Generating an Abstract User Interface from a Discourse Model Inspired by Human Communication*

Cristian Bogdan, Jürgen Falb, Hermann Kaindl, Sevan Kavaldjian,
Roman Popp, Helmut Horacek, Edin Arnautovic and Alexander Szep
Vienna University of Technology, Institute of Computer Technology
A–1040 Vienna, Austria
{bogdan, falb, kaindl, kavaldjian, popp, horacek, arnautovic, szep}@ict.tuwien.ac.at

Abstract

Programming user interfaces is hard, error-prone and expensive, but recent advances allow generating them from interaction design models. We present an approach for modeling interaction design that is inspired by human communication. Our interaction design models are discourse models, more precisely models of dialogues. They are based on theories of human communication and should, therefore, be more understandable to humans than programs implementing user interfaces. The main ingredients of our models are communicative acts *(Speech Act Theory), which are connected as* adjacency pairs *(Conversation Analysis) and via* RST relations *(Rhetorical Structure Theory). While RST provides useful means for modeling discourse in the sense of monologue, a dialogue results from connecting monologues via adjacency pairs. This paper presents a new metamodel that integrates these approaches. Based on it, we show how abstract user interfaces can be generated from such discourse models. In a nutshell, we generate finite-state machinery and employ rules devised by us to map parts of a discourse model to abstract widgets.*

1 Introduction

In previous work [6, 7], we have already been able to automatically generate usable user interfaces (UIs), even for multiple devices and for real-world applications. We generated such UIs from models, but since these models included finite-state machinery they were more in the spirit of abstract UIs rather than high-level interaction design.

*This research has been carried out in the *OntoUCP* (A Unified Communication Platform both for Machine-Machine and Human-Machine Interaction based on Ontologies) project, partially funded by the FIT-IT Program of the Austrian FFG as project number 809254/9312. We also acknowledge the (financial) support of the PSE division of Siemens AG Österreich.

More recently, we wanted to work with models that are more understandable and possibly easier to build for humans. Therefore, we studied several theories of human communication from various fields. Based on insights from some of these theories, we focus on high-level specifications of discourses in the form of models. These models specify discourses in the sense of dialogues, where monologues are embedded and connected.

From our previous work, we inherit the use of *communicative acts* (and references to domain knowledge). Communicative acts are derived from Speech Act Theory and express intentions in the communication, in the sense of desired effects on the environment.

By integrating communicative acts with some results from *Rhetorical Structure Theory* (RST) and *Conversation Analysis*, we developed a new approach to discourse modeling for interaction design [5]. Based on this approach, we defined a new discourse metamodel, which we present below. The metamodel defines what the discourse models should look like in our approach.

So, we strive for high-level modeling of discourse, including dialogues. Such a discourse model is inspired by human communication and serves as an interaction design for a traditional information system.

From such an interaction design, user interfaces for several devices are to be generated automatically. Since we knew already how to generate them from a kind of abstract UI model, we strived for generating an abstract UI from our new kind of interaction design model. We show how this can be done through applying new pre-rendering rules.

The remainder of this paper is organized in the following manner. First, we sketch some background material for making this paper self-contained. Then we present our new discourse metamodel. We show how models according to such a metamodel can be used for generating an abstract UI. Finally, we discuss our approach and compare it with related work.

2 Background

2.1 Communicative acts

Philosophers observed that human speech is also used to do something with intention — to act. Early and seminal work on speech acts was done by [18]. In this essay Searle claims that "speaking a language is performing speech acts, act such as making statements, giving commands, asking questions and so on". Such speech acts are basic units of language communication. Since speech act theory provides a formal and clean view of communication, computer scientists have found speech acts very useful for describing communication also apart from speech or natural language. To emphasize their general applicability, the notion communicative act is used in this context. Such communicative acts have been successfully used in several applications: inter-agent communication in Knowledge Query and Manipulation Language (KQML) [10] and FIPA Agent Communication Language (ACL), see `www.fipa.org`.

2.2 Rhetorical Structure Theory

Rhetorical Structure Theory (RST) [12] is a linguistic theory focusing on the function of text, widely applied to the automated generation of natural language. It describes internal 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.

2.3 Conversation Analysis

While communicative acts are useful concepts to account for intention in an isolated utterance, representing the relationship between utterances needs further theoretical means. We have found inspiration in Conversation Analysis [11] for this purpose. Conversation analysis focuses on sequences of naturally-occurring talk "turns" to detect patterns that are generally specific to human oral communication, and consequently such patterns can be regarded as familiar. In our work we make use of patterns such as "adjacency pair" and "inserted sequence", while we regard other patterns such as "noticing" and "teasing" to be too subtle for the purposes of our current work.

3 Our Discourse Metamodel

Our discourse metamodel is based on insights from these theories of human communication. The metamodel defines what the structure of discourse models should look like in our approach. We explain it here using the UML class diagram[1] in Figure 1 and through an example extracted from real systems, a simplified kid's zoo guide scenario. Figure 1 only shows the structure of our discourse models and does not reveal their semantics.

