A Programmable Multi-Blackboard Architecture for Dialogue Processing Systems

Matthias Denecke
Interactive Systems Laboratories
Carnegie Mellon University
Pittsburgh, PA 15217
denecke@cs.cmu.edu

Abstract

In current Natural Language Processing Systems, different components for different processing tasks and input/ output modalities have to be integrated. Once integrated, the interactions between the components have to be specified. Interactions in dialogue systems can be complex due in part to the many states the system can be in. When porting the system to another domain, parts of the integration process have to be repeated. To overcome these difficulties, we propose a multi-blackboard architecture that is controlled by a set of expert-system like rules. These rules may contain typed variables. Variables can be substituted by representations with an appropriate type stored in the blackboards. Furthermore, the representations in the blackboards allow to represent partial information and to leave disjunctions unresolved. Moreover, the conditions of the rule may depend on the specificity of the representations with which the variables are instantiated. For this reason, the interaction is information-driven. The described system has been implemented and has been integrated with the speech recognizer JANUS.

1 Introduction

When building an NLP application, several building blocks have to be integrated to form a working interactive system. Since, in the most cases, the components have been developed separately from one another, each of them has its own representations for input and output data and are optimized to achieve the task for which they have been designed, but not necessarily to optimize integrated behavior.

Partly for this reason, several blackboard and multi-agent systems have been proposed for spoken language processing in the past, one of the first being the HEARSAY system (Erman and Lesser1980). In some of these architectures, fine grained agent interaction may take place. Due to the inherent modularity, these architectures are easily extendible and reconfigurable. However, to our knowledge, little work has been focused on how the specification of this interaction may easily be adapted and extended to new tasks.

We adopt the hypothesis that a dialogue system is supposed to perform a limited set of parametrized actions and that the communicative goal of the user is to make the system perform one of these actions. Thus, we not only assume the dialogue to be task-oriented, we also assume that the behavior of the system is limited to determine which action, including its parameters, is compatible with the information conveyed by the users request and which is not. To do so, we propose to use *typed* representations that exclude the use of inappropriate information for an unintended action. Together with a type inference procedure, partially specified requests can be incrementally made more specific by using clarification dialogues.

Contrary to most multi-agent and blackboard systems in spoken language processing, we propose to control the interaction of the modules by a set of rules. The rules contain typed variables that can be instantiated with the representations stored in a discourse blackboard. The discourse blackboard stores four different levels of linguistic representations. These are orthographic representations (n best and word n best lists), syntactic/lexical semantic representations (parse trees generated by a semantic parser), the semantic representations of the utterances, and representations of the objects referring expressions may refer to. The different modules may make use of each level of representation to perform the action they implement. The advantage of representing the interaction between the modules in a set of rules are twofold. First, the rules are just another parameter that may easily be changed or adapted if the system is supposed to be ported to another domain. Second, since the variables in the rules are substituted with representations stored in the discourse history, the approach is information-driven and may take fully into account the specificity of the information entered by the user.

The system has been implemented for a map-based application in which it is possible to ask for locations and path descriptions and to make hotel and restaurant reservations. The system has been integrated with the speech recognizer JANUS (Waibel1996). To illustrate the portability, the system has been ported to a new and independent domain, a system with which fast food can be ordered.

2 Information-centered Representations

2.1 The Type Hierarchy

We use typed feature structures as defined in (Carpenter1992) throughout the entire system as representation formalism. The notion of a type in a feature structure refers to the fact that every feature structure is assigned a type from a type hierarchy. Moreover, for every type, a set of *appropriate* features is specified so that type inference is possible. In our applications, we primarily encode the domain knowledge in the type hierarchy.

According to Carpenter (Carpenter1992), the type hierarchy of the respective domain is given by a set Type of *types* and the ordering relation between types $\sqsubseteq$, the *subsumption relation.* Additionally, we describe which features from a set Feat a type may consist of by so-called *appropriateness conditions* (Carpenter1992). The type hierarchy allows us to express the IS-A relations (in the following noted in cursive letters) and IS-PART-OF relations (noted in capital letters) that hold between objects. Figure 1 shows a part of the domain that we use in our map application.

