Integrating Knowledge Sources for the Specification of a Task-Oriented Dialogue System

Matthias Denecke[1,2]
denecke@cs.cmu.edu
Interactive Systems Inc.[1]
1900 Murray Avenue
Pittsburgh PA 15217

Alex Waibel[2]
ahw@cs.cmu.edu
Interactive Systems Labs[2]
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 1521

Abstract

We show how the specification of a dialogue system can be divided into domain-dependent and domain-independent parts. We demonstrate how comparisons of actual representations in the dialogue history can help to infer hierarchical dialogue structure. The principles guiding the inference can be expressed in domain independent rules. Using typed feature structures as the only representation formalism for domain-dependent knowledge we retain the simplicity of frame-based systems in terms of gathering necessary information to fulfil a task. On the other hand, being able to easily integrate a type hierarchy into the representations and describing the systems behavior in clauses quantifying over feature structures in the dialogue history, we not only achieve a compact specification of the system's behavior, but also a type discipline that helps to detect errors in specification before system deployment. The described implementation is a first step towards the implementation of domain-independent task-oriented dialogue processing systems.

1 Introduction

In the recent past, several spoken-language dialogue applications have been implemented. In most of the cases, the implementations focus on one particular task such as Air Travel Information Service (ATIS) (see, e.g., [Ward, 1994]) or hotel reservation and travel information ([Constantinides et al, 1998]). In some cases ([Ferrieux and Sadek, 1994]), a shift towards task-independent implementations can be observed, leading to a principle-based implementation of a task-oriented dialogue system [Sadek et al., 1997], taking advantage of the *structural* similarity in task-oriented dialogues of different domains. Most of the above-cited applications have in common that they are able to perform a limited set of operations (such as hotel reservations) and that, in order to perform these operations, the user needs to specify a certain amount of information (such as arrival date). Put simply, the task of the natural language understanding component in these implementations is to determine the operation the user wants to perform, and then obtain the information necessary to perform the operation.

On the other hand, there are implementations of dialogue toolkits (see, e.g., [Sutton et al, 1996]) aiming at providing a platform to design dialogue systems without the need to take recourse on linguistic specifications. Approaching the problem of task-independent dialogue strategies from the other side, these systems typically offer an implementation of a template dialogue system bare of any task-specific knowledge at the expense of less sophisticated models of dialogue structure. When instantiating the system for a particular task, the system designer typically has to specify the flow of the dialogue, for example in form of a finite state automaton. Disadvantages of this approach are the stiff information flow following the specification and the fact that complementary information sources such as results from database requests can only be integrated with difficulties.

The work presented in this paper aims at combining advantages of the first type of system – such as natural dialogue structure – with the key advantage of the second type of system, namely easy deployment for new tasks. We assume that the behavior of a dialogue system can be sufficiently described by answering the following questions: (i) What are the entities, properties and actions the user and the system may refer to during dialogue? (ii) What kind of information is sufficient for the system in order to perform the action the user intended the system to perform? and (iii) How should the system perform the intended actions? Consequently, we are interested in separating domain-independent and domain-dependent knowledge in order to simplify as much as possible the specification for new systems. We show how the behavior of the natural language processing component in a dialogue system can be specified using declarations answering the three questions above, namely specification of a domain model, a task model and clauses describing the systems' behavior. In each instance, the

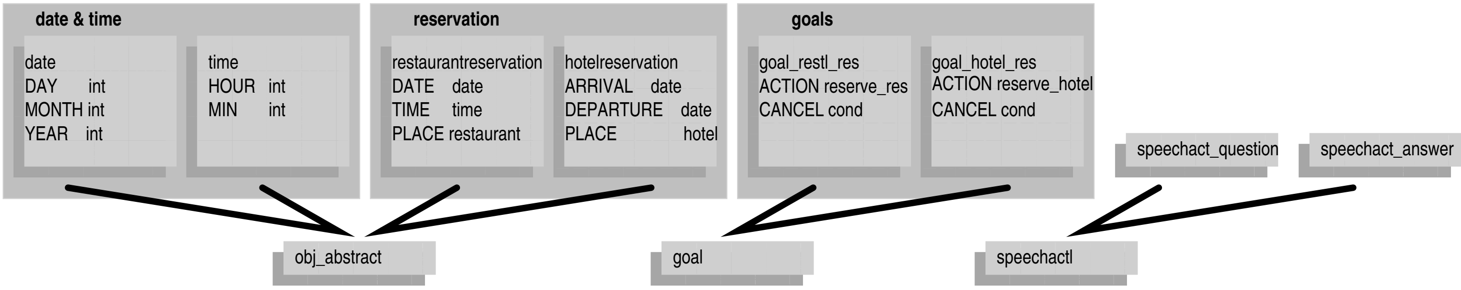

Figure 1: A part of the type hierarchy and its appropriateness conditions used in the map application. The least specific type is at the bottom of the tree. Information increases from the bottom to the top. Two sub-domain models, called date & time and reservations are merged with the application-specific declaration of the types of the goals. The part of the hierarchy declaring the speech acts is domain-independent. This is a simplified presentation of the domain model actually used in the system.

specifications consist of a set of domain-dependent and a set of domain-independent specifications.

