Policies and Procedures for Spoken Dialogue Systems

Matthias Denecke
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA, 15213
denecke@cs.cmu.edu

Abstract

In this paper, we present a framework for task oriented dialogue systems that separates as much as possible the different concerns in the design of the system. Specifically, we address how policies that control the form of interaction can be specified independently of domain or language specific specifications. We illustrate the application of policies by way of specifying three confirmation policies and apply them to four different dialogue systems.

1 Introduction

In spoken dialogue systems, there are often several ways to determine the users' intention. The system may alter the sequence of questions and database accesses to implement a dialogue strategy that fulfills certain criteria, such as imposing minimal burden on the user, or minimizing complexity of the expected answers.

In this paper, we present a framework that allows the system designer to specify and vary criteria according to which the actions of the dialogue system are determined. We proceed as follows. In section 2, we introduce a formal application description. The application description contains all the domain and language specific information a generic dialogue manager needs to process dialogues. In section 3, we propose a way to annotate the elements of the application description with certain properties. Policies then determine which of the applicable actions can be selected. To demonstrate the feasibility of our approach, we augmented four existing dialogue applications (that were designed previously without property and policy specifications) by adding policies and were able to influence the interactions of the system with the user "from the outside".

2 The Framework

In the following, we restrict ourselves to task oriented dialogue systems for which the following working hypothesis is true:

(i) The system needs to determine which task the user would like the system to perform,

(ii) the dialogue system needs to determine all parameters that are necessary for that task to be performed, and

(iii) once the information described in (i) and (ii) has been established, the dialogue system passes control to the back end application that executes the task.

The purpose of this hypothesis is to define the responsibilities of the dialogue system. It is used to derive a classification of dialogue state (called *abstract dialogue state*, see section 2.5). Note that step (i) and (ii) do not necessarily need to be executed in that order, but the completion of step (i) and (ii) is necessary before step (iii) can be executed. In this paper, we concern ourselves with the problem how the execution of step (i) and (ii) can be controlled by means of a specification, called *policy specification.* We show that policy specifications are orthogonal to the specifications used by the dialogue manager to achieve (i) and (ii), such as ontologies, dialogue goal specifications and so on. Policy specifications control selection of dialogue moves whenever the dialogue manager has several alternatives to implement step (i) and (ii). Furthermore, we address the question to what extent the policy specification is domain and language independent.

2.1 Representations

The representational formalism used in the framework are multidimensinal typed feature structures (Denecke and Yang, 2000), a simple generalization of typed feature structures (Carpenter, 1992). n dimensional typed feature structures are typed feature structures in which the nodes are annotated not with only one element drawn from an upper semilattice, but with an n dimensional vector, each element of which is drawn from an upper semilattice (typed feature structures are multidimensional feature structures of dimension 1). Unification and subsumption are then defined componentwise on the elements of the vectors.

This generalization is motivated by the need to annotate semantic representations with attributes relevant for dynamic selection of dialogue actions. For example, multidimensional feature structures allow us to express confidence annotation, or whether a value of a feature path has been confirmed or not. The decision which policy to select can be based (indirectly) on this information.

We would like to note, however, that the dialogue manager could in principle use any other description logic as long as that logic supports operations for subsumption, unification and determination of well-typedness.[1]

2.2 Discourse

There are two kinds of dialogue state, referred to as *discourse* (or concrete dialogue state) and *abstract dialogue state*. The concrete dialogue state is given by a tree shaped dialogue history along the lines of (Grosz and Sidner, 1986). Each node in the tree represents one turn and consists of three components. These are the text of the utterance of that turn, its semantic representation and a representation of the objects the semantic representation refers to. The representations are typed feature structures and contain domain specific concepts. In contrast, the abstract dialogue state, described in section 2.5, represents domain and language independent properties of the concrete dialogue state.

The text, semantic representation and object representation at a given turn in the dialogue t can be referred to by the variables $text(t)$, $sem(t)$ and $objs(t)$. The variable $text(t)$ refers to a text as received from the speech recognizer, whereas the variables $sem(t)$ and $objs(t)$ refer to a (possibly empty) set of descriptions that represent the semantics of the utterances and the objects, actions and properties ”in the real world” the utterance refers to. The cardinality of $sem(t)$ is greater than one in the case of ambiguous parse trees, while the cardinality of $objs(t)$ is greater than one in case there are multiply objects the utterance refers to.