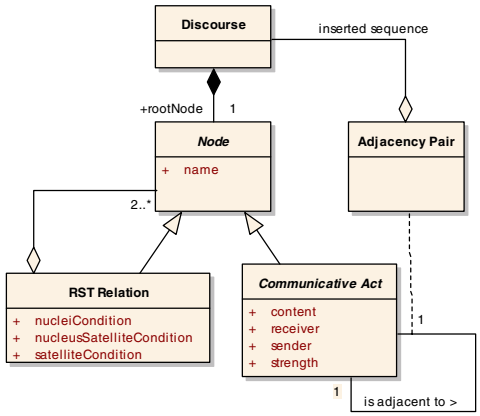

Figure 1. Sketch of our discourse metamodel.

The textual representations of the metamodel and the example model are written in OWL (Web Ontology Language, see `http://www.w3.org/2004/OWL/`), since it is well suited to describe and formalize descriptive structures, like our metamodel. In addition, OWL allows us to integrate existing domain knowledge (e.g., a zoo ontology), which the utterances are referring to, but this is beyond the scope of this paper. For better readability, we use a notation from the Univ. of Manchester in the presented examples.

The metamodel in Figure 1 consists of the following parts: communicative acts, RST relations, and their hierarchical structure. First we discuss the right part consisting of the *Communicative Act* class. The type of communicative act specifies the intention of the utterance like *requesting* information about a particular animal. Figure 2 shows a selection of three categories: *Assertions*, *Directives* and *Commissives*. Assertions convey information without requiring receivers to act beside changing their believes. Directives and commissives require an action by the receiver or sender and the advancement of the discourse by further properly composed communicative acts.

Figure 3 shows a discourse model for a zoo information system, where the utterances depicted in the rounded boxes at the bottom are cast in terms of communicative acts, e.g., the closed question for selecting an animal shown on the left

[1] At the time of this writing, the specification of UML is available at `http://www.omg.org`. More precisely, all these classes would have to be specified as *stereotypes*, but for better readability we avoid cluttering the diagram.

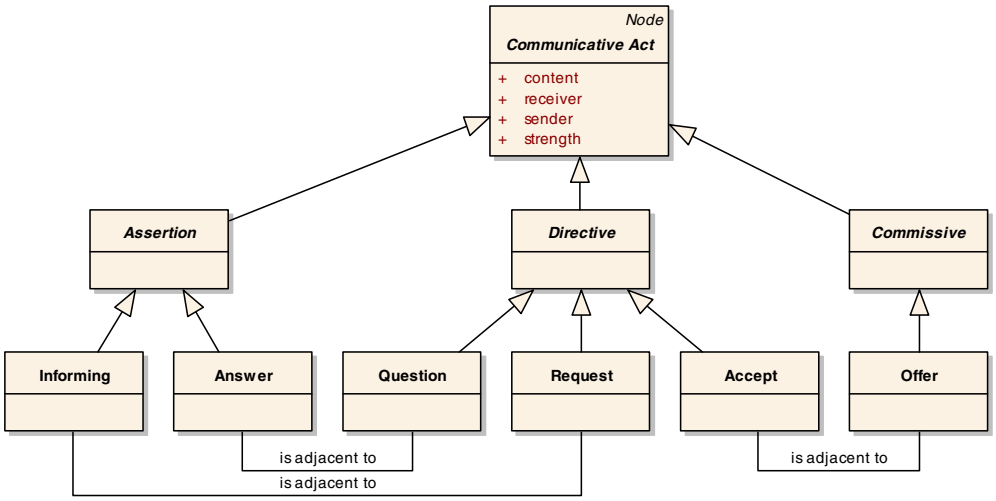

Figure 2. Selection of communicative acts connected as adjacency pairs.

of Figure 3. They can be viewed as a usage scenario for the zoo information system which advances from left to right. In fact, the scenario shown here contains some variations like optionally answering a quiz or selecting another animal.

Our representation and especially its visualization in Figure 3 show that the overall discourse is about presenting information about an animal selected by the user from a map. The information on animals is mainly achieved by the Informing communicative acts in the middle of Figure 3 and elaborated by a quiz (question-answer game on the lower-right in Figure 3) on the animal. The presentation of animal information requires that the user selects an animal through the closed question on the left of Figure 3. Almost all the other utterances are subordinate to the animal selection. This can be hierarchically continued as shown for the closed question about an animal quiz which is subordinate to the selection and the informing on the animal's name and can be repeated for each animal.

Since our discourse models are built on communicative acts, they specify the type of communicative act for each utterance. The type of communicative act carries the intent of such an utterance. E.g., Figure 3 shows *question* and their *answer* communicative acts. According to Conversation Analysis these are frequently occurring pairs of communicative acts — *adjacency pairs*. E.g., a question should have a related answer, and a request should have an accept or a reject. Typical adjacency pairs of communicative acts are modeled in our metamodel by *isAdjacentTo* relations. In our example, communicative acts which belong together are connected by black double arrows. Additionally, an adjacency pair may aggregate an *inserted sequence* (see Figure 1), required e.g., for clarification dialogues. (Further considerations of inserted sequences are out of scope of this paper.) For our running example, the question about the animal to present and its adjacent answer (left most communicative acts in Figure 3) are defined in (1) and (2):

```
Individual(askAnimal type(cdl:ClosedQuestion)
    value(cdl:content zoo.animals)
    value(cdl:strength cdl:Normal))                    (1)
    value(cdl:sender SystemA)
    value(cdl:receiver UserB)

Individual(provideAnimal type(cdl:Answer)
    value(cdl:content Elephant)
    value(cdl:strength cdl:Normal))                    (2)
    value(cdl:sender UserB)
    value(cdl:receiver SystemA)
```

The communicative acts of one communication partner are related by RST relations in a hierarchical manner. The relations are distinguished by the constraints they place on each node they relate. We use two categories of RST relations: symmetric (multi-nuclear) and asymmetric (nucleus-satellite) relations.