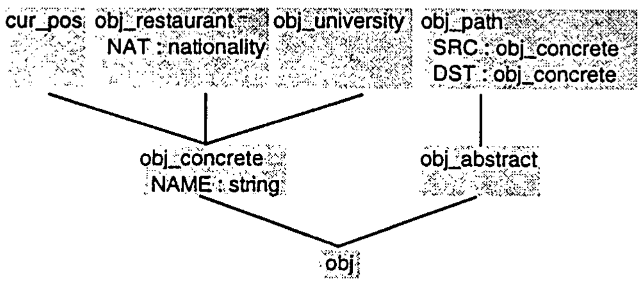

Figure 1: A part of the type hierarchy and its appropriateness conditions used in the map application. The least specific type is at the bottom of the tree.

The information in the type hierarchy not only provides the types for the feature structures and defines the relations between them but serves also to restrict variable substitutions in the rules described below.

2.2 The Semantic Representations

Oftentimes, requests formulated in natural language encode only partial information or are ambiguous. The representations of a natural language processing system have to account for this fact. Naturally, feature structures are well-suited for representing partial information. However, they do not adequately represent ambiguity. For this reason, *underspecified feature structures* have been developed. As feature structures, underspecified feature structures can encode partial information. In addition to feature structures, they are able to leave disjunctions unresolved. Figure 2 shows examples for a typed feature structure and an underspecified feature structure.

$$\begin{bmatrix} \text{obj\_museum} & \\ \text{TOWN} & \text{pittsburgh} \end{bmatrix}$$

(a)

$$\begin{bmatrix}
\text{obj\_museum} & \left\{ \begin{bmatrix} \text{obj\_museum(2)} & \\ \text{HREF} & \text{http://www.warhol.org/warhol/warhol.html} \end{bmatrix} \right\} \\
\text{NAME} & \text{string*} \left\{ \begin{array}{l} \text{" carnegie museum of natural history "(1)} \\ \text{" andy warhol museum "(2)} \\ \text{" fort pitt museum "(3)} \end{array} \right\} \\
\text{ADDR} & \begin{bmatrix} \text{address} & \\ \text{STREETNAME} & \text{string*} \left\{ \begin{array}{l} \text{" forbes ave" (1,3)} \\ \text{" sandusky st" (2)} \end{array} \right\} \\ \text{STREETNUMBER} & \text{int*} \left\{ \begin{array}{l} \text{4400(1)} \\ \text{117(2)} \\ \text{100(3)} \end{array} \right\} \\ \text{TOWN} & \text{pittsburgh} \end{bmatrix}
\end{bmatrix}$$

(b)

Figure 2: Examples of a typed feature structure (a) representing the semantics of the noun phrase `museums in pittsburgh` and an underspecified feature structure (b) representing objects that are compatible with (a).

In the attribute-value-matrix notation that we use to display underspecified feature structures, the type marked with an asterisk is the most specific lower bound of the types in its scope. The scope is indicated by curly brackets. The alternatives are represented inside curly brackets. Indices behind types identify the typed feature structure to which this information belongs. If there are no indices, the information belongs to all feature structures. Features that are common to only a subset of all represented feature structures are in the scope of the most specific type that is in common to that subset.

Underspecified feature structures represent sets of feature structures efficiently in that they express both the information that is common to and the information that differs between the feature structures in question. The fact that underspecified feature structures represent informational differences is used when generating clarification questions to generate uniquely referring NPs.