2.3 Primitives

As mentioned above, the dialogue system has several degrees of freedom in selecting its next action. In the current framework, possible actions include (i) *database access* to resolve referring expressions, (ii) generation of *information seeking questions* for slot filling or *disjunctive questions* aiming at disambiguating results from a database request, (iii) reactions to *presupposition violations* for error handling, and (iv) passing control to the back end application sanctioned by dialogue goals once all relevant information has been established. Each of these *dialogue objects* can be composed of a set of primitive objects that form a canonical base both of the dialogue objects and the functionality supported by the dialogue system. The primitive objects ecapsulate functionality needed in the dialogue manager, such as access of databases, or invocation of a service in the back end application.

We proceed to define an ontology of these primitive objects. This is similar to the approach taken by (Crowley et al., 2002) who define an ontology of objects for an application in computer vision. Figure 1 illustrates how the objects are related to each other. In the following section, we will define dialogue objects in terms of the primitives.

[1]We also need to determine the compatibility between representations, but this can be done using copying and unification. We also note at this point that subsumption is a stronger relationship than compatibility; if one representation subsumes another, the representations are compatible, whereas the opposite is not true.

2.3.1 Constraints

Constraints partition the space created by the abstract dialogue states. A constraint is of the form $\sigma : v \circ c$ where σ is the sort of the constraint, v is a variable, c is a constant and $\circ$ is a relation that may or may not hold between c and v. Examples of constraints are given below in section 2.5 when the abstract dialogue state is introduced.

2.3.2 Descriptions

Descriptions partition the space created by the objects under discussion. Descriptions are of the form $\sigma : F$, where σ is the sort of the description and F is a term in the chosen description logic (typed feature structures in our case). Descriptions come in three sorts, namely *preconditions*, *postconditions* and *properties*. It is the purpose of preconditions and postconditions to make explicit a *contract* of the dialogue object: The dialogue system may choose to apply a given dialogue object only if the dialogue state fulfills the constraints in its precondition. But if it does so and the action specified by the dialogue object can be carried out successfully, then the dialogue object guarantees that the dialogue state fulfills the constraints in its postcondition after execution. On the other hand, properties are descriptions of dialogue objects that allow policies to select dialogue objects based on their characteristics.

2.3.3 Bindings

Bindings describe method invocations of the back end application. There are two different sorts, one for method invocations and one for associative retrieval (such as database requests or web searches).

2.4 Dialogue Objects

The four kinds of dialogue objects, namely *databases*, *goals*, *moves*, and *presuppositions*, each of which consists of constraints, descriptions, and bindings. We can now proceed to a formal definition of dialogue objects.

Definition 2.1 *Dialogue Object A* dialogue object *is a tuple* $DO = \langle D, C, S \rangle$, *where* $D = \{d_1, \ldots, d_m, d'_1, \ldots, d'_n\}$ *is a set of descriptions,* $C = \{c_1, \ldots, c_p\}$ *is a set of constraints, and* $S = \{s_1, \ldots, s_p\}$ *is a set of services.*

Informally, a dialogue object is not much more than a notation of an IF-THEN construct convenient for dialogue processing. It says that if all the constraints c_i are satisfied and the information in the discourse $objs(t)$ is subsumed by one of the descriptions d_j, but by none of the descriptions d'_k, then it is appropriate to invoke the services s_l. The descriptions d_j are interpreted as preconditions and the descriptions d'_k as postconditions. It is expected that the invocation of the services brings about enough information so that after integration of that information into the discourse, $objs(t+1)$ is subsumed by one of the d'_k.

Surprisingly, the generic concept of dialogue object is generic enough to describe all aspects of task oriented dialogue processing that are needed for our purposes. In the following, the different dialogue objects are described in more detail.

