STAR - A MULTIPLE DOMAIN DIALOG MANAGER

Pedro Madeira, Márcio Mourão, Nuno Mamede
L2F-INESC-ID Lisboa / IST, R. Alves Redol 9, 1000-029 Lisboa, Portugal
Email: {pdsm;mdam}@mega.ist.utl.pt;nuno.mamede@inesc-id.pt

Keywords: Generic Dialog, Frames, Rules, Problem Solving Acts, Multiple Domains

Abstract: We present a dialogue manager that is able to handle multiple domains simultaneously. Our dialogue manager consists of five modules, one of which, the Task Manager, deserves special attention. Each domain is represented by a frame, which is in turn composed by slots and rules. Slots define domain data relationships, and rules define the system's behavior. The use of frames allowed the remaining modules to become domain independent. This is, beyond any doubt, a step ahead in the design of conversational systems.

1 INTRODUCTION

A dialogue system involves the integration of several modules: speech recognition, language understanding, dialog management, dialogue representation, communication with external applications, answer generation, and speech synthesis.

We decided to introduce the frame concept, to allow the representation of different domains and to make the software modules domain independent. A frame contains a set of rules and has to be defined for each domain handled by the dialog system. This approach has the advantage of making the dialog control independent of the domain.

To facilitate the understanding of our approach, we will incrementally present an example of a frame for the ticket booking domain.

2 ARCHITECTURE

Our architecture (see Figure 1) is based on the one developed at Rochester University, defined for the TRIPS project (Byron, 1999). Each component is defined and encapsulated in an autonomous agent, which interacts with the others, forming a net. A brief description of each agent's goals follows:

Interpretation Manager — interprets the user's utterances and selects the best interpretations according to the dialogue flow.

Task Manager — maintains knowledge about domain-specific tasks, and provides services to be used by the other (domain-independent) components. This agent acts has a bridge between the database and the dialogue management layers. It also allows the generic description of several domains. When working with more than one domain, the Task Manager is also responsible for managing the different frames that describe them.

Discourse Context — holds all the information needed to co-ordinate the conversational behavior of the dialog system.

Behavioral Agent — is responsible for the overall behavior of the system in problem resolution.

Generation Manager — coordinates the generation activities. It has to find the best way to express what the Behavioral Agent has decided.

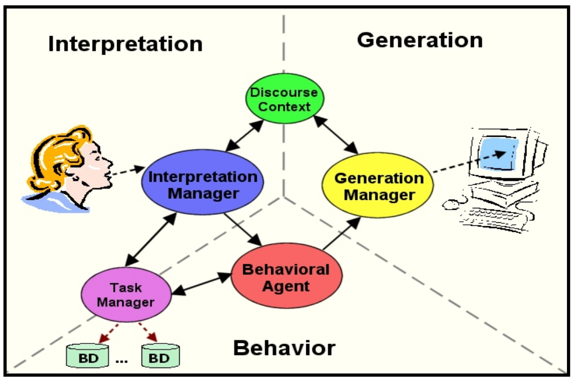

Figure 1: Architecture.

```
<DESCRIPTION>
        <FRAME_NAME>RB</FRAME_NAME>
        <DB_NAME>RB</DB_NAME>
        <SLOT key="1" type="string">Request</SLOT>
        <SLOT key="2" type="string">StartCity</SLOT>
        <SLOT key="3" type="string">DestinationCity</SLOT>
        <SLOT key="4" type="time">DepartureTime</SLOT>
        <SLOT key="5" type="time">ArrivalTime</SLOT>
</DESCRIPTION>
```

Figure 2: A frame node description for the ticket booking domain.

3 FRAMES

Faced with a multiplicity of possible dialogues, we restricted our manager to task-oriented dialogues in order to reduce the information acquisition needs. For each domain, it is necessary to define the possible actions available to the user. The dialogue manager's task is therefore to determine the action intended by the user and request the information needed to perform it. Frames are managed by the Task Manager component (see section 2).

3.1 Frame Definition

A frame is a matrix with empty spaces, named slots that have to be filled in (McTear, 1998). The information referred in a frame is maintained in a relational database. Thus, each slot represents a set of values that are kept in table registers of the corresponding database. Each empty slot can be filled with the information of one or more fields of the respective table. Therefore, each frame contains a reference to the database that holds the information about that domain.

