the Discourse Model. This screen is rendered from a subtree (partition) of the Discourse Model tree that contains only the communication elements present in the respective state. Note that these screens can, in principle, be of potentially unlimited size. Breaking down screens to the size of device displays is explained in [9].

The declarative Discourse Model of the online shop example is shown in Figure 2 and it can be compared with the version of the model in Figure 1 that uses Procedural Constructs. The left part of the dialog models the communication that represents the actual shopping, before the user proceeds to checkout. Since the dialog can either take place before checkout or at checkout, an $Alternative$ RST relation is used to model that.

In the left subtree, a product category selection is made and, when the discourse of product category choice is modeled, an $Elaboration$ of the category takes place, by a discourse allowing the user to choose a product. As an $Alternative$, a checkout option is presented (left most). In the right subtree, a discourse represents the situation when checkout was chosen, providing overview of the selected products and entering the billing information.

Comparing this to the model in Figure 1, we can see that the main difference relation-wise is the lack of the Procedural Construct $IfUntil$ on top, replaced by a declarative (and standard RST) $Alternative$ construct. Also, to model the replacement of $IfUntil$ both edges of the replacing $Alternative$ node need to be annotated with constraints. One keeps the annotation of the $then$ edge of the original node, while the other gets its negated form.

To simplify the description of the declarative approach introduced here, a shorthand notation is used for each constraint (Boolean expression) as follows:

$$A \leftarrow checkout = true \quad (1)$$

$$B \leftarrow productCategory.products > 0 \quad (2)$$

The corresponding constraints, with shorthand names, are also shown on top of the original expression in Figure 2. Based on these constraints and the declarative Discourse Model, we will show how the PSTM can be generated, by first determining the possible states and then the transitions between the states and the events that trigger the transitions.

A. Determining the states

In the proposed declarative approach, the states can be derived by first computing all the constraint combinations present in the Discourse Model tree. In our example (see Figure 2), there are 2 constraints (A, B) which can have one of 2 values ($true$ or $false$) and, therefore, we have $2^2 = 4$ combinations:

$$c_1 \leftarrow \neg A \wedge \neg B \quad (3)$$

$$c_2 \leftarrow \neg A \wedge B \quad (4)$$

$$c_3 \leftarrow A \wedge \neg B \quad (5)$$

$$c_4 \leftarrow A \wedge B \quad (6)$$

From here the viable states are computed by applying the constraint values and tree hierarchy. For instance, in the example tree (Figure 2), for constraint combination c_1 ($\neg A \wedge \neg B$) the right branch of the topmost $Alternative$ is pruned (since it is annotated with A which is $false$) and the right branch of $Elaboration$ (since B is $false$ as well). This leaves a subtree with only leftmost Communicative Acts, which corresponds to the first state, before the $Customer$ made any category selection. Similarly, c_2 results in a subtree that includes also the $Background$ relation (since now B is $true$), which is the second state. Thus we derived the following two states:

$$S1: \quad \neg A \wedge \neg B \quad (7)$$

$$S2: \quad \neg A \wedge B \quad (8)$$

Several constraint combinations can result in the same subtree. For example, for the constraint combinations c_3 and c_4 the whole left branch (annotated with $\neg A$) of the topmost $Alternative$ is pruned because A is true. The remaining subtree (right branch) has no occurrence of constraint B. Therefore, the value of B does not matter and the combinations c_3 and c_4 result in just one state:

$$S3: \quad A \quad (9)$$

Once we have determined which branches remain (after the pruning described above) in each state, the next step is to specify which Communicative Acts are part of the respective state (can be uttered in that state), and therefore can be represented on the GUI screen created from the respective state. For our online shop example, in state $S1$, the Communicative Acts that can be uttered are $ProductCategory\ ClosedQuestion$ (CQ) followed by its $Answer$ (A), together forming an Adjacency Pair denoted $CQ{-}A(ProductCategory)$, and $Informing\ checkout$ (I). This corresponds to the leftmost Adjacency Pair in the tree and the single communicative act above it.

$$CA(S1) = \{CQ{-}A(ProductCategory), I(checkout)\} \quad (10)$$

In state $S2$, after the category has been selected, the operationalization engine will show extra options for adding a product to the shopping cart ($CQ - A(SelectProduct)$) and provide the information about the selected category ($I(Category)$).