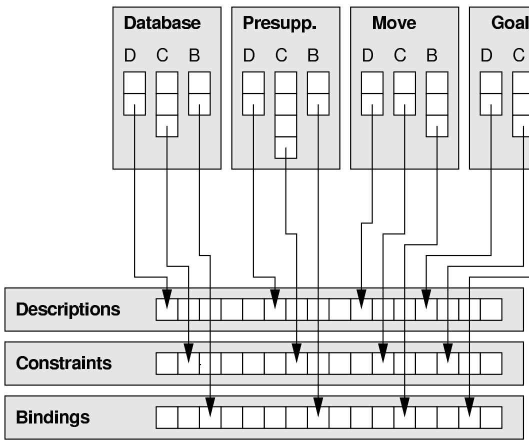

Figure 1: Ontology of objects

2.4.1 Databases

A database is a dialogue object where the preconditions among the d_i are interpreted as guards. The information in the concrete dialogue state needs to be at least as specific as one of the preconditions. Using this mechanism, it is possible to access databases only after certain information has been established in the discourse.[2] Figure 2 shows a simple database specification stating that songs can be retrieved from a database and inserted in the discourse according to the conversion equa-

[2]This is helpful to have the user of a flight information system specify, say, departure and arrival city first before a database request takes place in order to avoid having too large record sets copied in the concrete dialogue state

tions, provided that either the name or the artist of the song is given.

```
database Song {
    precondition:
        [ obj_song            ]
        [ ARTIST     string   ]
        or
        [ obj_song            ]
        [ NAME       string   ]  - >
    bindings:
        table Song {
            Name = ARTIST
            Artist = ARTIST
            Filename = FILENAME
        };
};
```

Figure 2: Database specification

2.4.2 Moves

A move is the specification of any communicative action of the dialogue system. Moves typically result in utterances processed by the text-to-speech system, but can also (alternatively or in conjunction) modify display or environment through service invocation. Dialogue moves are similar to the ones described in (Bohlin et al., 1999). A turn by the dialogue system consists of a sequence of moves in which only the last move cedes the turn to the user. Figure 3 shows an example of the dialogue move. The purpose of the postcondition is to prevent multiple instantiations of the the same actions once it has been yielded the desired result.

```
move Song {
    precondition:
        [ obj_song ]
    postcondition:
        [ obj_song            ]
        [ NAME       string   ]
    goal:(PlaySong = determined) - >
    bindings:
        say "What is the name of the song you
            would like me to play?"
};
```

Figure 3: Move specification

2.4.3 Presuppositions

Presuppositions serve to specify actions to be taken in cases where the user assumes the back end application to be in a different state than it currently is. They are useful to avoid misunderstandings and keep the dialogue on track.

2.4.4 Goals

Dialogue goals establish the link between information in the discourse and the services offered by the back end application. There is exactly one precondition for each dialogue goal. A dialogue is said to be *finalized* (i.e., the goal is reached) *iff* $objs(t)$ is subsumed by the description of exactly one goal. This implies that the descriptions for any given pair of goals need to be incompatible in order to avoid unreachable goals. [3] This condition can be verified at compile time. Figure 4 shows a simple dialogue goal specification.

```
goal PlaySong {
    precondition:
        [ act_playsong                          ]
        [       [ obj_song                ]     ]
        [ ARG   [ FILENAME     string     ]     ]
        [                                       ]
    path:($objs.[ARG | FILENAME] : num = 1) - >
    bindings:
        playSong $objs.[ARG | FILENAME];
};
```

Figure 4: A dialogue goal.[4]

2.5 Abstract Dialogue State

The purpose of the abstract dialogue state is to describe in domain and language independent terms how far the dialogue has progressed towards a goal and how the progress has been achieved. It can be seen as an ensemble of predicates that describe how the dialogue objects relate to the information that has been established in the discourse (Denecke, 2000). More specifically, the dialogue manager needs to know which of the dialogue objects can be applied to the current dialogue. This is dependent on the information established in the discourse, as represented by $sem(t)$ and $objs(t)$. Depending on the class and number of dialogue objects that stand in a certain relationship to the discourse, the dialogue manager decides on the next action.

In order to determine the abstract dialogue state, each description (as described in section 2.3.2) is

[3]If this were not the case, we had, e.g., precondition d_1 of goal 1 subsume precondition d_2 of goal 2. This implies that goal 1 is always reached before goal 2. Since a reached goal means the end of the dialogue, goal 2 can never be reached.

endowed with a state. The state of a description represents its relation with $objs(t)$. The following example illustrates this by way of dialogue goals. Consider the dialogue goals shown in figure 4 and figure 5.

```
goal PlaySong {
     precondition:
          [ act_stopsong]
- >
     bindings:
          stopSong ;
};
```

Figure 5: A second dialogue goal.