To keep the filling of the several slots consistent, it is necessary to indicate the set of values with which a slot can be instantiated. Moreover, there can be multiple valid forms to fill several slots that compose a frame. To avoid invalid combination of values, we have defined a meta-language to express the constraints that must be satisfied. This meta-language will be presented in section 3.1.3. In short, a frame is a domain abstraction, and it includes: (i) DESCRIPTION, a description of the domain; (ii) RECOGNITION_RULES, used to specify the set of values that each slot may hold; (iii) VALIDATION_RULES, contains a set of domain restrictions, i.e., invalid combination of slot values; (iv) CLASSIFICATION_RULES, used to specify the actions that must be performed when some conditions are satisfied, i.e., the valid combinations of values.

3.1.1 Description

A description (see Figure 2) consists of the following entities: (i) <FRAME_NAME>, used to distinguish each frame from the other frames that co-exist in the dialogue manager; (ii) <DB_NAME>, the database where the domain information is maintained; (iii) <SLOT>, the map of internal registers (slots) with database fields.

Each slot has a key, an identifier represented by an integer value. The type attribute is used to characterize the values the slot can be instantiated with. Some possible types are: "string", "time", "boolean" and "number" (integer values).

3.1.2 Recognition Rules

Recognition Rules (see Figure 3) are used to determine the domain an object belongs to. They may also be used to describe the set of values that a slot may hold. Each rule is composed by: (i) <SLOT_KEY>, identifies the slot the rule should be applied to; (ii) <SQL_DOMAIN>, the SQL query that gets all the possible values for the slot.

3.1.3 Operators, Functions, Constants.

Both classification and validation rules (presented below) use operators to enhance their expressiveness.

Operators can be logical (and, or, not, equal, implication), conditional (if) or relational (ascending). There are two functions to access the contents of frame slots (SLOT_VALUE and SLOT_FILLED).

```
<RECOGNITION_RULES>
    <RECOGNITION_RULE>
          <SLOT_KEY>2</SLOT_KEY>
          <SQL_DOMAIN>SELECT Tab_City.city_id, Tab_City.description FROM Tab_City
                   WHERE Tab_City.description[$Comp]'[$Key]'
          <SQL_DOMAIN>
    </RECOGNITION_RULE>
</RECOGNITION_RULES>
```

Figure 3: Recognition rule for the slot "Start City".

The two available constants are *True* and *False*. Each operator may contain other operators and use functions or constants (see Figures 4 and 6). Logical operators have the usual semantics, and by default, the composition of operators is the conjunction.

Each rule, operator or function may contain an optional attribute, named *resultStr* that is used to generate the database query when the rule is applied.

```
<CLASSIFICATION_RULES>
   <RULE  prob='0.2187' name="BuyTicket"
        resultStr="SELECT * FROM Tab_Travels, Tab_Schedule
                     WHERE  (Tab_ Travels.StartCity[$Comp2]'[$Key2]') AND
                                        (Tab_ Travels.DestinationCity[$Comp3]'[$Key3]')  AND
                                        (Tab_ Travels.id = Tab_Schedule.travelID) ">
        <SLOT_VALUE prob=' 0.3' slotKey="1">Ticket</SLOT_VALUE>
        <AND prob='0.729'>
          <SLOT_FILLED prob='0.9'>2</SLOT_FILLED>
          <SLOT_FILLED prob='0.9'>3</SLOT_FILLED>
          <OR prob='0.9'>
                <SLOT_FILLED prob='0.9' reultStr="Tab_Schedule.DepartureTime[$Comp]'[$Key]'">
                     4</SLOT_FILLED>
                <SLOT_FILLED prob='0.9' resultStr="Tab_Schedule.ArrivalTime[$Comp]'[$Key]'">
                     5</SLOT_FILLED>
          </OR>
        </AND>
     </RULE>
</CLASSIFICATION_RULES>
```

Figure 4: Classification instantiated rule sample.

The database query is recursively assembled by concatenating the suboperators attribute *resultStr*.