Multi-nuclear relations like *Joint* link discourse subtrees or communicative acts of the same type. In our example, the *Joint* relation links three *Informing* communicative acts representing detailed information about the selected animal. Note, that the order of these actions (presenting the information) is not specified by the *Joint* relation; that is why in this particular example also another scenario would fit in, where, e.g., first the audio information is presented and then the image is shown. On a user interface, both can be presented also in parallel to the user. This Joint relation example is defined in (3):

```
Individual(AnimalInfo type(cdl:Joint)
    value(cdl:relatesNucleus animalImageInforming)
    value(cdl:relatesNucleus animalAudioInforming)     (3)
    value(cdl:relatesNucleus animalDescriptionInforming)
```

Nucleus-satellite relations link a discourse subtree that represents the main intention (nucleus) and a discourse subtree that supports the nucleus. For example, in Figure 3

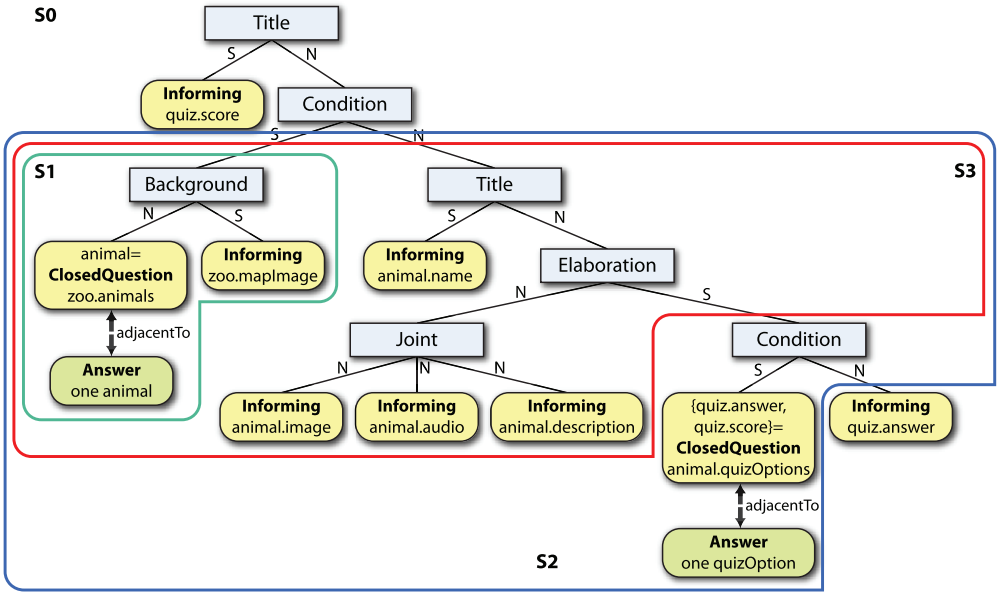

Figure 3. Example discourse model for a zoo information system.

the *Background* relation asks for selecting an animal supported by a graphical information on its location in the zoo. The *Background* relation is defined as presenting general information of any sort that is likely to help understand the nucleus. Thus, the satellite subtree of the Background relation shall only contain Informing (or in general assertive) communicative acts. According to our metamodel, the *Background* relation is defined in (4), as a subclass of the *Ideational* RST relations[2]. A *restriction* specifies that there must be only RST relations or assertive communicative acts in the satellite subtree.

```
Class(Background partial Ideational
    restriction(hasSatellite someValuesFrom(
        oneOf(RSTRelation Assertive)))
    restriction(hasSatellite cardinality(1))
    restriction(hasNucleus cardinality(1)))
```
(4)

Figure 1 shows how the hierarchical structure of RST is modeled in our metamodel, by generalizing *RST relations* and *communicative acts* to *nodes*, which can be related via the *relates* association by superordinate RST relations and thus form a tree structure. Our metamodel interpretation is more restrictive than defined by the UML class diagram, since the UML class diagram would also allow building more general graphs besides tree structures. A *discourse* consists of such a tree structure made up from related nodes. The attributes of RST relations and of communicative acts allow for modeling their usual properties.

[2]RST relations are also divided into two categories in our metamodel, *Ideational* relations which associate subtrees based on their subject and *Presentational* relations which relate subtrees only on organizational aspects of the discourse.