From a processing point of view, we describe a system in which *the way of determining information to be exchanged* is domain-independent whereas *the exchanged information* itself may be domain-dependent. As a result, we arrive at a specification of a dialogue system in which domain-specific and domain-independent knowledge are orthogonal.

The system has been implemented in a travel information booth setting. Currently the system is capable of performing hotel and restaurant reservations and generating path descriptions to sites of touristic interest.

2 The Representations

2.1 The Domain-Model

We chose as the basic representation formalism throughout the system *typed feature structures* [Carpenter, 1992]. The types are ordered in a conceptual model, the *type hierarchy*, which represents domain-specific as well as domain-independent terminological knowledge using IS-A and IS-PART-OF relations. Figure 1 shows a schematic view of part of the type hierarchy we use in our interactive map application.

There are several small domain-specific sub-models for semantically closed domains. Among these are hierarchies introducing concepts of time, days and dates, or reservations, or objects that can be displayed on a map. In addition, there are domain models representing different speech acts, gestures in case of multimodal input and so on. These domain models are domain-independent. The domain model for one particular application is then combined with several domain-dependent sub-models and the domain-independent model. In addition, there is one particular type hierarchy declaring the information necessary for the application to perform the goals. The junction of all type hierarchies is subsequently referred to as the *domain model* . The domain model answer the first of the three questions, namely which are the entities, properties and actions in the domain and how do they relate to each other.

Note that since the domain model is a type hierarchy, and as such allows techniques such as inheritance, reasoning (such as reasoning based on the questions if the goal has been determined uniquely) about the nature of the goal may take place without knowing what specifically the goal is. This fact is the computational basis that allows us to express dialogue strategies in a domain-independent way, while retaining the possibility of overloading goal execution operators with domain-specific procedures.

2.2 Semantic Representations

Typed Feature Structures

We use typed feature structures [Carpenter, 1992] such as the one shown in figure 2 to represent the semantics of the users' requests. Each structure represents the semantics of a phrase of one of the main syntactic categories NP, VP, or PP. Feature structures are particularly well-suited for dialogue processing since partial information may be modelled adequately. This allows for easy integration of additional knowledge bases. As an example, consider the result of a database request filling out a partially instantiated feature structure.

Since the feature structures are typed we can use them to express anything from definite descriptions, to speech acts and intentions and goals. This allows us to perform any actions, such as unification, compatibility check or disambiguation, on representations of speech acts and intentions in the same way as we do on representations of objects.

Compact Representations

In order to implement a domain-independent dialogue-processing module, we need to be able to generate referring expressions that help us to discriminate different representations. As an example, consider two hotels carrying the same name but being located in different addresses. From a representational point of view, we

$$\begin{bmatrix} \textit{speech\_act} & \\ \text{ACTION} & \textit{show\_object} \\ \text{OBJECT} & x \end{bmatrix}$$

$$x = \begin{bmatrix} \textit{obj\_museum} & \\ \text{NAME} & \textit{" andy warhol museum "} \\ \text{ADDR} & \begin{bmatrix} \textit{address} & \\ \text{STR-NME} & \textit{" sandusky st "} \\ \text{STR-NUM} & \textit{117} \end{bmatrix} \\ & \end{bmatrix}$$

Figure 2: An example of a typed feature structure representing a request to show a museum.

are looking at a set of feature structures some of which contain common information. In order to generate a clarification question, prompting the user to select, say, one of the two hotels, the system should be able to separate similarities and differences in the representations. This is a necessary precursor for generating clarification questions in a domain-independent way.

Sets of feature structures can be represented in an underspecified representation factoring out similarities and differences in the different feature structures. For example, two feature structures of the form $[\ \theta_1\ \mathrm{F}\ \sigma_1\ ]$ and $[\ \theta_1\ \mathrm{F}\ \sigma_2\ ]$ respectively can more compactly be represented as $[\ \theta_1\ \mathrm{F}\ \sigma\{\sigma_1, \sigma_2\}]$, σ being the greatest lower bound of σ_1 and σ_2 in the type hierarchy. In addition, the types and features are annotated with indices of feature structures in order to avoid overgeneralization, being similar in spirit to named disjunctions. Figure 3 depicts a generic form for a compact representation of feature structures in which the feature F is defined. Figure 4 shows the informational content of two feature structures F and G and their common information H from an information-set perspective.

By the same token, we can determine compatible information between feature structures.[1] As a result of this operation, we obtain a representation separating compatible from incompatible information. This is helpful for example in determining which constraints specified by the user can not be fulfilled and to establish close solutions. The structures separating incompatible information are similar in structure to, yet differ in semantics with, the one shown in figure 3. In figure 4, I depicts the compatible information of the feature structures F and G.