Before the dialogue starts, we have $sem(0) = objs(0) = \perp$. Assume the users' input is *"Please play me some music."*. Assuming that speech recognition and parsing succeed, the semantic representation $sem(1)$ equals $[act\_playsong]$. The description of goal 2 is incompatible with $objs(1)$, meaning that goal 2 cannot possibly represent the users' intention. The description of goal 1 is compatible with $objs(1)$ but does not subsume $objs(1)$. In order for $objs(t)$ to be subsumed by the description of goal 1, the information which song the user wishes to hear needs to be established. The dialogue system asks an information seeking question accordingly, integrates the information provided by the user, performs a database request to retrieve the name of the song and integrates that information in the discourse to form $objs(2)$. Now, we have

$$objs(2) = act\_playsong \tag{1}$$

meaning that the intention of the user has been determined uniquely, and all necessary information is in place.

A variable called *Intention* represents the following relationships between the descriptions of dialogue goals anf $objs(t)$. Possible values are $selected$ (more than one dialogue goal is compatible with $objs(t)$, $determined$ (exactly one dialogue goal is compatible with $objs(t)$, but does not subsume it), $finalized$ (exactly one dialogue goal subsumes $objs(t)$) and $inconsistent$ (no dialogue goal is compatible with $objs(t)$.

Table 1 summarizes the above example w.r.t. the values of $intention$.

Turn	0	1	2
Goal 1	compatible	compatible	subsumes
Goal 2	compatible	incompatible	incompatible
Intention	selected	determined	finalized

Table 1: Example

Note that the last line of the table does not rely on any domain specific specifications. Consequently, the variable $Intention$ allows us to formulate dialogue strategies independent of the application domain. For example, we can express the following (simplistic) dialogue strategy using constraints over the variable *Intention*: If $variable$: $(Intention = selected)$ holds, ask disambiguation question between the goals, if $variable$: $(Intention = determined)$, ask information seeking question for unfilled feature paths, if $variable$: $(Intention = finalized)$, confirm the execution of the goal, and if $variable$: $(Intention = inconsistent)$, apologize and start over. This strategy works independently if the user wants to order pizza or book a flight.

Assuming monotonically increasing specificity of $objs(t)$ over time, the value of variable $Intention$ can change only according to the diagram in figure 6 (cf. the last line in table 1).

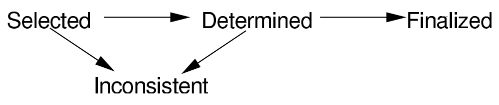

Figure 6: Possible transitions of the values of the variable *Intention*, assuming monotonic increasing representations in the discourse.

Similarly, we can abstract over the states of the databases, presuppositions and moves. However, there is a fundamental difference between goals and the other dialogue objects. Dialogue goals establish a link between the intention of the user with the services to be invoked (cf. working assumption in section 2). This in turn implies the requirement that goals be incompatible (cf. section 2.4.4). Moves, database accesses and presuppositions, however, are not subject to this requirement. In fact, in the case of databases, presuppositions and moves, we would like to have multiple objects selected as possible actions. The resulting set of possible actions can then be reranked using policy specifications. The object with the highest ranking represents the most appropriate action. In other words, the transition from the state $selected$ to the state $determined$ as shown in figure 6 is achieved through applying a policy to all selected objects in the case of databases, moves and presuppositions.

2.6 Application Specification

The application specification for the dialogue manager is the ensemble of dialgue objects, together with the ontology over which the feature structures are defined. Appendix B shows a complete example of a dialogue and the corresponding abstract dialogue states.

Definition 2.2 *Application Specification An* application specification AS *is given by a tuple* $AS = \langle O, G, M, P, D, Gr^P, Gr^G \rangle$ *where* O *is the ontology of domain specific concepts, G is the set of goals,* M *is the set of moves,P is the set of presuppositions, and D is the set of databases. In addition, parsing and generation grammars are given by* Gr^P *and* Gr^G*, respectively.*

Due to the fact that we use typed feature structures for our description language, we have the ontology O given by the type hierarchy as needed for typed feature structures, $O = \langle \mathsf{Type}, \sqsubseteq \rangle$.