The upper part of Figure 3 illustrates how all these utterances conceptually belong together as a whole in our example. This structure is composed of RST relations. For an explanation, let us start at the top with the relation called *Title*. It denotes that there is an overall quiz score for the user always presented by elaborating the information contained in the right subtree. In general, we define the *Title* relation as an elaboration, restricting the additional detail of some element of subject matter to a short description, either title or caption. In the course of the utterances, one requires certain conditions to be fulfilled. E.g., informing the user about an animal requires that the system has asked for selecting an animal. Such conditions have to be fulfilled by preceding utterances, for example answering the question by selecting *one animal*. These dependencies are modeled in Figure 3 with the RST relation *Condition*. Thus, we define the *Condition* relation as a relation which requires the successful execution of the satellite subtree in order to utter the nucleus subtree. This implies an order of issuing the communicative acts in the subtrees.

The main part of talking about an animal is modeled by an *Elaboration* relation denoting that the left branch elaborates the basic animal information (image, audio and description) by asking a question about this animal followed by an information on the given answer. Our definition of the *Elaboration* relation states that the satellite subtree contains additional details (or allows gathering additional details by questioning) about some element of subject matter presented in the nucleus. The elaboration of the details can

be achieved in one of the following ways: set – member, abstraction – instance, whole – part, process – step, object – attribute, generalization – specific.

Such a tree of RST relations could be viewed as the design rationale of the utterances. Alternatively, it can be viewed as a "plan" structure of the discourse for arriving at some goal, e.g. gathering as much information as possible and a high quiz score. In this view, it is actually a non-linear plan (see the *Joint* relations in this example), while the usage scenarios are related linear plans.

4 Generating an Abstract UI

The main issue that we address in this paper is how to transfer models as exemplified in Figure 3 to user interfaces at the abstract widget level. In particular, it means a transformation from a largely declarative model to a procedural interface. Our abstract user interface is not independent of the target device, since the transformation considers the device's available space for structuring the user interface. However, our abstract user interface is completely independent of the considered UI toolkit (e.g., Java Swing or WindowsForms). From such an abstract user interface, given a target device and a UI toolkit, a concrete user interface can be generated, as there are one-to-one correspondences between abstract and concrete user interface widgets.

The general principle behind our solution approach is that the abstract user interface is made out of "presentation" units that are made visible when the logic of the interaction with the user requires so. The presentation units can, e.g., correspond to a page/form in the WWW, or a window, form or dialog box in window/widget-oriented user interface toolkits, or a "screen" in handheld user interface toolkits. In our zoo example, it makes little sense to make a presentation unit visible that shows a quiz about an animal when there is no animal selected by the user. Hence, our approach also needs to define navigation paths between presentation units. Once this principle is established, our problem can be specified as follows:

- Given a discourse tree with communicative acts as leafs, generate the possible set of presentations, and the navigation paths between these presentations. Since a presentation has to be a coherent discourse, it corresponds to a subtree of the overall discourse tree. As such, we call this problem *the tree partitioning problem*.
- Given a presentation as a discourse subtree, generate an abstract user interface based on transformation rules that obey usability guidelines. Since this effectively "pre-renders" a discourse tree into an abstract user interface, we call this problem *the pre-rendering problem*.

4.1 Addressing the Tree Partitioning Problem

4.1.1 General principles

In transforming from a discourse model to a set of user interface presentations, we have found the variables in the model to play a central role. Variables refer to elements of the domain of discourse and represent the current state and advancement in the dialogue. In the speech act theory the value of a variable has a correspondent in the notion of a speech act's "propositional content". The variables are employed in several ways in the discourse model.

First, many communicative acts need the values of some variables in order to be uttered, for example **`Informing`** `animal.name` depends on "animal" being bound (set) to an object, and that object having a "name" property. The dot in our notation is used to form a hierarchy of variables. A bound subordinate variable always requires that all variables up in the hierarchy are bound, too.

Second, some communicative acts assign a value to one or more variables (e.g., `animal=`**`ClosedQuestion`** `zoo.animals`), i.e., they bind one or more variables. Binding a variable, binds all subordinate variables, too, if the required information is available. For example, if the variable *animal* becomes bound, *animal.image*, *animal.audio* and the other subordinate variables become bound as well. If the subordinate information is not available and becomes available at a later point in time, the corresponding variable has to be bound explicitly in the course of the discourse.

Third, the objects denoted by the variables provide services that a certain communicative act may use to determine its propositional content, for example `zoo.animals` would be used by the **`ClosedQuestion`** mentioned above.

Starting from the first and third variable roles, the basic principle in the approach we take is to *present a subtree of a discourse tree only when all the needed variables are bound*, and to do the necessary rendering changes when a variable changes its value. In other words, a user interface presentation corresponds to a defined set of bound variables, and only one presentation unit can correspond to that set.

Besides helping us with binding the variables, we need also some rules defining dependencies between variables necessary for an effective rendering. Either the discourse modeler or the domain engineer needs to provide some rules about

- which variables are bound in the initial state of the dialogue,
- which variables are fixed (constant) during the dialogue,

- which variables become also bound when a certain variable is bound, and
- which variables become also unbound when a certain variable becomes unbound.

We call these rules governing the variable binding as the *binding rules*.