[1] In order for the operation to yield a uniquely determined result, we n2eed to prioritize feature path equivalences. This means that in the example shown in figure 4, even in the presence of inconsistencies, path equivalences are always to be found in H. though. As it turns out, this does not impose a serious limitation in practice.

$$\begin{bmatrix} \theta \left\{ \begin{matrix} \begin{bmatrix} \theta_1 \left\{ \begin{matrix} \theta_{1,1} \\ \theta_{1,2} \\ \dots \\ \theta_{1,n_1} \end{matrix} \right\} \end{bmatrix} \\ \begin{bmatrix} \theta_2 \left\{ \begin{matrix} \theta_{2_1} \\ \theta_{2_2} \\ \dots \\ \theta_{1,n_2} \end{matrix} \right\} \end{bmatrix} \end{matrix} \right\} \\ \mathrm{F} \quad \sigma \left\{ \begin{matrix} \begin{bmatrix} \sigma_1 \left\{ \begin{matrix} \sigma_{1_1} \\ \sigma_{1_2} \\ \sigma_{1_3} \end{matrix} \right\} \end{bmatrix} \\ \begin{bmatrix} \sigma_2 \left\{ \begin{matrix} \sigma_{2_1} \\ \sigma_{2_2} \\ \sigma_{2_3} \end{matrix} \right\} \end{bmatrix} \end{matrix} \right\} \end{bmatrix}$$

Figure 3: An underspecified feature structure. The types θ, θ_i, θ_{i_j} are represented in trees that preserve the subsumption relation from the type hierarchy. Types and features are annotated with indices referring to the feature structures that contain them in order to be able to extract the feature structures correctly from the compact representation.

In both cases, the nodes trees consist of decision trees whose elements are annotated with the indices of the original feature structures. For disambiguation, the dialogue strategy may select one or more of the decision trees according to some strategy specific criterion. The selection criteria might be to disambiguate the feature path whose value has a decision tree of maximal or minimal entropy, according to the way the question is generated (for a more detailed presentation on the generation of clarification questions, see [Denecke, 1997]). Due to the co-indexed types and features in the underspecified representation the disambiguation of one feature path typically reduces the ambiguity in other feature paths as well. The compact representation helps us to select discriminating information when generating clarification questions.

It should be noted that although the construction of the decision trees relies on domain-specific knowledge (e.g. a museum is more specific than an object in the above example) the implementation of the underspecification algorithm does not since the selection of the decision tree can be formulated in terms of entropy and specificity and constitutes thus a necessary prerequisite for domain-independent specification of dialogue strategies.

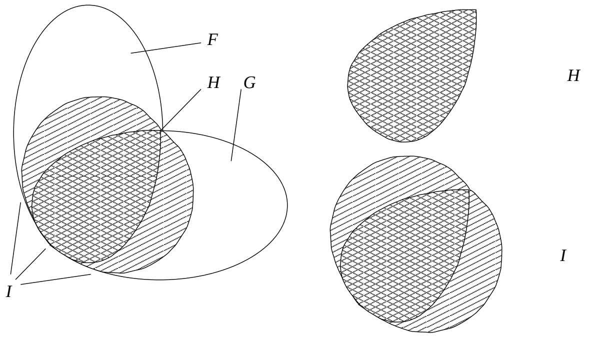

Figure 4: The information represented in two feature structures F and G

Not only may underspecified feature structures be used to represent differences and similarities of objects being ambiguously referred to, but they also serve to represent ambiguous references to goals or actions. The same clarification strategies may then be used to disambiguate between multiple objects, intentions or actions that are referred to by one description.

The Task Model

The task model consists of a set of typed feature structures, referred to as the *task descriptions*. Informally, a task description serves to specify a minimal amount of information that is necessary in order to perform a specific task, and the conditions that have to be verified in order for the execution of the task to be admissible. Consequently, each task description consists of two parts. The first part describes lower bounds on information related to the execution for the task associated with the task description. The second part describes an escape condition that has to be verified in order for the system to perform the goal. This is a reformulation of the concept of a *persistent goal* [Cohen and Levesque, 1994] in terms of feature structures. The representations of the task model only constrain the information necessary in order to perform a task; it does not describe how the task should be carried out. This is done by clauses as described in section 3.4.

Since the task model describes lower bounds on information particular to one application it is application-specific and can not be reused in general. However, only the task model describes the informational part of the tasks the dialogue system may carry out.

In case the provided information is still not specific enough to determine the intended task uniquely, an underspecified representation of all possible task representations allows to generate clarification questions to seek additional information. If, in the course of the dialogue, the acquired information is more specific than one or more of the representations in the SUBGOAL list, the action associated with the subgoals are carried out while the persistent goal remains on the goal stack. This is helpful for generating feedback to the user in the midst

$$\begin{bmatrix} \textit{goal\_reservation\_rest} & \\ \text{INFO} & \begin{bmatrix} \textit{reserve\_restaurant} & \\ \text{OBJ} & \begin{bmatrix} \textit{restaurantreservation} & \\ \text{NAME} & \textit{string} \\ \text{DATE} & \textit{date} \\ \text{TIME} & \textit{time} \end{bmatrix} \end{bmatrix} \\ \text{CANCEL} & \begin{bmatrix} \textit{reserve\_restaurant} & \\ \text{TRUTH} & \textit{false} \end{bmatrix} \\ \text{SUBGOALS} & \left\langle \begin{bmatrix} \textit{reserve\_restaurant} & \\ \text{OBJ} & \begin{bmatrix} \textit{restaurantreservation} & \\ \text{NAME} & \textit{string} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

Figure 5: Strictly lower bounds of the information necessary to perform a restaurant reservation is represented in the value of the INFO feature. The CANCEL feature represents information that will remove the goal from the stack and thus represents an escape condition. The SUBGOALS feature returns a value of lower bounds on information that, if satisfied, trigger the execution of subgoals while not changing the stack of persistent goals.

of a dialogue (see also the example in figure 10).