$$CA(S2) = \{CQ{-}A(ProductCategory), I(checkout) \\ CQ{-}A(Product), I(ProductCategory)\} \quad (11)$$

In state $S3$, the generated GUI will not contain anything shown previously and only present the Communicative Acts as rendered for entering the credit card information ($OQ - A(CreditCardInfo)$) and the products in the shopping cart ($I(ShoppingCart)$).

$$CA(S3) = \{OQ{-}A(CrediCardInfo), I(ShoppingCart)\} \quad (12)$$

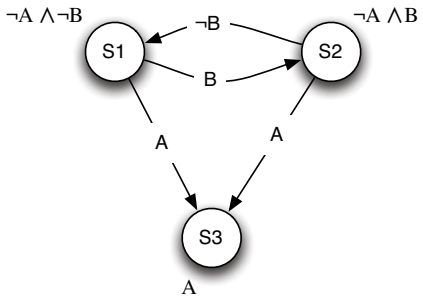

Figure 3. State machine for the online shop discourse.

B. Determining transitions

To determine the transitions between the states, the events that can change the values of the constraints need to be identified. Events can either be the utterance of Communicative Acts or they can be external events (e.g., events like "the robot arrived at place X", from the robot communication model shown in [11], or timer events). In the online shop example, all constraints depend only on the utterances of Communicative Acts and are not modified by external events. The values of the constraints A, B can change based on the following Communicative Acts (noted as $CA(x)$ where x is the constraint):

$$CA(A) = \{I(checkout)\} \quad (13)$$

$$CA(B) = \{CQ{-}A(ProductCategory)\} \quad (14)$$

That is, constraint A, which is $checkout = true$, can only be changed by the leftmost $Informing$ Communicative Act (marked with $set\ checkout$). The $productCategory.products > 0$ constraint, noted B, can be changed by the $Answer$ to the $ClosedQuestion$ with annotation ending in $set\ productCategory$.

After concluding which events (Communicative Act utterances in our case) a given constraint depends on, we can find out in which states these events can occur. This is achieved by checking which state a respective event-firing Communicative Act is part of (10 – 12) . Some of these Communicative Acts can be part of more than one state. This gives us the associations between the states and the constraints. In the online shop example, these associations are as follows (noted as $S(x)$ where x is the constraint):

$$S(A) = \{S1, S2\} \quad (15)$$

$$S(B) = \{S1, S2\} \quad (16)$$

In this example, A can be changed in $S1$ because the $Informing$ Communicative Act that can change A (13) is part of $S1$ and $S2$ (10, 11). B can be changed by the $Answer$ to $ProductCategory\ ClosedQuestion$ (14), which is also part of $S1$ (10) and $S2$ (11).

We now know in which states each constraint condition can change. This information tells us whether transitions between various states exist. For example, the transitions between $S1$ ($\neg A \wedge \neg B$) and $S3$ (A) are obtained by changing the constraint value of A. Since A can change in $S1$ (see (15)), the $S1 \rightarrow S3$ transition exists. Furthermore, since A cannot be changed in $S3$, $S3 \rightarrow S1$ does not exist. The sole transition between $S1$ and $S3$ is visible at the left part of Figure 3. Similarly, there is a single transition from $S2$ to $S3$, which is visible on the right side of Figure 3.

The transitions between $S1$ ($\neg A \wedge \neg B$) and $S2$ ($\neg A \wedge B$) are obtained by changing the constraint value of B. In this case transitions exist in both directions because B can be changed in both states (16). These transitions are shown on the top of Figure 3.

C. Determining events that lead to transitions

In the PSTM the state transitions are triggered by the events (Communicative Act utterances in our example). The next step in the declarative Discourse Model operationalization is to refine the state machine by showing which Communicative Acts need to be uttered for a certain transition to take place. Such a state machine is also produced by the approach using Procedural Constructs described in [4], therefore we aim to generate this state machine to be able to compare results. To generate such a PSTM, we need to transform our abstract state change transitions from Figure 3 to Communicative Act utterances. Each transition is replaced by a number of transitions, one for each $Customer$ Communicative Act utterance (yellow in Figure 2) of the originating state. For example, transition $S1 \rightarrow S3$ denoted as A is transformed to transitions triggered by the utterance of $A(ProductCategory)$ and $I(checkout)$ (see Figure 4), since these are the Communicative Acts that can be uttered by the $Customer$ in state $S1$ as seen in (10).