Another important principle that we arrived at is that *the set of user interface presentation units together with their navigation paths corresponds to a finite state machine*. In such a state machine, states are the presentation units, and a state transition corresponds to the act of binding (or changing the value of) or unbinding one or more variables resulting in a different set of bound variables. These correspond to a different user interface presentation, i.e., arriving at a different state. For this reason we use the words "state" and "presentation" interchangeably below. We make the convention that the user interface presentation corresponding to the whole tree is called S_0. The user interface presentation is illustrated in Figure 3 and shown as state together with transitions in Figure 5.

4.1.2 Partitioning example

Using the example in Figure 3, we illustrate how our principles apply to transforming the model shown there into a state machine with user interface presentations as states. Assuming that the binding rules specify the `zoo` variable as initially bound and fixed (*initiallyBound(zoo)* $\wedge$ *fixed(zoo)*), the initial presentation S_1 is depicted in Figure 3 as the largest subtree that can be instantiated when only `zoo` is bound. In detail, the service "animals" for retrieving all animals of the zoo and the zoo map are also initially available and therefore get bound, too.

It is necessary to check whether there is a transition from S_1 to itself. This would be possible if there was any communicative act in S_1 or binding rule that changes the `zoo` variable's binding or value. Since that is not the case in this example (indeed, it makes little sense to believe that the user will go to another zoo while using the application) this transition does not exist.

From inspection of S_1 we can see that the `animal` variable can be bound if the user gives a satisfying answer to the closed question. Once `animal` is bound (so both `animal` and `zoo` are bound), the presentation S_2 depicted in Figure 3 is obtained. There we see that the `Condition` relation is only partially rendered in S_2. This is a case specific to Condition relations, which informs the interface generator that the condition needs to be fulfilled for the nucleus to be shown.

It may be that the `animal` variable bound does not have the subordinate variable `quizOptions` bound (i.e., there is no quiz for that animal available). Figure 3 illustrates the state S_3 where the animal has no quiz associated with it.

In the S_3 presentation there is an `Elaboration` with just a nucleus. Even if one could think of eliminating it, the information is still useful for the user interface generator, as it suggests that in an adjacent state, more space may be needed for the quiz. This information can be also presented to the user by leaving an empty space for the missing subtree in the corresponding presentation. The space can be "filled" by binding explicitly `animal.quizOptions`, which would then lead to showing more information as part of another presentation, in our case S_2.

Since in the S_3 presentation the variable `animal` can be changed through another answer to the **`ClosedQuestion`** communicative act, there is a transition from S_3 to itself.

Finally, starting from S_2, the variable `quiz.answer` will be bound by answering a quiz. `quiz.score` is initially bound and, therefore, only its value will be changed (to e.g., compute the current score) whenever the closed question is answered by the user. So, once `quiz.answer` is bound, the presentation corresponding to the entire tree is obtained (S_0). A transition from S_0 to itself can take place with either the change of the `quiz.answer` variable, or with the change of the `animal` variable.

Also starting from either S_2 or S_3, if there is a `quiz.score` (current score) but no `animal.quizOptions`, we can get the S_4 presentation depicted in Figure 4. Since in S_4 there is no quiz to answer as `animal.quizOptions` is unbound, the only transition from S_4 to itself takes place upon a change of the `animal` variable.

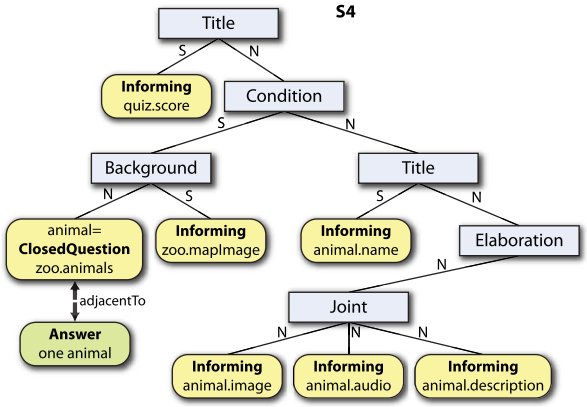

Figure 4. State S_4 of zoo discourse model with missing quiz and overall quiz score.

The state machine representing the navigation possibilities between presentations derived from the discourse tree is shown in Figure 5. It is interesting to note that if some of the binding rules would be different than the ones illustrated