2.3 Generating Noun Phrases and Clarification Questions

Noun phrases containing descriptions of objects are generated by traversing the feature structure representing the object in depth-first order and mapping the features and types to strings. Since underspecified feature structures represent unresolved disjunctions, they are an adequate point of departure for generating clarification questions. Underspecified feature structures explicitly represent the differences between feature structures. To generate a clarification question to disambiguate an underspecified feature structure, a noun phrase for every disjunct is generated. The information in the noun phrase must be specific enough to reduce ambiguity in the underspecified structure. The noun phrases are then filled into a template of the form

Do you mean $\langle np_1 \rangle, \ldots, \langle np_{n-1} \rangle$ or $\langle np_n \rangle$?

The following example shows the information used for generating two clarification questions to disambiguate the structure shown in figure 2.

	Example 1	Example 2
Disjunct 1	[NAME]	[ADDR | STREETNAME] [ADDR | STREETNUM]
Disjunct 2	[NAME]	[ADDR | STREETNAME]
Disjunct 3	[NAME]	[ADDR | STREETNAME] [ADDR | STREETNUM]

(a)

```
(1)   Do you mean carnegie museum of natural
      history, andy warhol museum or fort
      pitt museum?
(2)   Do you mean the one at 4400 forbes ave,
      the one at sandusky st or the one at 100
      forbes ave?
```

(b)

Figure 3: Generating clarification questions. The paths shown in (a) single out the information that is sufficient to completely disambiguate the underspecified feature structure shown in figure 2 for any of the three disjuncts. The paths and their values are mapped to strings that are filled into a template to produce the questions shown in (b)

3 The Blackboard System

The dialogue system is implemented as a blackboard system. The system consists of multiple blackboards each of which stores a separate database. Moreover, a certain number of *agents* is linked to the system. The agents implement operations on the representations stored in the discourse blackboard in a modular way. The operations are used to formulate rules that control the interaction between blackboards and agents. The rules are evaluated by a central processing unit, the *general manager*, that passes control to the agents to evaluate their local operations.

3.1 The Agents

The task of agents is to perform operations on representations stored in blackboards. To this end, each agent disposes of a set of procedures that execute the actions. To specify the interface with the dialogue system, each agent exports a set of *signatures* containing information about the number and form of the procedures' parameters to the general manager. When the procedure assigned to a given signature is evaluated, the general manager passes the parameters on to the agent. After having executed the procedure, the agent returns status information and possible return values, if any, to the general manager which, in turn, uses this information to decide upon the control strategy.

3.2 The Blackboards

Among the blackboards, there is one distinguished blackboard, called the *discourse blackboard* that stores different levels of representations of the discourse history. Database blackboards are hooked up to the discourse blackboard to make the representations more specific.

3.2.1 The Database Blackboards

The database blackboards store a set of feature structures. The functionality of a database blackboard is to provide procedures to insert, remove, and lookup feature structures. Any database lookup will return an underspecified feature structure representing all feature structures that are compatible with the feature structure passed to the lookup procedure.

3.2.2 The Representation of Discourse

Aside from the database blackboards, there is one distinguished blackboard, the *discourse blackboard* representing the discourse history. For the time being, the discourse structure is a list of the generated representations. In order to access all levels of representations, this list is maintained for orthographic, syntactic and semantic representations as well as representations of referred objects which allow the discourse blackboard to be seen as four database blackboards in parallel. Moreover, links exist between the objects to access the different levels. This organization is similar to the *discourse pegs* (LuperFoy1995), with the main difference being that discourse pegs compile the different representations in one discourse unit while in our approach the different levels are separated and only accessible via links.

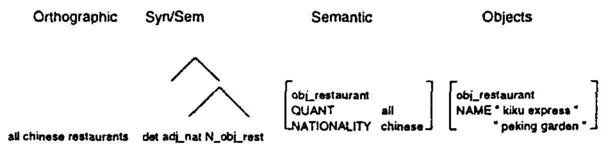

Figure 4: The four different levels of representation. Links exist between the representations in order to access the representation across the levels.