An utterance of a Communicative Act (for example changing a product category with $Answer(ProductCategory)$) does not imply that a dependent constraint value has changed. For example, constraint B ($productCategory.products > 0$) can stay true between the product category changes. For this reason,

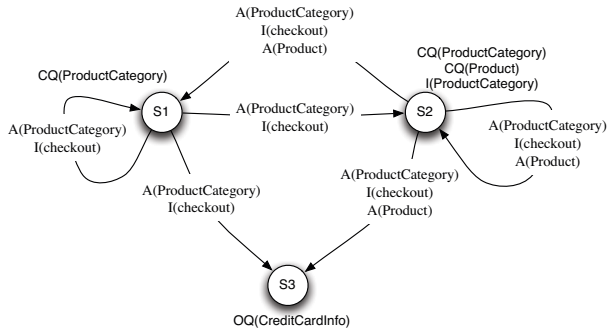

Figure 4. PSTM for the online shop discourse with events that lead to transitions.

the states that have outgoing transitions need to be extended with transitions from themselves to themselves. One such transition is needed for each Communicative Act that can be uttered by the $Customer$ in the given state (see Figure 4). For example, $S1$ has outgoing transitions, thus it is extended with two new transitions to itself corresponding to the $A(ProductCategory)$ and $I(checkout)$ Communicative Acts.

To give a complete picture, Figure 4 also shows, next to each state, the Communicative Acts uttered by the $OnlineShop$ (green in Figure 2). For example, for $S1$ this is only $CQ(ProductCategory)$.

D. Partitioning state machine comparison

Together with the authors of the approach described in [4] we have determined that their approach generates the same state machine, from the Discourse Model in Figure 1, as the state machine shown in Figure 4. Since the two state machines are represented internally in the same data format, it was possible for us to empirically confirm that for the online shop models (Figures 1 and 2) the declarative approach produces the same results (in terms of PSTM) as the approach with Procedural Constructs.

We have performed several comparisons on the PSTMs for several other models as well, and we were able to confirm that the state machines are the same for these cases, too. However, since the construction of a declarative model for a given model with Procedural Constructs is currently done manually, we still lack a formal proof of equivalence of the two methods. This is subject to further research.

V. An operationalization algorithm

Let us now give a general description of the declarative model operationalization algorithm, to show how it can be applied on any declarative Discourse Model.

In its first step, the algorithm traverses the Discourse Model and finds all the constraints present in the tree. It does a symbolic analysis to only get unique absolute constraints, i.e. if a negated version of a constraint was already found, it is ignored. For example, in Figure 2 the annotations $NOTcheckout = true$ and $checkout = true$ represent different occurrences of the same constraint (noted as $A \leftarrow checkout = true$).

Next, the states are determined according to Algorithm 1. In particular, all the combinations of constraint truth values ($\mathbb{C}$) are generated. Each constraint combination (c) is checked against the Discourse Model ($\mathbb{T}$). If any constraint in the combination ($k \in c$) appears on the tree edge in non-negated or negated value ($\mid getConstraint(edge) \mid\equiv\mid k \mid$), it is added to the set of used (appearing) constraints ($\mathbb{U} \leftarrow \mathbb{U} \cup k$). The branches of the tree that are annotated with the negated version of the constraint ($getConstraint(edge) \equiv \neg k$) are removed ($t \leftarrow t - edge$). If the remaining subtree is not empty ($t \neq \emptyset$) and a state with the same constraints combination does not already exist ($state(\mathbb{U}) \notin \mathbb{S}$), the new state ($state(\mathbb{U})$) is added to the set of all states, and the Communicative Acts present in the subtree are associated with the newly found state ($CA(state(\mathbb{U}))$, e.g., 10–12).