3 Properties, Policies and Procedures

In the previous section, we established an ontology of dialogue object classes each of which controls one aspect of the functionality of the dialogue manager. It is our goal to control the form of the dialogue by means of a specification, called *policy specification*, external to dialogue objects. For each dialogue object, we need to be in position to determine whether its execution implements the current policy. We achieve this by endowing each dialogue object (except goals, see below) with a description of its properties (recall that descriptions of dialogue objects come in three sorts, one of which is *properties*). The properties can then be checked against the properties required by the current policy. A policy specification then induces a binary function, called *policy*, that determines if a given dialogue object adheres to the policy specification or not.

3.1 Properties

Properties describe certain aspects of dialogue objects that are relevant for their selection. For example, a question can be classified as one seeking explicit confirmation of previously established information, or one implicitly confirming what has been said while moving on to the next information to be established.

Since it is not possible to predict all properties relevant to all applications, a vocabulary over which properties are to be expressed needs to be established.[5]

3.2 Policies

Informally, a policy is a governing principle that mandates or constrains actions. The actions to be mandated or constrained, of course, are those specified by the dialogue objects. In other words, policies, mandate or constrain the application of dialogue objects to the current dialogue. Policies describe how the interaction between dialogue system and the user lead to one of the specified goals. For this reason, goals are not subject to policies as they represent the final state of some interaction; subjecting dialogue goals to policies would be alike to specifying interactions with varying goals.

Definition 3.1 *Policy Specification A* policy specification *is given by a tuple* $P = \langle C, D^G, D^M, D^P \rangle$ *where the set of constraints* $C = \{c_1, \ldots, c_m\}$ *defines the* domain *of the policy, and the sets of descriptions* $D^M = \{d_1^M, \ldots, d_n^M\}$*,* $D^P = \{d_1^P, \ldots, d_n^P\}$*, and* $D^D = \{d_1^D, \ldots, d_n^D\}$ *describe the properties of those goals, moves, presuppositions and databases, respectively, whose services are permitted to be invoked under the policy* P*.*

This leads then to the definition of policy as follows.

Definition 3.2 *Policy A* policy *implemented by a policy specification is a mapping* $p(ads, d) = do$ *from an abstract dialogue state ads and a discourse d to a dialogue object do where do is a pre-*

[5]This seems to be contradictory to the claim that properties are domain independent. However, if existing property vocabularies are found to be inappropriate for new applications, new property vocabularies (say, for a high security application), can be established. These, in turn, can be applied to existing applications. This does not always make sense, so it might be advisable to establish "classes of applications" such as internet shopping, tutorial application, high security applications, etc., each of which comes with its own property vocabulary specific to that class of application.

supposition, move or database such that its preconditions and constraints are satisfied.

3.3 Procedures

The mechanisms introduced so far allow to annotate dialogue objects with properties and to express desired properties in policies. In addition, there needs to be an instance that determines dialogue objects according to the selected policies and controls their execution. This is done by *procedures*. In the current implementation of the system, there are four different procedures. Each procedure is responsible for generating a sequence of speech acts, called *interaction pattern*, designed to update the informational content in the discourse. The procedures can generate the following interaction sequences.

1. The QUESTION interaction pattern seeks to obtain information from the user to be added to the discourse. An example is `"Would you like` *a,b* `or` *c*`?"`.

2. The UNDO interaction pattern causes to remove information from the discourse. This interaction pattern can be triggered through user utterances such as `"Undo"` or `"No, not` *a*`."`.

3. The CORRECTION interaction pattern both adds and removes information from the discourse. Examples are `"I said` *a* `not` *b*`"` (for a user initiated example) or `"I do not know` *a b* `but I do know` *a c* `or` *d b*`. Which one would you like?` for cooperative system initiated example.

4. The STATE interaction pattern does not add to nor remove information from the discourse but has influence on the dialogue flow. This interaction pattern can be triggered through user utterances such as `"I don't know"`, `"Repeat"` or `"Help"`.

Please note that interaction patterns can be instantiated in various shapes. The concrete shape of the interaction pattern is determined as the dialogue develops and depends on the abstract classification of dialogue state. For example, the dialogues `"Would you like` *a,b* `or` *c*`?" "`*a*`."` and `"Would you like` *a,b* `or` *c*`?" "`*a*`." "Please say again." "`*a*`." "Did you say` *a*`? Please say 'yes' or 'no'." "Yes."` are two instantiations of the same interaction pattern. The selection of the appropriate actions taken by the dialogue manager is determined by the policies.