In the remainder of the paper, the level of representation may also be referred to by a number ranging from 0 (object level) to three (orthographic level). Typed feature structures in the semantic level representing noun phrases can be seen as partial descriptions of objects, each of which is compatible with the description. This allows us to determine the objects by compatibility check. In figure 2 (b), the underspecified feature structure might be the result of a database request completing the description shown in 2 (a). This is the reason why the database blackboards are attached to only one of the four levels. Every time a feature structure is added to one of the four levels, the appropriate database procedures, if any, provided by the database blackboards are executed to complete the feature structure. For instance, the object level of the discourse blackboard can be seen as database for anaphora resolution: when the representation of an anaphor is added to the semantic level of the discourse blackboard, the object level of the discourse blackboard is considered to be a standard database blackboard, and the antecedents are determined.

The natural language input is analyzed by the PHOENIX parser developed by Ward (Ward1994). The parser generates a set of parse trees each of which covers a part of the input sentence. The roots of the parse trees are specified by *top-level frames.* Top-level frames can be changed during dialogue interaction to check if the input is or is not conform to what has been expected. Words not covered by the parse trees rootes at the top-level slots are ignored. The partial semantic parse trees are converted to a semantic representation by traversing the parse tree and applying construction rules to the nodes. The construction rules operate mostly on the semantic slots in the parse trees so that synonymy and paraphrases of expressions can be handled. The semantics of an utterance is given by a set of possibly partially specified feature structures that are stored in a discourse history. Each structure represents the semantics of a phrase of one of the main syntactic categories NP, VP, or PP. Examples of how the semantic representations of a request might look like are given in figure 6 and 7.

4 The Form of Rules

The interaction of the agents and the blackboards is governed by a set of expert system style rules. The rules are composed of constants, typed variables, functions and predicates. The predicates and functions can take variables over either possibly underspecified feature structures over Type and Feat, feature paths over Feat, or events that enable the communication with the speech recognizer and other external processing modules.

4.1 Constants

A constant is given by any type from the type hierarchy. Moreover, integers and strings are considered to be subtypes of the types *string* and *int* respectively and are also treated as constants. Constants stand for atomic feature structures whose type is given by the constant name.

4.2 Variables

Variables range over feature structures and underspecified feature structures. Variables are *typed* in the sense that they impose an informational lower bound on the type of the feature structures with which they will be substituted. Names of variables ranging over feature structures have to start with the name of their type, with a capital letter to distinguish variables from constants. Feature structures substituting variables have to be stored in the discourse blackboard. Variables are indexed with the level of representation, as in

$$Obj : 0$$

Moreover, parts of the feature structures can be accessed by specifying a feature path such as

$$Obj : 0@[\textsc{position} \mid \textsc{x}]$$

The feature path has to obey the well-typed conditions as imposed by the type hierarchy.

Variables ranging over underspecified feature structures are indicated by curly brackets as in

$$\{Obj\_path\} : 0.$$

Here, too, feature path application

$$\{Obj\_path\} : 0@[\textsc{dst}].$$

is possible, the path value of an underspecified feature structure being the underspecified structure of all values of the path when applied to the feature structures represented by the underspecified feature structure.

The variables in the rules may be instantiated with representations on each of the four levels. Consequently, there is, contrary to systems that process data sequentially, no restriction that predetermines the point at which some future agent has to perform an action simly because it relies on a specific level of representation. This fact makes the architecture well-suited for repair and rescore mechanisms that integrate scores from the speech recognizer and semantic domain knowledge.

4.3 Functions

Functions as well as predicates have to be introduced by *signatures* that define informational lower bounds on the arguments (if present) and the return value. The signature for the function picturename that returns a string for any given type that is as least as specific as *obj_concrete* is given by

$$\mathsf{picturename} : obj\_concrete{+} \rightarrow string$$

where a following '+' or '-' sign indicates whether or not the argument has to be defined when evaluating the function.