The task model is specific for one particular application and needs to be specified by the application designer. It answers the second of the three questions, namely which actions can the system perform and what is the information it needs to do so.

2.3 The User and the System Model

Currently, the user model simply consists of a single stack containing representations of intentions. The intentions are those inferred by the system the user wants to achieve. The user model and the system model hold representations that are inferred dynamically during dialogue processing. They are used to represent current mental states of the system and the user.

3 Relating Goals, Intentions and Structures in Discourse

In the following, we show how the specifications of domain and task model are used by the system for dialogue processing. In order to do so, we do not need to rely on any prestructured dialogue model such as dialogue grammars or finite-state automata. Moreover, we show that although determining the discourse relations may rely on domain-specific knowledge the formulation of the algorithms is domain-independent. Consequently, one particular dialogue strategy can be used in different domains. We understand by *dialogue strategy* any sequence of actions undertaken by the dialogue system in order to obtain information towards one or more goals. We understand by *goal* any amount of information sufficient for the system to perform one of the tasks it has been designed to perform. By saying '*a task is executed*' we refer to the state of the system in which *(a)* enough information for a goal to be identified uniquely and *(b)* all required information necessary to perform the actions associated with the goal have been gathered.

As long as a user is engaged with the system in a dialogue, it is then the task of the dialogue system

1. to determine if the user intends to have the system perform one of the tasks known to the system, and if so,
2. to interactively acquire all the information that is needed for the system to uniquely determine the task to be executed and all its parameters, and
3. finally to notify and pass control to the subsystem responsible for the task execution once this state has been reached.

This section describes how this strategy has been realized.

3.1 Components of the Dialogue Manager

In the current implementation, the dialogue manager has access to the components shown in figure 7.

First, there is the dialogue history. The dialogue history is a blackboard consisting of four different levels. Each level can be organized, independently of the others, as a linear list, a tree or a stack each of which hold possibly underspecified typed feature structures. The four layers correspond to orthographic, syntactic and semantic representations as well as representations of the objects (*"the world"*) the utterances refer to.

It should also be noted that the dialogue manager has access to all levels of representation at any time. As a consequence, the processing steps are not required to be in a certain predetermined order (such as parsing followed by semantic construction followed by database access followed by clarification questions or some feedback generation). Instead, conditions posed on the representations may trigger any kind of action at any time in the process, which greatly increases the flexibility of the dialogue processing.

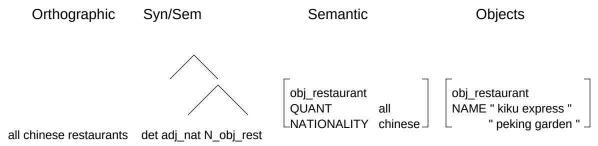

Figure 6: The four different levels of representation. At all times, the dialogue manager has access to all four levels and may pass any information on to other processing modules without being restricted by a static flow of information

Second, there are two stacks holding feature structures representing the user's and the system's intentions, respectively. The dialogue manager updates the model of the user's intentions with all instantiations of task models that are compatible with the information in the current dialogue history. The need to organize the intentional model hierarchically stems from the fact that before the user can give all the information necessary for the system to accomplish a goal (e.g. hotel reservation) the user might need the system to accomplish a sub-goal first (e.g. give complementary information on the hotel). The intentional model of the system typically holds a copy of the intentional model of the user. However, since the user's and the system's models are separated, it is possible to further restrict the intentions of the system by additional conditions so that the system would not need to do everything the user asks it to do. Hereby triggered incompatibilities between the system's and the user's intentions can also be used as conditions for additional recovery strategies.

Clearly distinguished from the intentional models is a representation of the current communicative goal. While the user's intention might be to perform a hotel reservation, the current communicative goal might be to specify the arrival date (which in turn might trigger subdialogues on its own, e.g. the user accidently referring to the 30^{th} of February).

The dialogue manager has access to the feature structures held in the different components. The dialogue manager is programmed by a set of if–then–else clauses described below.

The system architecture is client/server based, enhanced by an additional message passing scheme. The speech recognizer JANUS has been integrated in the system. The output of the speech recognizer is analyzed by the Phoenix Semantic Parser [Ward, 1994]. The resulting semantic parse tree is converted into a typed feature structure representing the semantics of the utterance. The semantic representation is added to the history which, in turn, triggers the dialogue processing described below.

3.2 Discourse Structure

We consider the performative aspect of a speech act to be limited to an update of the discourse, including models of intentions. This is in contrast with one commonly held view that speech acts are direct incarnations of actions. Continuing this line of thought, a speech act may be seen as a function mapping a context onto a context [Levinson, 1983], again the context including the mental states of the participants in the conversation. This allows us to clearly separate the actions invoked by a speech act leading to an update of the discourse and the execution of the application-specific tasks. Consequently, we have a domain-independent formulation of dialogue strategy in terms of discourse update as an advantage of our system.