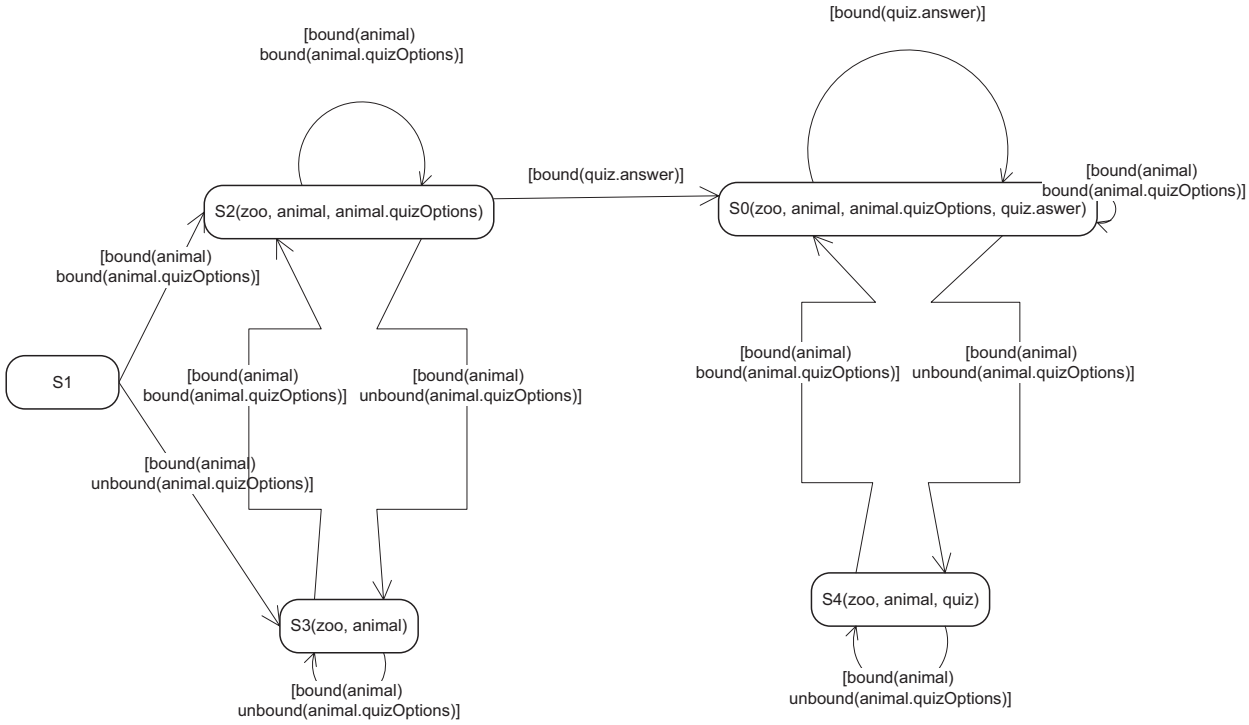

Figure 5. State machine showing possible navigation paths between presentations.

above, the state machine would look quite differently. For example, by allowing the unbinding of the `animal` variable, there would be transitions from any of S_0 and S_4 to one of S_2 and S_3.

4.1.3 Algorithm for Discourse Partitioning

Let us generalize from the above examples. Given a discourse model together with binding rules, we propose the following algorithm for partitioning the tree into presentation units and transitions between them:

1. The initial presentation S_1 corresponds to the subset of the discourse tree that results from the set of the initially bound variables V.

2. For each possible set of bound variables $\{v_1, \ldots, v_n\}$ resulting through permutation of all combinations of bound/unbound variables, check whether the set breaks any binding rules. If it does not, determine a new candidate presentation as the subset of the tree (which can be a tree or a forest) that corresponds to $\{v_1, \ldots, v_n\}$ being bound.

3. For each candidate presentation S_i except S_1, check whether it is possible to arrive at the presentation S_i via transitions starting from S_1 or from any other state that has already been linked to S_1 during step 3. If it is not possible, eliminate S_i.

4.2 Pre-rendering a Presentation

Once the presentation units have been identified, rendering of an individual one has the corresponding subtree as its input. Since this leads to an abstract UI, we call this pre-rendering here. In general, the discourse tree can be pre-rendered correctly in a large number of ways, i.e., our models per se are under-constrained for the pre-rendering. Some heuristics for rendering communicative acts are described in our previous work [7] and serve as a basis for rendering communicative acts in this work.

Our principles for pre-rendering consist of a number of general principles, followed by a number of rules or templates which we call "rendering rules".

4.2.1 General Rendering Principles

First, if there is not enough space available for both the nucleus and the satellite of an RST relation in a particular user-interface unit, the nucleus is preferred (the only exception is the Condition relation). This is inspired by the nucleus concept in RST which gives more importance to the nuclear information. As a consequence of this, a general strategy is

to first determine the "most nuclear part" of a presentation and allocate space for it first. That should correspond to the most important information for the user to see.

Second, a heuristic already illustrated deals with the neighboring states in the state machine, and finding commonalities with them, and maybe grouping them in one single user-interface unit. One can take advantage of specific RST relations used, for example an *Elaboration* with a missing satellite (S_3 in Figure 3) is an indication that something can "show up" there in a neighboring presentation. Also, a missing nucleus for a *Condition* (S_2 in Figure 3) is an indication that something of importance for the user shows up there in a neighboring presentation so it makes sense to combine the two.

Third, depending on the relation, the desirability of showing the satellite along with the nucleus varies. So, it depends on the definitions of the RST relations and other concepts. For example, it is preferable that a *Title* is shown with the content it denotes. It is also preferable that the satellite of a *Condition* is shown when it is available. It is less desirable to show the satellite of a *Background*, so it is transformed into a link when space constraints demand.

4.2.2 Rendering rules

After having introduced the general rendering principles, we introduce some rules that are specific to certain structural patterns occurring in the discourse models. We have found many such patterns during our modeling experience, and we continue to find new ones. Due to limited space, we only exemplify a few rules which we hope illustrate the principle. Every rule can be applied to a subtree of the zoo discourse model example.