Algorithm 1 Determining the states

```
𝕊 ← ∅
for all c ∈ ℂ do
    t ← 𝕋
    𝕌 ← ∅
    for all edge ∈ t do
        for all k ∈ c do
            if | getConstraint(edge) |≡| k | then
                𝕌 ← 𝕌 ∪ k
            end if
            if getConstraint(edge) ≡ ¬k then
                t ← t − edge
            end if
        end for
    end for
    if t ≠ ∅ & state(𝕌) ∉ 𝕊 then
        𝕊 ← 𝕊 ∪ state(𝕌)
        CA(state(𝕌)) ← getCommActs(t)
    end if
end for
```

Next, for each constraint, the algorithm checks which Communicative Acts can change its value ($CA(k)$, e.g., 13,14) and then goes through each state and checks whether these Communicative Acts are part of it. If so, the state is added to the constraint list of dependent states ($S(k)$, e.g., 15,16).

Finally, Algorithm 2 goes through each state ($s_i \in \mathbb{S}$) to generate the state transitions. For each of the state's dependent constraints ($k : s_i \in S(k)$) the algorithm checks whether a change of that constraint (while keeping unchanged the values of other constraints in the state $s_i \wedge \neg k$) would result in any of the existing states ($\Rightarrow s_j$), including itself. If such a state is found (s_j), the transitions from s_i to s_j are generated ($createTransition(s_i, s_j, ca)$) for each Communicative Act

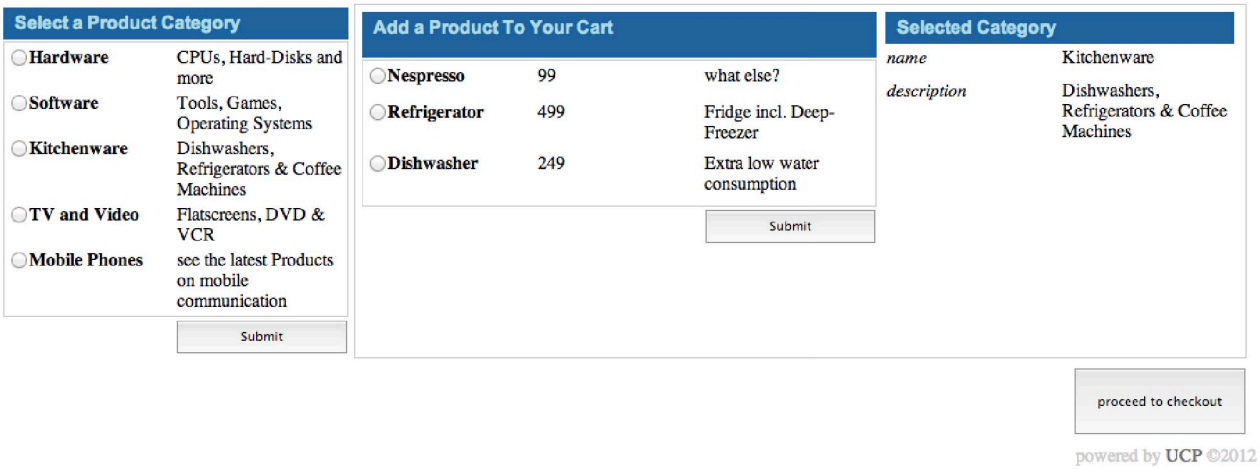

Figure 5. The generated screen

($ca \in CA(s_i)$) in the originating state (s_i) that can be uttered by the user ($utteredByUser(ca)$).

Algorithm 2 Determining the transitions

```
for all s_i, s_j ∈ 𝕊 do
    for all k : s_i ∈ S(k) do
        if (s_i ∧ ¬k) ⇒ s_j then
            for all ca ∈ CA(s_i) do
                if utteredByUser(ca) then
                    createTransition(s_i, s_j, ca)
                end if
            end for
        end if
    end for
end for
```

Having determined states and transitions, the algorithm thus completes the Partitioning State Machine.

VI. Generation of a GUI

The PSTM generated from the Discourse Model is part of the GUI behavior model that takes part in the full process of generation of GUIs described in [9]. As shown in that paper, several other models are required, including the device-specific Structural UI Model that is also generated from the Discourse Model. The changes we have done to the model in Figure 1 to obtain the purely declarative model in Figure 2 have no effect on the generation of the Structural UI Model. For this reason, we were able to run the GUI generation process, in cooperation with the authors of [9], using our declarative model and the state machine operationalized from it, and we were able to generate the same GUI as the original approach from the model with Procedural Constructs. The GUI is illustrated with screen shots in Figure 5.

This result further motivates our claim that the discourse modeling for the generation of GUIs can be achieved by using purely declarative models.