The information provided by some speech act may

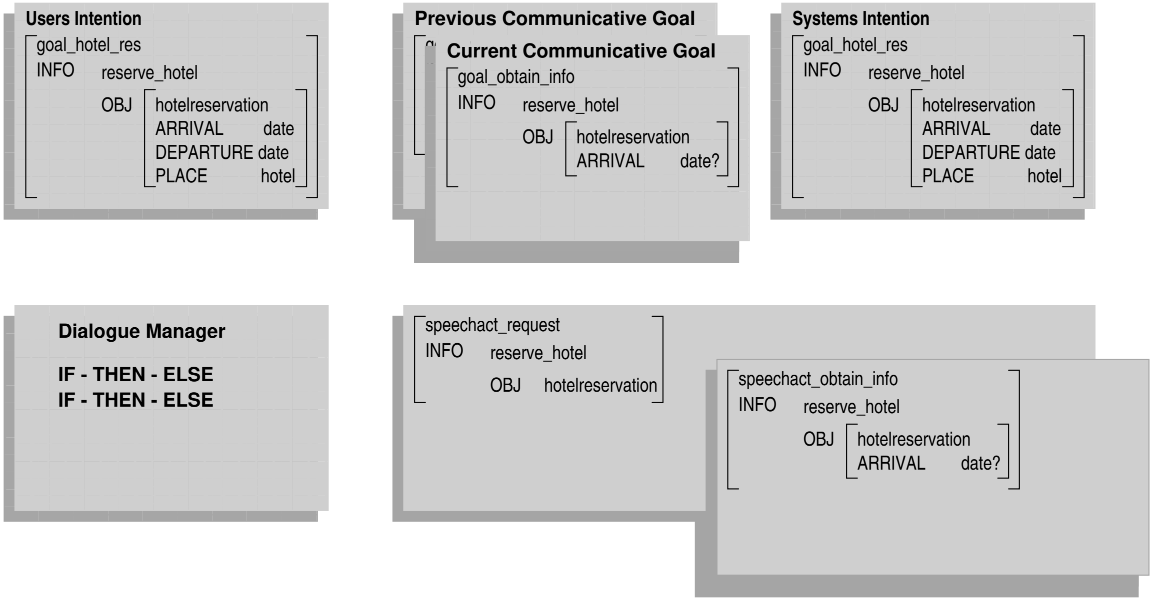

Figure 7: The overall architecture of the system. The users intention is to make a hotel reservation. The system opens a subdialogue in order to inquire the arrival date. The communicative goal of the user is temporarily replaced by the goal the generated by the system-initiated subdialogue. However, the intentions of user and system remain unchanged as the hotel reservation could not yet take place. Only the semantic layer of the four-layered blackboard is shown. The representations shown are simplifications of the representations actually used in the system.

contribute to the information available in the discourse in different ways. First of all, incoming information may be compatible with the information available in the discourse and increase the specificity towards a goal. A typical case would be the answer of a clarification question. Second, new information may be compatible with the intention of the speaker, but incompatible with the information established in the discourse. A speech act of this kind constitutes a repair. Third, information may be incompatible with the intention of the speaker and possibly the information in the discourse which indicates a subdialogue. In other words, the informational relation between the speech act and the dialogue state determines partly the way of updating the discourse. Since the representations of the speech acts are constructed by unification of feature structures in function of the parse tree, lexical information can be projected up to the speech act level in case where lexical information already constrains the type of the speech act.

The relations between information in the discourse and the intentions of the user help us to infer a hierarchical structure of discourse. Each level in the hierarchy consists of a list of possibly underspecified feature structures and references to levels below the current level. There are no predefined dialogue structures. Rather, the structure is inferred as information enters the system. If a new speech act is classified as opening a subdialogue, a new level below the current one is created. If a communicative goal is reached, the current level is closed and the level above the current one becomes the current level again.

Comparing to [Grosz and Sidner, 1986], we equate somewhat simplistically the structure of discourse with the hierarchical representation on the semantic level, while the intentional structure is expressed by the possibly underspecified representations in the intentional states of the user and the system. The focus of attention is limited to the current level of discourse and the levels accessible towards the top.

It is important to note that while the procedures to update the discourse and to infer the dialogue structure may rely on domain-specific knowledge, the formulation of the clauses does not. Instead, the discourse update may be expressed in terms of subsumption and compatibility of different representations. The specification of the discourse update remains thus domain-independent.

3.3 Reference

Due to separate levels of representation for descriptions and objects referred to ambiguous or empty reference can be modelled explicitly. Referring expressions typically trigger database requests returning variable numbers of possible referents. The referents are represented in underspecified feature structures separating similarities and differences in the representations. Figure 8 shows an ambiguously referring, a uniquely referring and an overspecified description as they are represented in the third and fourth level of the discourse history as shown in figure 6.