Multimedia "Joint" Rule The *Joint* Rule refers to the presentation of *Informing* communicative acts of different media types, referring to the same subject (e.g., images, audio files and textual description of an animal). Figure 6 shows the specification of the Joint rule for the *Joint* relation with *Informing* communicative acts of different media types. According to this rule, all multimedia Joints will have the image on top, the text below, and the audio link to the right-bottom of the image. In our zoo example, this rule can be used to arrange the different kinds of information related to the selected animal.

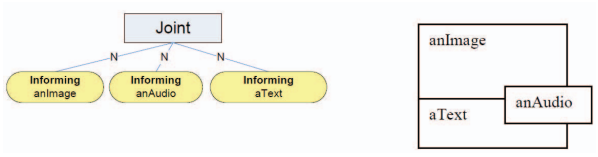

Figure 6. Multimedia Joint Rule

"ImageMap" Rule An image may serve as background information for a closed question, more precisely a *Closed Question* communicative act is associated with an *Informing* communicative act through a *Background* relation. For such a construction, we will typically display an image (satellite) where certain areas will serve as representations for possible choices (nucleus), thus forming an image map. Figure 7 shows the *ImageMap* rule transforming the Background relation. The *ImageMap* rule is an illustration of a rule that takes into account the type of media involved (image in this case). In our zoo example, this rule produces an interactive zoo map where the user can click on the animal's spot.

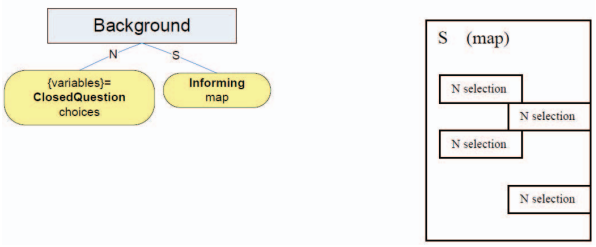

Figure 7. ImageMap Rule

"Condition" Rule The basic principle of the *Condition* Rule is that some information (denoted *N(content)* in Figure 8) can only be displayed after a question has been answered by the user and the condition becomes true. Figure 8 shows the rule for the *Condition* relation. The rendered user interface will show a presentation containing the question (satellite) first, and after the question is answered, its space will be taken by the information in the nucleus subtree ("content"). Optionally, the question can be left on display if there is enough space, to remind the user of the context in which the data is displayed. In our zoo example, this rule can be applied to the *Condition* relation that associates the animal's quiz question and its subsequent informing on the correctness of the answer by the user. In this specific example, it also makes sense to leave the question on the user interface after the user answered it.

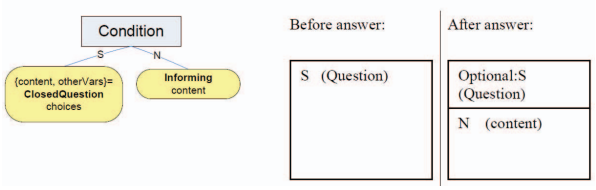

Figure 8. Condition Rule

5 Discussion

Our approach to generating abstract user interfaces from discourse models does not require programming skills for creating a user interface. Instead, it involves discourse and application domain modeling. This allows persons familiar with the problem and the domain to do the modeling work or contribute to it, potentially resulting in novel forms of end-user development.

Still, we have to address several open issues of our modeling approach. First, most of the models we work with are "optimistic", in the sense that they do not yet account for all kinds of errors, e.g., dealing with invalid requests from the user. Our current plan to address this shortcoming is to use the binding of error variables to instantiate relevant parts of the discourse tree that repair the dialogue.

Second, some discourse models allow the instantiation of large parts of the tree at the same time. This situation can make it difficult to decide what to render without cluttering the interface. For example, assume a discourse model that allows booking both, train and plane trips, and the user may engage with the plane booking part first. According to our current approach, it is hard to decide which booking part should be rendered with less detail (e.g., only providing a link). This decision can be made in the current approach only after the user started with either train or plane booking.

We are currently engaged in the evaluation of our modeling approach, assessing both the *model representation* according to our new metamodel and an already implemented *tool* that users can employ for modeling. The early results are encouraging in that most respondents understand our models well and can complete an assigned modeling task themselves (creating a discourse for booking fitness club services).

6 Related Work

Our approach to specifying discourses on a high level can be viewed as a kind of interaction design. The focus is on interactions in terms of communications rather than the structure of concrete user interfaces. Widely used approaches to interaction design build on usage scenarios [3] or abstract use cases [4]. One way to automatically generate user interfaces from use cases is presented in [13]. Interactions in this approach are presented using message sequence charts (MSC) enriched with information related to the user interface. Instead, our metamodel integrates domain information. In addition, it is on a much higher abstraction level.

Since design and implementation of user interfaces are both costly and time-consuming, and due to an increasing variety of user devices, several approaches and systems dealing with automated generation of user interfaces have been proposed. Thus let us compare our work with other work on model-based UI specification and automated UI generation.