4.4 Predicates

As is the case with functions, predicates are introduced by signatures. Examples are

$$\begin{array}{lcl} \mathsf{unify} & : & bot + \times bot+ \\ \mathsf{subsumes} & : & bot + \times bot+ \end{array}$$

for the unification operation and the subsumption relation on feature structures. An example for an application-specific predicate is

$$\mathsf{draw} : string + \times string + \times int + \times int+$$

whose purpose it is to draw the icon given by the second argument into the window given by the first argument at the position that is identified by the third and the fourth arguments.

4.5 Rules

Rules are formed using constants, variables, functions and predicates together with conjunction and implication connectors. They have the general form

$$\begin{array}{lll} p_1(t_{1,1},\ldots,t_{1,n_1}) & ,\ldots, & p_k(t_{k,1},\ldots,t_{k,n_k}) \quad \rightarrow \\ p_{k+1}\left(t_{k+1,1},\ldots,t_{k+1,n_{k+1}}\right) & ,\ldots, & p_l(t_{l,1},\ldots,t_{l,n_l}) \end{array}$$

where the p_i are predicates, and the t_{ij} are terms constructed over constants, variables and functions. For example, the rule displaying every object is given by

```
→ draw( "map",
        picturename(Obj_concrete : 0),
        Obj_concrete : 0@[POSITION | X],
        Obj_concrete : 0@[POSITION | Y]).
```

5 Interaction and Control

The rules are the only means to specify the interaction between agents and blackboards and between blackboards and user. Consequently, only the rules have to be modified if the system should behave differently.

5.1 Variable substitutions

If a rule contains variables, variable substitutions have to be calculated before evaluating the rule. This is done in the following manner. Let v be a variable of the form $\theta : l@\pi$. All possibly underspecified feature structures of type θ' with $\theta \sqsubseteq \theta'$ that have been added or non-monotonically modified since the last stop of the inference procedure are looked up in the discourse blackboard. If the signature of the function or the predicate requires the argument to be defined, all feature structures for which the path π is not defined are removed. From the remaining feature structures, an underspecified feature structure is generated. In the same way, the other variables in the rule are looked up. All possible combinations of instantiations form the set of substitutions. The only way to look up the data stored in the discourse blackboard is to generate variable substitutions.

5.2 Evaluating a rule

For each rule to be evaluated, the set of variable substitutions is calculated. For each substitution, the variables of the rules are instantiated and each predicate of the condition is evaluated until either one predicate fails or the condition yields true. Evaluation of a predicate or function means to pass the variable values to the procedure implementing the predicate or function and to leave control to the agent associated with the procedure. If the predicates that form the condition of the rule are verified, the remaining predicates are evaluated. If the evaluation of one of these predicates fails, the name of the failing predicate and the variable instantiations can be passed on to an error handling procedure. [1]

5.3 Evaluating a Set of Rules

The rules are evaluated using a forward chaining inference procedure. The evaluation of the program consists of the subsequent evaluation of the rules, in the order in which they are specified. After termination, all feature structures in the blackboard are marked so as to prevent re-execution of an already applied rule.

The forward-chaining inference procedure allows the system to react information-driven which means that, in essence, the information entered into the system determines which rules are evaluated. Consequently, there is no predetermined dialogue model that predicts the type or the information of the next utterance.

The set of rules forms the program that directs the interaction of the different components given the users' input. Modifying the system's behavior requires modification of the program rather than hard-coding and recompiling. This allows for rapid prototyping. To provide output functionality that can easily be adapted to new domains, the predicates also offer the possibility to call Tcl scripts.

5.4 Examples

Our first example is taken from the map application. The task of the rule shown in figure 5.4 is to completely disambiguate the representation of the destination of a path.

```
    DISAMBIGUATE :
    isambiguous({Obj_path} : 0@[DST])
→   settclvar("text1"," Do you mean")
    settclvar("text2",
          translatedifferences({Obj_path} : 0@[DST])),
    tcleval("DisplayQuestion $text1 $text2"),
    setnewtoplevelslots(gettranstoplevelslots()),
    waitforevent(EVENT_TEXTINPUT),
    tcleval("UndisplayQuestion"),
    setoldtoplevelslots(),
    add(3, %eti_text),
    iscompatible({Obj_path} : 0@[DST], parse(%eti_text)),
    unify({Obj_path} : 0@[DST], parse(%eti_text)),
    reevaluate().
```

Figure 5: The rule serving to disambiguate completely an underspecified feature structure.