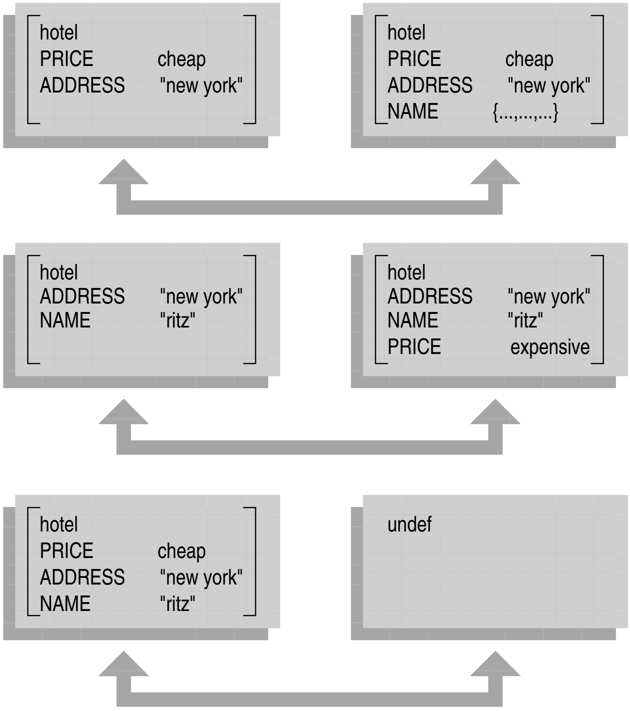

Figure 8: Different representation of descriptions and their referents. The first description refers ambiguously to a set of objects that is compactly represented in an underspecified feature structure. The second description refers uniquely to one object. The third description is overspecified; no object in the data base fits the description. The representations are depicted in a simplified fashion.

```
EVAL_GOAL :
                 isunique (top (U)) ,
                 isatomic(
                     subtract(
                         top(U)@[INFO],
                         top(S)@[INFO]))
→                evalgoal (top(S)) ,
                 pop((U)),
                 pop((S)).
```

Figure 9: The rule triggering the action associated with the task descriptions. U and S refer to the user's and the system's model of intentions, respectively. pop and top are the usual operations on stacks. If the intention of the user is uniquely determined (first condition) and all the information in the discourse meets the constraints imposed by the value of the [INFO] feature (second condition) then the goal is evaluated and the current intentions of system and user are removed from the stack.

In addition to the parallel representation of descriptions and referents, links explicitly represent the relationship between the representations. Since the links are also accessible at the clause level, it is possible to determine for each expression the number of referring objects. This helps us to identify "what the speaker intends to be picked out by a noun phrase" [Cohen and Levesque, 1994]. Consequently, it is possible to formulate dialogue strategies depending on the kind of references found in the discourse.

Moreover, it should be noted that since not only definite descriptions and objects but also actions and goals are represented in feature structures, similar conditions can be formulated on representations of intentions and actions.

3.4 Integrating Knowledge Sources

The dialogue manager is programmed by a set of if–then–else clauses [Denecke, 1997] each of which consists of three lists of predicates representing the condition, the positive conclusion and the negative conclusion, respectively. The predicates range over typed variables. The typed variables in the predicates are instantiated with feature structures held in the focus of the discourse, intentional models and communicative goal.

The clauses can be divided into domain-specific and domain-independent clauses, the former ones governing domain-specific interaction with domain-specific databases, possibly hardware and the like, while the latter ones specify the dialogue strategy. The clauses for the dialogue strategy rely on the differences in representations contained in the discourse and the intentional states and quantifications over these. These differences are then expressed in underspecified typed feature structures as shown above. Although differences are expressed using domain-specific types from the type hierarchy, the calculation of the differences and their quantification is independent from the particular domain model.

Figure 9 shows a domain-independent clause stating that if the intention of the user has been determined uniquely and all information necessary to evaluate the goal related to the user's intention could be accumulated then the goal should be evaluated and the current representations of the intentions of the user and the system respectively should be removed from the stacks.

In addition to defining rules, the language allows for the definition of procedures, and going along with this, overloading. The evalgoal predicate typically is overloaded with some application-specific definition related to the goal. The possibility to overload predicates is one of the crucial features that allows us to specify dialogue strategies in a domain-independent fashion. The overloaded predicates specify the behavior of the system when the intention of the user has been determined and sufficient information has been accumulated to make the execution of the task possible. The specification of the domain-specific rules answers the third question of how to perform the intended tasks.

The input stemming from the parser triggers appropriate clauses to fire. If the current input is the first one for a new dialogue, all compatible task descriptions are retrieved, and an underspecified feature structure representing all of them is loaded in the model of the user's intention. The first step to do now is to disambiguate the intention if it is not unique. Since database requests and processing of semantic representations can be interleaved, information query results may additionally increase the specificity of the representations thus leading to fewer clarification questions.

The application-specific clauses together with the task model and parts of the domain model are the only instances that describe the behavior of the dialogue system, meaning that a move to a new application domain would require modification of only these instances.

The described features have been implemented in a travel information booth setting. The overall turnaround time, i.e. the time between receiving the hypothesis of the speech recognizer and producing the output of the system, is typically between one and two seconds on a 200 MHz Pentium II Linux machine. The execution time depends primarily on the number of and the operations performed on the objects returned by the database requests.