4 Evaluation

We conducted an experiment to verify the characteristics of policies discussed in the previous section. The goal of the experiment is to show that, with the help of policy specifications, these applications can be extended such that

4.1 Setup

We took four application specifications developed by participants in a previous experiment (Denecke, 2002). In that experiment, the participants developed application specifications for different dialogue systems to demonstrate the system's capabilities for rapid prototyping. The domain of the application could be chosen freely by the participants, as long as it was conform with the working hypothesis (cf. section 2). The dialogue objects in these application specifications do not contain properties; nor are there policy specifications available for those applications. The dialogues generated by these application specifications do not perform any form of explicit or implict confirmation.

As these specifications were designed for a previous version of the dialogue system, we had to perform minor syntactic transformations (such as the inclusion of namespaces). However, these transformations did not alter the behavior of the specified system.

We then determined the confirmation moves to be added to the specification. We proceeded as follows: First, we had to determine those feature paths whose values can be enunciated by the user. Then, we added a move for explicit confirmation for each feature path. We also duplicated existing moves and augmented the output by statements confirming acquired information implicitly. Finally, we added moves to confirm all acquired information.

Example dialogues generated by a dialogue system without confirmation policy (as specified by one of the participants in the earlier experiment) and dialogue with three different confirmation policies are shown in appendix 5. Finally, we specified four different policies as follows:

(i) no confirmation,

(ii) explicit confirmation of everything that has been said directly after the turn,

(iii) explicit confirmation of all goal relevant information just before the services associated with the finalized goal are to be invoked,

(iv) implicit confirmation.

4.2 Property Vocabulary

As properties are expressed in typed feature structure, we need to define a signature (type hierarchy) over which the feature structures can be constructed. Since in this experiment, our concern is the selection of confirmation questions, we need to characterize the moves accordingly. The type hierarchy used is shown in figure 7. This illustration is simplified in that only the speech acts and properties are shown that are directly relevant to the discussion.

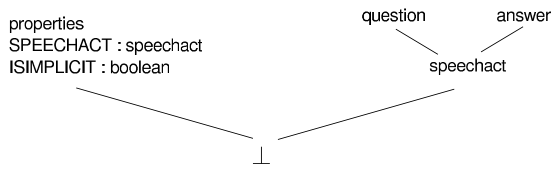

Figure 7: Vocabulary and type hierarchy used to express properties of moves.

4.3 Results

In all four cases, it was possible to ”retrofit” the existing application specifications such that each system can run using one of the four policies. This show that policies are independent of dialogue move specifications.

5 Summary

We presented a framework for spoken dialogue systems that separates the specification of a domain and language specific application specification from a policy specification. The policy specification allows to control the form of the dialogue independently from the domain.

Further work will investigate how to select policies automatically. The selection of policies could be made dependent on the abstract dialogue state. For example, a deteriorating dialogue would trigger the selection of a policy favoring moves with constrained anwers.

The presented framework would even be more powerful if combined with a real generation grammar, as opposed to the template based generation that is used in the current implementation. Then, the dialogue moves would specify a set of semantic representations as well as a set of properties the generated output needs to satisfy.

References

P. Bohlin, R. Cooper, E. Engdahl, and S. Larsson. 1999. Information states and dialogue move engines. In *IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*.

B. Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press.

J.L. Crowley, J. Coutaz, G. Rey, and P. Reignier. 2002. Perceptual Components for Context Aware Computing. In *UBICOMP 2002, International Conference on Ubiquitous Computing, Goteborg, Sweden*.

M. Denecke and J. Yang. 2000. Partial Information in Multimodal Dialogue Systems. In *Proceedings of the International Conference on Multimodal Interfaces*. Available at http://www.is.cs.cmu.edu.

M. Denecke. 2000. Informational Characterization of Dialogue States. In *Proceedings of the International Conference on Speech and Language Processing, Beijing, China*.

M. Denecke. 2002. Rapid Pprototyping for Spoken Dialogue Systems. In *Proceedings of the International Conference on Computational Linguistics, Taipei, Taiwan*.