Declarative languages are used for the description of device-independent user interfaces with abstractly defined UI structures [1, 9, 17]. All this work on languages is very important but on a much lower level of detail than ours, which has its focus on specifications of discourses from which abstract UIs are to be generated.

Browne et al. [2] use declarative descriptions of three kinds of UI models: *Presentation models* for the appearance of user interfaces in terms of their widgets, *Application models* representing which parts (functions and data) of applications are accessible from the user interface and *Dialogue models* representing end-user interactions. While this work utilizes descriptions of interactions represented by dialogue models, it does not model discourse structures as in our work.

Mobi-D is a model-based interface development environment [16]. In order to present different views of a UI, Mobi-D utilizes several declarative models at different abstraction levels. This approach is similar to ours regarding its emphasis on declarative models. However, the discourse structures used in our approach allow a higher-level specification that also facilitates automated UI synthesis, while Mobi-D addresses design support.

An advanced approach to specifying multi-device user interfaces based on task models is presented in [14]. The basic approach is to start from tasks to be supported by the application and to separately generate user interfaces for different devices according to specific device characteristics. While this approach seems to be of great help for user interface developers, it is still widget-oriented employing abstract descriptions of the UI elements as the basic abstraction. Also several of the transformations between models have to be done manually. In contrast, our approach understands user interfaces as a form of communication and involves high-level discourse structures.

Gajos and Weld [8] treat interface generation for different devices as an optimization problem, taking into account interface elements, device properties and usage patterns. The interface elements represent abstract UI widgets and are specified in terms of data exchanged between the user and the application. The abstraction level is relatively low, since the "raw" data to be shown on the screen are specified. This tool concentrates on the definition of single screens, while our approach focuses on the generation of user interface units and their possible sequences when engaging in the related discourse.

Nylander and Bylund [15] describe user-service interaction in a modality and device independent way. Their approach focuses on the data transferred, while our specifications focus on a discourse for modeling an interaction design.

7 Conclusion

In this paper, we present a new approach to modeling discourse in the sense of dialogue. It is derived from results of human communication theories, cognitive science and sociology. In this sense, our models are inspired by human communication.

We apply these models for specifying interaction design of human-computer interaction of information systems. From such a model, we can generate an abstract UI. Taking this together with our previous work on automatically generating concrete UIs [6, 7], this paves the way for automatic generation of concrete UIs from our new interaction design models.

References

[1] M. Abrams and C. Phanouriou. UIML: An XML language for building device-independent user interfaces. In *Proceedings of the XML 99*, 1999.

[2] T. Browne, D. Dávila, S. Rugaber, and K. Stirewalt. Using declarative descriptions to model user interfaces with MASTERMIND. In F. Paterno and P. Palanque, editors, *Formal Methods in Human-Computer Interaction*. Springer-Verlag, 1997.

[3] J. M. Carroll, editor. *Scenario-Based Design: Envisioning Work and Technology in System Development*. John Wiley & Sons, New York, NY, 1995.

[4] L. Constantine and L. A. D. Lockwood. *Software for Use*. ACM Press, New York, NY, 1999.

[5] 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*, pages 754–759, New York, NY, USA, 2006. ACM Press.

[6] J. Falb, R. Popp, T. Röck, H. Jelinek, E. Arnautovic, and H. Kaindl. Using communicative acts in interaction design specifications for automated synthesis of user interfaces. In *Proceedings of the 21th IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 261–264, Piscataway, NJ, USA, 2006. IEEE Computer Society Press.

[7] J. Falb, R. Popp, T. Röck, H. Jelinek, E. Arnautovic, and H. Kaindl. Fully-automatic generation of user interfaces for multiple devices from a high-level model based on communicative acts. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS-40)*, Piscataway, NJ, USA, Jan 2007. IEEE Computer Society Press.

[8] K. Gajos and D. S. Weld. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interface (IUI '04)*, pages 93–100, New York, NY, USA, 2004. ACM Press.

[9] X. W. Group. XForms — The next generation of Web forms. http://www.w3.org/MarkUp/Forms/, 2005.

[10] Y. Labrou and T. Finin. Semantics and conversations for an agent communication language. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 584–591, 1997.

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

[12] W. C. Mann and S. Thompson. Rhetorical Structure Theory: Toward a functional theory of text organization. In *Text*, pages 243–281, 1988.

[13] A. Martínez, H. Estrada, J. Sánchez, and O. Pastor. From early requirements to user interface prototyping: A methodological approach. In *Proceedings of the 17th IEEE/ACM International Conference on Automated Software Engineering (ASE'02)*, page 257, Washington, DC, USA, 2002. IEEE Computer Society.

[14] G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering*, 30(8):507–520, 8 2004.

[15] S. Nylander and M. Bylund. The Ubiquitous Interactor: Universal access to mobile services. In *Proceedings of the 10th International Conference on Human-Computer Interaction (HCII'03)*, 2003.

[16] A. Puerta. A model-based interface development environment. *IEEE Software*, 14(4):40–47, 1997.

[17] A. Puerta and J. Eisenstein. XIML: A common representation for interaction data. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI'02)*, pages 214–215, New York, NY, USA, 2002. ACM Press.

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