4 An Example

In this section, we will illustrate the interaction of knowledge sources as specified by the clauses. We first look at some specific processing steps before providing an example of a full dialogue.

A user's request, e.g. `I would like to reserve a table` may be mapped, due to recognition errors and partial parsing, to the following partial representation

$$\begin{bmatrix} \textit{goal\_reservation} & \\ \text{INFO} & \begin{bmatrix} \textit{reserve} & \\ \text{OBJ} & \textit{reservation} \end{bmatrix} \\ & \end{bmatrix}$$

The two matching task descriptions would be the one for hotel reservation and restaurant reservation, the corresponding underspecified feature structure representing both descriptions would have the value of the path INFO set to $\textit{reserve}\ \{\textit{reserve\_hotel}, \textit{reserve\_restaurant}\}$ which would prompt a corresponding clarification question. Subsequent unification with the semantic representation of the answer `a restaurant reservation please` will disambiguate entirely the representation on top of the users' stack. Since now the intention of the user has been determined, clauses calculating the informational differences between the information required in the task description and the information available in the discourse fire to obtain complementary information. In this case, the system will prompt for the arrival date. The communicative goal of this action is to obtain the specified information, consequently, a representation of the goal is pushed onto the stack G and a semantic representation containing the propositional content of the question is generated. This leads to the situation depicted in figure 7. The user's answer `the day after tomorrow` generates a semantic representation of the form $[\textit{date}\text{REL\_DAY} + 2]$ which, in turn, will trigger a database lookup, unifying a representation of the actual date with the representation of the deictic expression. Since this information is more specific than the communicative goal, the sub dialogue is closed and the new information is integrated in the representation of the intention of the system. The requirement for the path ARRIVAL is thus fulfilled, and another path is selected in order to obtain complementary information.

Note that the user's response can also generate a sub dialogue. If for example the systems question `Which category would you prefer` is answered with `How much is the cheapest`, the incoming information is not compatible with the communicative goal and, moreover, is not a repair, so a new subdialogue is entered. in this way, hierarchical dialogue structure is inferred.

An utterance like `i don't need a reservation any more` will generate a representation of the form

$$\begin{bmatrix} \textit{goal\_reservation} & \\ \text{INFO} & \begin{bmatrix} \textit{reserve} & \\ \text{TRUTH} & \textit{false} \end{bmatrix} \\ & \end{bmatrix}$$

This representation will cause the comparison of the value of the INFO feature in the semantic representation with the value of the CANCEL feature of the currently activated task description on the stack to fail. This, in turn, will cause this task description and all descriptions on top of it to be removed.

After having examined some details of the dialogue processing, we now present an example of a full dialogue in figure 10.

5 Comparison to related Work

A variety of different approaches to dialogue processing have been proposed in the past. An approach similar to ours is the one reported in [Papineni et al, 1999]. Similarly to the goal descriptions presented here and in [Denecke, 1997], tasks are described by what is called a *form*. A form is basically a set of slot-filler pairs. Interaction between the user and the system is governed by declarative scripts that contain the specification of the task model, the message prompts and possible error messages. These scripts determine the functionality of the entire application. This approach also displays one of the key features of our approach, namely information-driven dialogue flow. However, the authors do not report on how ambiguity and inconsistency is represented. Moreover, we feel that the type discipline imposed on the

User:	Ich moechte einen Tisch reservieren (I would like to reserve a table) The first utterance contains enough information to identify the intention uniquely
System:	Wo moechten Sie reservieren? (Where would you like to make the reservation?) The system prompts for missing information that is required in the dialogue goal
User:	In dem italienischen Restaurant in der Kaiserstrasse (In the Italian restaurant on Kaiserstrasse) The user specifies constraints on the object in the discourse
System:	Ich kenne kein italienisches Restaurant in der Kaiserstrasse (I don't know any Italian places on Kaiserstrasse) There was no object in the database fulfilling the constraints specified by the user
System:	Ich kenne la Gioconda und Scala (I know la Gioconda and Scala) (The system displays icons of the restaurants on a map) However, there are two Italian places on another street. Determining maximally compatible subsets separates incompatible information. An application-specific preference valued the nationality constraint higher than the location constraint. The resulting objects were represented in an underspecified feature structure that are distinguishable by the value of the NAME feature. Since both representations are more specific than the structure specified in the SUBGOAL feature of the dialogue goal, the icons are displayed on the map.
User:	Im la Gioconda bitte. (in la Gioconda, please) The answer disambiguates the underspecified structure
System:	(The system removes one icon from the map) Fuer wann moechten Sie reservieren? (When would you like to make the reservation?) The next value in the dialogue goal specification to be queried is the date.
User:	Fuer uebermorgen um acht Uhr. (For the day after tomorrow at around eight o'clock.) The user specifies date and time. Although the answer contains more information than the system asked for the information is correctly inserted in the task description.
System:	Sie haben eine Reservierung im la Gioconda, Akademiestrasse am 20. Juni 1999 um acht Uhr. (You have a reservation in La Gioconda the 20th of June at eight pm.) All information necessary to perform the reservation has been specified, and the system confirms the reservation to the user.