The condition of the rule yields true if the semantic representation of the destination describes more than one object. If so, the remaining predicates are evaluated. In this particular case, a clarification question is generated. The predicate translatedifferences() determines the relevant feature paths

[1] The functionality is foreseen to allow interactive error recovery. If, for example, the answer to a clarification question is incompatible with the expected value, an appropriate message should be communicated to the user, along with the possibility to provide complementary information as well as to cancel the dialogue. However, in the current implementation, only the message is displayed on the screen.

and types for generating a clarification question and maps them to strings as shown in section 2.3.[2] It also determines the top level slots corresponding to the expected answers which are accessed with the predicate gettranstoplevelslots(). The question is displayed on the screen and the execution halts until some text has been entered (either via keyboard or via speech recognizer). The variable *%eti_text* is assigned to the event EVENT_TEXTINPUT and contains the entered text. The text is then added to the orthographic level of the discourse blackboard. If the semantic representation of the text is compatible with the underspecified feature structure, the representations are unified to reduce ambiguity. If the iscompatible() predicate fails, the evaluation of this rule is aborted, and other rules apply to process the text entered on the orthographic level. This allows the processing of answers that do not convey the expected information. Finally, the predicate reevaluate() forces the rule to be re-evaluated with the same substitution until the destination is disambiguated completely or an incompatible answer is given.

Note that the formulation of this rule does not make any domain-specific assumptions except that there is a type *obj_path* that carries a feature [DST]. In another application that provides functionality to order items, the same rule may apply to disambiguate the items. In order to adapt the rule, one would only have to replace $\{Obj\_path\}$: 0@[DST] with $\{Speechact\_orderobject\}$: 0@[OBJECT] where the request to order an item is represented in a feature structure subsumed by

$$\begin{bmatrix} \textit{speechact\_orderobject} \\ \text{OBJECT} \quad \textit{obj} \end{bmatrix}$$

Our next example is also taken from the map application. It demonstrates how database access and rule application interact. Suppose the user utters `show me how to get to the museum`. We assume for the sake of example that `zoom in` was recognized instead of `the museum`, and that the semantic parser skips `zoom in` . The representations that are stored on the semantic level after the input has been parsed and processed are shown in figure 6.

We consider the rules shown in figure 5.4. In this example, the second part of the first rule will be evaluated in the case of a missing the destination of the path. The rule is repeated until the feature [DST] carries a value or the user enters information that causes the unification to fail (well-typed unification). After unification, another procedure ensures that the new information is inserted correctly in the discourse blackboard. If, e.g., the user entered `the museum`,

$$\begin{bmatrix} \textit{speechact\_showpath} \\ \text{OBJECT} \;\; \boxed{1}\,\textit{obj\_path} \end{bmatrix}$$

$$\boxed{1} \quad \textit{obj\_path}$$

Figure 6: The representation on the semantic level after having processed the utterance `show me how to get to the museum` with a misrecognition on `the museum`.

$$\begin{bmatrix} \textit{speechact\_showpath} \\ \text{OBJECT} \;\; \boxed{1}\,\textit{obj\_path} \end{bmatrix}$$

$$\boxed{1} \; \begin{bmatrix} \textit{obj\_path} \\ \text{DST} \;\; \boxed{2}\,\textit{obj\_museum} \end{bmatrix}$$

$$\boxed{2} \quad \textit{obj\_museum}$$

Figure 7: The semantic representation of the request after the first question has been answered

the semantic level looks like the one shown in figure 7.