VII. Discussion

Having described how a partitioning state machine can be devised based on a purely declarative model, let us make some reflective remarks. The declarative approach is based on the constraint annotations of the Discourse Model tree. In order to supply for the lack of Procedural Constructs used to generate behavior in the other approach, the declarative approach makes *symbolic* and *algebraic* analysis, in order to determine states and transitions.

We have shown that our approach can produce the same partitioning state machine for discourse model operationalization as the approach based on procedural concepts, but using a purely declarative model instead. The state machine comparison that we have illustrated represents empirical evidence that the declarative approach can model the same behavior and GUI as the other approach.

To be able to demonstrate the approach in this paper, we needed to limit the example to small amount of constraints and Communicative Acts. The algorithm has been tested on larger models and the size of the models has not had a significant impact its performance. An interesting question could be potential state-explosion. Since the number of states are the same as with the approach that uses procedural constructs, this problem should be addressed on the general discourse-based modeling level, and is out of the scope of this paper.

Determining the state transitions is a complex operation, as illustrated. As a potential simplification, we have come to realize that the states are more important to determine than the transitions, because the states are tree partitions that play an important role in rendering [10]. The states often

need to be inspected by the modeler, since the layout of the generated interface is usually a delicate matter for, e.g., interaction designers and other stakeholders. Once states are determined, it should be possible to render the GUI dialog corresponding to the initial state (which can be obtained from the initial values of the involved conditions) and re-evaluate all constraints after each event to determine the next state. We could call this approach to model operationalization a *pseudo state machine*. Still, determining transitions as we did in this paper (through symbolic and algebraic analysis of the Discourse Model tree) has both theoretical significance and practical, implementation importance. The theoretical significance is that it helped us empirically show that the results (in the terms of the partitioning state machine and generated GUIs) are the same as from the procedural approach. The practical importance is that it enabled us to integrate with code from the existing operationalization implementation [5], which is not easily possible with the pseudo state machine sketched.

Even with the current approach, there is a potential to simplify the state machine by analyzing the PSTM further and removing certain transitions that may never occur. The presentation of these simplifications as well as analysis of their implications are left for future work.

VIII. Related work

The declarative vs. procedural discussion in AI Knowledge Representation has been influential to us. The declarative side is represented by, e.g., [12] and the procedural side by, e.g., [13].There are also attempts to devise mixed declarative-procedural approaches [14].

In the context of user interface modeling, from early works [15], [16] to modern approaches [17], the term declarative has been associated to related models. However, the definition of this term in such cases has been taken loosely and not according to AI Knowledge Representation. Their usage of declarative can be understood as "higher-level" as compared to procedural programming code.

Mainstream user interface modeling, which is mainly centered around task models [18], [19], [17], [20], specifies, e.g., sequence and iteration by defining temporal operators based on process algebra. However, these concepts are procedural as they define the order of execution directly and explicitly in the models. As stated in [21], models should be declarative rather than procedural in order to support successive transforms and to be suitable for the use of computer tools.

A low-level declarative UI modeling approach is described in [22]. This approach uses declarative queries to annotate the concrete UI model and generate the controller code of the application. Such an approach has been shown to be well understood and adopted by modelers without programming knowledge [23].

IX. Conclusion

In this paper, we present a purely declarative approach to high-level interaction models for GUI generation in the sense that they do not include any procedural construct that would directly define sequences, iterations etc. These models are inspired by and based on theories of human communication, but they have to be precise and complete with regard to logic constraints. We also show, in this paper, that and how such models can be operationalized in the sense that behavioral GUI models can be generated from them, which are then used for automated generation of GUIs.

Certain stakeholders in the interaction design process may benefit from purely declarative interaction models especially if they are not trained in procedural programming (or similar engineering skills). In addition, since declarative representations are better suited for formal verification and automated reasoning than procedural representations, our contribution facilitates the application of such advanced techniques for interaction designs. Therefore, we propose to use our declarative models, or ones yet to be defined along these lines, in interaction design.

To the best of our knowledge, this is the first approach to define and operationalize high-level purely declarative interaction models. Future work will investigate whether the advantages of declarative knowledge representation will carry over to this domain. In addition, comparisons of the use of such models by human interaction designers with the use of previously defined ones will be performed.