Figure 10: An example for a dialogue.

representations by the type hierarchy facilitates system design since it not only allows us to detect errors in the specification during system design, but also allows for a graceful degradation should inconsistent representations combine.

Some features of our system bear similarity with features implemented in the Artimis system [Sadek et al., 1997]. These include domain-independent speech acts, the joint application of a domain-independent and a domain-dependent model and explicit representation of a persistent goal. However, the systems differ in the way information is processed. The behavior of the Artimis system is specified by a set of basic rational principles, expressed in modal logic. Principles governing communication are domain-independent, while non-communicative principles may be domain-dependent. The action to be undertaken by the dialogue system is determined by an inference process. In contrast, our system relies on less powerful logical foundations (the description logic underlying typed feature structures) and inference processes. Instead of having a theory based on rational principles, our system periodically compares available information with the information necessary to perform one of the possible goals. Consequently, a specification of a task resolves to a specification of a lower bound of information (expressed in a feature structure), together with the associated actions (expressed in a clause). Since these concepts are closer to forms and standard programming languages, a system designer may find these specifications more convenient to use than axioms in modal logic.

Compared to dialogue systems that have explicit representations of states such as finite-state-based systems, we feel that our information-centered approach leads to more flexible dialogues and potentially avoids unnecessary clarification questions. The reason is that for example database requests may be executed at any time in the processing chain and partially instantiated representations may be filled with information stemming from databases instead of having to ask the user to provide complementary information.

6 Discussion

We described a dialogue system in which domain-specific and domain-independent specifications are separated. We showed, as a prerequisite of a domain-independent dialogue strategy, how to determine the semantic content for clarification questions in a domain-independent way. We showed how the underlying dialogue strategy seeks to obtain information specific enough to select one among a set of possible tasks to fulfil and, subsequently, to obtain the information necessary to actually accomplish the task.

We demonstrated that, as a consequence of such design, it is possible to formulate discourse update and dialogue strategy in a generic way, taking advantage of informational differences in different representations. The resulting dialogue specification template is instantiated with domain models and domain-specific lists of actions in order to fulfil the tasks.

We chose to determine the speakers intention in a rather simple fashion, namely by selecting all possible goals that are compatible with the semantic content of the utterances so far. This comes at the expense of being able to deal with indirect speech acts only insomuch as the intended speech act may be inferred during semantic construction, a characteristic that stands in contrast to plan-based approaches. However, it is our hope that a more sophisticated inference procedure intended to determine the purpose of the utterance may overcome this problem by constructing semantic representations that are less closely related to the verbatim interpretation of the utterance. If and how this problem can be solved in a domain-independent way remains an open question for the time being.

References

[Carpenter, 1992] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge University Press, 1992.

[Cohen and Levesque, 1994] P.R. Cohen and H.J. Levesque. *Preliminaries to a Collaborative Model of Dialogue* Speech Communications 15 (1994), pages 265-274.

[Constantinides et al, 1998] P. Constantinides, S. Hansma, C. Tchou and A. Rudnicky. *A schema based approach to dialog control.* Proceedings of the International Conference on Spoken Language Processing, pages 409 - 412, Sidney, Australia, 1998.

[Denecke, 1997] M. Denecke. *A Programmable Multi-Blackboard Architecture for Dialogue Processing Systems.* Proceedings of the Workshop on Spoken Dialogue Processing, ACL/EACL, Madrid, Spain, 1997.

[Denecke, 1997] M. Denecke and A. Waibel. *Dialogue Strategies Guiding Users to Their Communicative Goals* Proceedings of Eurospeech, Rhodos, Greece, 1997.

[Ferrieux and Sadek, 1994] A. Ferrieux and M.D.Sadek. *An Efficient Data-Driven Model for Cooperative Spoken Dialogue* Proceedings of the International Conference on Spoken Language Processing, pages 979 - 982,Yokohama, Japan, 1994.

[Grosz and Sidner, 1986] B. J. Grosz, and C. L.Sidner. *Attention, intentions, and the structure of discourse.* Computational Linguistics, 12, 1986, pages 175-204.

[Levinson, 1983] Steven C. Levinson. *Pragmatics.* Cambridge, 1983.

[LuperFoy, 1995] Susann LuperFoy. *Implementing File Change Semantics for Spoken Language Dialogue*

Managers Proceedings of the ESCA Workshop on Spoken Dialogue Systems, pages 181 - 184, Vigso, Denmark, 1995.

[Papineni et al, 1999] K.A. Papineni, S.Roukos and R.T. Ward. *Free-Flow Dialogue Management Using Forms* Proceedings of Eurospeech 99, Budapest, Ungarn, 1999.

[Sadek et al., 1997] M.D. Sadek, Bretier, Panaget. *ARTIMIS: Natural Dialogue meets Rational Agency* Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997.

[Sutton et al, 1996] S. Sutton, D. G. Novick, R. A. Cole, and M. Fanty. *Building 10,000 spoken-dialogue systems.* Proceedings of the International Conference on Spoken Language Processing, Philadelphia, PA, October 1996.

[Ward, 1994] Wayne H. Ward. *Extracting Information in Spontaneous Speech.* Proceedings of the International Conference on Spoken Language Processing, pages 83-87, Yokohama, Japan, 1994.