Now, since a new object has been entered on the semantic level and since there is a blackboard that provides a database access procedure for all objects that are subsumed by *obj_concrete* (the object database), a database lookup is executed. The noun phrase `the museum` does not refer uniquely to one object, as shown in figure 2, thus, an underspecified feature structure is generated on the object level. Now, the disambiguation rule explained above will initiate a clarification dialogue to disambiguate the object. Once this is achieved, the index of the intersection of the destination is stored in the path object by the following rule. The following rule copies the index of the intersection of the current position into the path object, if the source of the path is not specified. If the source of the path is specified, the index of the source intersection is calculated using rules similar to those calculating the destination index (not shown in this example for brevity). Finally, the shortest path is calculated and the result is stored in the path object as a list of line segments. Depending on the speech act type the path is an object of, there may be subsequent rules that may perform complementary operations on the data such as calculating the path length or travel time, generating a path description, or highlighting the street segments belonging to the path.

To illustrate the behavior of the rules, we show the complete dialogue:

[2] At this time, a very restricted language model is generated on the fly. Basically, it consists of all expected answers and some standard words that are always active. The next speech input is rescored using this language model. However, at the time being, there is no possibility to determine if the input corresponds to the language model or not.

```
    ADD_PATH_DST :
    isundefined(Obj_path : 0@[DST])
→   settclvar("text1","Where do you want to go today?"),
    tcleval("DisplayQuestion$text1"),
    setnewtoplevelslots(obj_concrete),
    waitforevent(EVENT_TEXTINPUT),
    tcleval("UndisplayQuestion"),
    setoldtoplevelslots(),
    set({Obj_path} : 1@[DST], parse(%eti_text)),
    reevaluate().

    DISAMBIGUATE : as above

    ADD_PATH_SRC :
    isundefined({Obj_path} : 0@[SRC])
→   set(Obj_path} : 0@[SRC], Current_position@INDEX),

    CALC_PATH :
    isunique({Obj_path} : 0@[DST | ADDR | STREETNAME]),
    isunique({Obj_path} : 0@[DST | ADDR | STREETNUMBER])
→   set(Obj_path : 0@[INDEX_DST],
      getintersection(
          {Obj_path} : 0@[DST | ADDR | STREETNAME]
          {Obj_path} : 0@[DST | ADDR | STREETNAME])),
    calcpath(Obj_path : 0@[PATHLST], Obj_path : 0@[INDEX_SRC], Obj_path : 0@[INDEX_DST]).
```

Figure 8: The rules used to calculate the shortest path

U: Show me how to get to the museum
S: Where do you want to go?
U: To the museum.
S: Do you mean carnegie museum of natural history, andy warhol museum or fort pitt museum?
U: the andy warhol museum.
S: displays path to and icon of the museum

In our next example, we consider an information system in which the user can query prices and characteristics of items, place orders, and obtain a bill for the the ordered items. We suppose a price request to be represented by a feature structure more specific than the following :

$$\begin{bmatrix} \textit{speechact\_requestprice} & \\ \text{OBJECT} & \textit{obj} \end{bmatrix}$$

Now, the description of the objects may vary in specificity which makes it refer to many different objects. The desired behavior of the system is to enumerate the prices if the description refers to few (e.g., three) objects , or to display a price range if the description refers to many objects. The rules shown in figure 9 calculate the text containing the price information.

Remember that the variable

$$\textit{Speechact\_requestprice} : 1@[\text{OBJECT}]$$

is instantiated with the semantic representation of the description as uttered by the user, the variable

$$\textit{Speechact\_requestprice} : 0@[\text{OBJECT}]$$

is instantiated with *one* object that is adequately described by the description and

$$\{\textit{Speechact\_requestprice}\} : 0@[\text{OBJECT}]$$

is instantiated with the underspecified representation of all objects fitting the description. The rules are shown in figure 9.

The condition of the first rule yields true if the description refers to more than three objects. For this reason, the system paraphrased the noun phrase conveyed by the user to refer to the objects and the minimum and maximum prices are filled in a template. If there are less than four objects, the second rule will be evaluated. Since the rule will be instantiated for each item represented in the underspecified feature structure, the prices of all objects will be appended to the text variable.