B. Grosz and C. Sidner. 1986. Attention, intentions, and the structure of discourse. *Computational Linguistics*, pages 175–204.

Appendix A Detailed Example

Turn	sem(i)	objs(i)	dialogue object states	abstract dialogue state
	$\perp$	$\perp$	$goal1 = selected,\ goal2 = selected$ $db1 = selected,\ db2 = selected$	$Intention = selected$ $Reference = undefined$ $ReferringExpressions = undefined$
1	**User:**"Please play some music"			
	$[\ act\_playsong]$	$[\ act\_playsong]$	$goal1 = selected,\ goal2 = deselected$ $db1 = selected,\ db2 = selected$	$Intention = determine$ $Reference = undefined$ $ReferringExpressions = undefined$
	System:"What would you like to hear?"			
2	**User:**"Abba please"			
	$\begin{bmatrix} act\_playsong & \\ \text{ARTIST} & \text{"Abba"} \end{bmatrix}$	$\begin{bmatrix} act\_playsong^{(1,2)} & & \\ \text{ARTIST} & \text{"Abba"}^{(1,2)} & \\ \text{NAME} & string^{(1,2)} & \begin{cases} \text{"Waterloo"}^{(1)} \\ \text{"Super Trooper"}^{(2)} \end{cases} \\ \text{FILENAME} & string^{(1,2)} & \begin{cases} \text{"file1"}^{(1)} \\ \text{"file2"}^{(2)} \end{cases} \end{bmatrix}$	$goal1 = selected,\ goal2 = deselected$ $db1 = finalized,\ db2 = deselected$	$Intention = determine$ $Reference = finalized$ $ReferringExpressions = determine$
	System:"Would you like to hear 'Waterloo' or 'Super Trooper'?"			
3	**User:**"Waterloo"			
	$\begin{bmatrix} act\_playsong & \\ \text{ARTIST} & \text{"Abba"} \\ \text{NAME} & \text{"Waterloo"} \end{bmatrix}$	$\begin{bmatrix} act\_playsong^{(1)} & \\ \text{ARTIST} & \text{"Abba"}^{(1)} \\ \text{NAME} & \text{"Waterloo"}^{(1)} \\ \text{FILENAME} & \text{"file1"}^{(1)} \end{bmatrix}$	$goal1 = finalized,\ goal2 = deselected$ $db1 = finalized,\ db2 = deselected$	$Intention = finalized$ $Reference = finalized$ $ReferringExpressions = finalized$
	System:"I am playing 'Waterloo'."			

Table 2: Example dialogue illustrating the interactions between the different components, no policy specification given.
This example shows the interaction between concrete dialogue state, abstract dialogue state and dialogue objects in a simplistic dialogue. The application description that generates this example contains the dialogue goals, databases and moves given in the paper. In addition, there is a second database, storing unrelated information, shown here for expository reasons. The example is simplified in that not all variables of the abstract dialogue state are shown.

Appendix B Evaluation Example Dialogues

User:	Call Michael
System:	What is the first name of the person you would like to call?
User:	Michael.
System:	What is the last name of the person you would like to call?
User:	Bett.
System:	I am calling Michael Bett. Please wait.

(a) Example dialogue without confirmation policy as specified by one participant in the experiment.

User:	Call Michael
System:	What is the first name of the person you would like to call?
User:	Michael.
System:	What is Michael's last name?
User:	Bett.
System:	I am calling Michael Bett. Please wait.

(b) Example dialogue with implicit confirmation policy. Please note that all necessary concepts are confirmed.

User:	Call Michael
System:	What is the first name of the person you would like to call?
User:	Michael.
System:	What is Michael's last name?
User:	Bett.
System:	I am calling Michael Bett. Please wait.

(c) Example dialogue with implicit confirmation policy. Please note that all necessary concepts are confirmed. In one case.

User:	Call Michael
System:	You said you would like to make a call. Is this correct?
User:	Yes.
System:	What is the first name of the person you would like to call?
User:	Michael.
System:	You said you would like to call Michael. Is this correct?
User:	Yes.
System:	What is Michael's last name?
User:	Bett.
System:	You said you would like to call Michael Bett. Is this correct?
User:	Yes.
System:	I am calling Michael Bett. Please wait.

(d) Example dialogue with explicit confirmation policy, confirmation taking place after each turn.