Acknowledgment

We would like to thank David Raneburger and Roman Popp from Vienna University of Technology for their support with using the Unified Communication Platform tools and source code.

References

[1] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A Unifying Reference Framework for multi-target user interfaces," *Interacting with Computers*, vol. 15, no. 3, pp. 289–308, 2003.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice Hall, 2003.

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

[4] R. Popp, J. Falb, E. Arnautovic, H. Kaindl, S. Kavaldjian, D. Ertl, H. Horacek, and C. 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)*, 2009.

[5] R. Popp, D. Raneburger, and H. 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*, ser. EICS '13. New York, NY, USA: ACM, 2013.

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

[7] W. C. Mann and S. Thompson, "Rhetorical Structure Theory: Toward a functional theory of text organization," in *Text*, 1988, pp. 243–281.

[8] P. Luff, N. Gilbert, and D. Frohlich, *Computers and Conversation*. Academic Press, 1990.

[9] D. Raneburger, R. Popp, H. Kaindl, J. Falb, and D. Ertl, "Automated generation of device-specific WIMP UIs: weaving of structural and behavioral models," in *EICS*, 2011, pp. 41–46.

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

[11] H. Kaindl, R. Popp, D. Raneburger, D. Ertl, J. Falb, A. Szep, and C. Bogdan, "Robot-supported cooperative work: A shared-shopping scenario," in *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*, ser. HICSS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2011.366

[12] B. Hayes-Roth, "An architecture for adaptive intelligent systems," *Artif. Intell.*, vol. 72, no. 1-2, pp. 329–365, Jan. 1995. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(94)00004-K

[13] J. E. Laird, A. Newell, and P. S. Rosenbloom, "SOAR: an architecture for general intelligence," *Artif. Intell.*, vol. 33, no. 1, pp. 1–64, Sep. 1987. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(87)90050-6

[14] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah, "Declarative and procedural goals in intelligent agent systems," in *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 2002, Conference Proceedings.

[15] P. P. da Silva, "User interface declarative models and development environments: A survey," in *Proceedings of the 7th international conference on Design, specification, and verification of interactive systems*, ser. DSV-IS'00. Springer-Verlag, 2001, pp. 207–226. [Online]. Available: http://www.springerlink.com/index/6q0n3xw31deutjac.pdf

[16] P. A. Szekely, P. N. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher, "Declarative interface models for user interface construction tools: the mastermind approach," in *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*. London, UK, UK: Chapman & Hall, Ltd., 1996, pp. 120–150. [Online]. Available: http://dl.acm.org/citation.cfm?id=645348.650690

[17] F. Paternò, C. Santoro, and L. D. Spano, "MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments," *ACM Transactions on Computer-Human Interaction*, vol. 16, no. 4, pp. 1–30, Nov. 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1614390.1614394

[18] G. Meixner, M. Seissler, and K. Breiner, "Model-Driven Useware Engineering," *Model-Driven Development of Advanced User Interfaces*, pp. 1–26, 2011. [Online]. Available: http://www.springerlink.com/index/87458MK25324Q320.pdf

[19] G. Mori, F. Paterno, and C. Santoro, "Design and development of multidevice user interfaces through multiple logical descriptions," *IEEE Transactions on Software Engineering*, vol. 30, no. 8, pp. 507–520, 8 2004.

[20] F. Paterno and E. Zini, "Applying information visualization techniques to visual representations of task models," in *TAMODIA 04*, 2004.

[21] Q. Limbourg and J. Vanderdonckt, "Comparing task models for user interface design," *The handbook of task analysis for human-computer interaction*, vol. 6, pp. 135–154, 2004.

[22] F. Kis and C. Bogdan, "Lightweight low-level query-centric user interface modeling," in *Proceedings of the 2013 46th Hawaii International Conference on System Sciences*, ser. HICSS '13. Washington, DC, USA: IEEE Computer Society, 2013.

[23] C. Bogdan and R. Mayer, "Makumba: the Role of Technology for the Sustainability of Amateur Programming Practice and Community," in *Proceedings of the fourth international conference on Communities and technologies - C&T '09*. New York, New York, USA: ACM Press, Jun. 2009, pp. 205–214. [Online]. Available: http://dl.acm.org/citation.cfm?id=1556460.1556490