It is important to note that the system paraphrases the noun phrase it understood, using the translate predicate. In this manner, feedback can be conveyed to the user without explicitly asking a questions.

6 Discussion

We proposed a multi blackboard architecture communication mechanism between different processing modules in a dialogue system. The agents forming part of the processing modules implement a set of procedures. We proposed that a set of expert system like rules can be used to mediate the communication between different modules. The rules are formed using predicates and functions that are linked to pro-

```
    PRICE_INFO_RANGE :
    isgreaterthan(num({Speechact_requestprice}@OBJECT), 3)
→   appendtclvar("text"," The prices of"),
    appendtclvar("text", translate(Speechact_requestprice : 1@[OBJECT])),
    appendtclvar("text"," vary from"),
    appendtclvar("text", min({Speechact_requestprice}@[OBJECT | PRICE]),
    appendtclvar("text"," to"),
    appendtclvar("text", max({Speechact_requestprice}@[OBJECT | PRICE]).

    PRICE_INFO_DETAILED :
    isgreaterthan(4, num({Speechact_requestprice}@OBJECT))
→   appendtclvar("text", "The price of"),
    appendtclvar("text", translate(SPEECHACT_REQUESTPRICE:1@[OBJECT])),
    appendtclvar("text", "is"),
    appendtclvar("text", Speechact_requestprice@[OBJECT | PRICE]),
    appendtclvar("text"," dollars").
```

Figure 9: The rules generating the text to convey the prices of items. Possible values of the variable *text* after having processed the rules in the fast food application are `The prices of our pizzas range from 3.99$ to 10.99$` and `The price of the large tomato salad is 4.50$. The price of the small tomato salad is 3.50$.`

cedures formulated in imperative programming language. The purpose of the rules is twofold. One, they provide a uniform access mechanism to procedures that are implemented in a traditional imperative programming language and that are linked to the predicates. Two, they control the human-computer interaction based on the *specificity* of the available information in discourse and databases. This causes the human-computer interaction to be *information driven* rather than controlled by a dialogue model.

Our approach has several advantages. First, there is a well-defined uniform access functionality between representations in the discourse and the procedures operating on the representations. The information flow from the linguistic representations to the procedures is governed by the rules and is not hard-coded. This allows future extension of functionality. Moreover, the dialogue program can proceed dependent on the success of the operations performed. Second, the dialogue is controlled by (i) the data stored in the different levels of the discourse blackboard (including resolved database requests), (ii) the rules and (iii) the users input. No interaction is hard coded. This makes the approach information-driven. Third, since the human-computer interaction is controlled by rules, rapid prototyping of different dialogue strategies is possible by providing a different set of rules. Since agents can also function as "wrappers" around existing modules, providing uniform access to the functionality of the modules, existing modules can easily be integrated.

Acknowledgements

I would like to thank Alex Waibel, Wayne Ward and Bernhard Suhm for discussions, advice and suggestions concerning the topics discussed in the paper. Furthermore, I would like to thank Minh Tue Vo and Markus Baur for help by the implementation. Also, I would like to thank the three anonymous reviewers for their helpful comments.

References

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

L.D. Erman and V.R. Lesser. *The Hearsay-II Speech Understanding System: A Tutorial* In: Trends in Speech Recognition, A.Waibel, K.F.Lee, (eds), pages 361-381, Prentice-Hall, 1980.

Susann LuperFoy. *Implementing File Change Semantics for Spoken Language Dialogue Managers* ESCA Workshop on Spoken Dialogue Systems, pages 181 - 184, Vigso, Denmark, 1995.

Alex Waibel. Interactive Translation of Conversational Speech. *Computer*, 29(7), July 1996.

Wayne H. Ward. Extracting Information in Spontaneous Speech. *Proceedings of the International Conference on Speech and Language Processing,* 1994, Yokohama, Japan.