3.1.3.1 SLOT_VALUE, SLOT_FILLED

The SLOT_VALUE function has as attributes a reference to a slot and an optional value. When the optional value exists, it returns true if the slot contains the value specified as argument, false otherwise. If the optional value does not exist, this function returns the actual value of the slot.

The SLOT_FILLED function has as argument a reference to a slot and returns true when the slot is filled, false otherwise. This operator has two optional arguments: *certaintyDegree*, a confidence value threshold; and *comp*, indicating how the current certainty degree should be compared with the threshold (the value specified by the certaintyDegree argument). The certaintyDegree and comp are used to model dynamic frames, an ongoing project. In these cases, the slot is only considered filled when it has an instantiated value and its certainty degree is greater, equal or lesser than a threshold value.

3.1.4 Validation Rules

Validation rules describe the combinations of slot values that are not acceptable. Figures 6 and 7 shows a validation rule of the ticket booking domain: (i) the start city and the destination city cannot be the same; and (ii) the departure time (slot 4) must be greater than the arrival time (slot 5).

3.1.5 Classification Rules

Classification rules contain the action that must be performed whenever all the information needed to perform a transaction has been gathered.

For example, in the ticket booking domain, a rule may express that the start and the destination cities are required to consult the schedules. Figure 4 shows a classification rule used to purchase a ticket.

4 FRAME USES

The Dialog Manager controls the interactions with the user to perform a task. In each interaction the user provides information and the system tries to complete the task. If some information is missing, the system requests it to the user.

A frame is used to describe: (i) a task the user may request; (ii) the information needed to perform

```
<CLASSIFICATION_RULES>
        <RULE prob='0.27' name="BuyTicket" resultStr="SELECT ... see previous picture">
        <SLOT_VALUE prob='0.3' slotKey="1">Ticket</SLOT_VALUE>
        <AND prob='0.9'>
                    <SLOT_FILLED prob='0.9'>3</SLOT_FILLED>
        </AND>
     </RULE>
</CLASSIFICATION_RULES>
```

Figure 5: Classification rule sample after user's utterance and problem solving act processing.

```
<VALIDATION_RULES>
      <NOT><EQUAL>
          <SLOT_VALUE slotKey="2" />
          <SLOT_VALUE slotKey="3" />
     </EQUAL></NOT>
     <ASCENDING>
          <SLOT_VALUE slotKey="4" />
          <SLOT_VALUE slotKey="5" />
     </ASCENDING>
</VALIDATION_RULES>
```

Figure 6: Initial Validation rule.

that task; and (iii) a set of constraints.

The Task Manager is responsible for generating and the managing frame instantiations.

4.1 Generation of Problem Solving Acts

An instantiation of a frame, called Problem Solving Act (PSA) distinguishes the needed information already available (already provided by the user during the interaction) and the needed, but missing, information required to perform the task. Each PSA also contains the classification rules that have to become true to complete the frame, along with the validation rules that prevent its misuse. For example, assuming the initial PSA is the one shown in figure 4, and that the user provides "Lisbon" as the start city and "10:00" as the pretended departure time, then the <SLOT_FILLED>2</SLOT_FILLED> and the <OR>…</OR> operators are removed (see figure 5).

Beyond its logical value, each (instantiated) rule also has a score, which is used to determine the best rule to be used in the next iteration towards the solution. Therefore, the classification rules scores of each PSA are used to compute the PSA's recognition score (Allen, et al, 1999). In the same way, the scores of the validation rules support the computation of the answer scores (Allen, et al, 1999).

The recognition score evaluates the rule requirements already accomplished. The answer score is a measure of the correctness of the data already provided by the user. A "bad" answer score can trigger the dispatch of a message to the user, reporting the constraint violation. The justification for the existence of these scores can be found in (Allen, et al, 1999).

4.1.1 PSA Score Values

Each operator has, by default, a heuristic weight (operator score) of 1.0, except the ones in Table 1. When generating a PSA, the score of each operator and the score of each rule are computed. These values are obtained recursively by multiplying each operator score with the score of the suboperators that compose the rule.

AND suboperators are valid when they have the true value, NOT suboperators are valid when they have the false value, ASCENDING suboperators are valid if the first one is lesser then the second one, and so on. If all the rules suboperators are valid then the rule has the maximum score. If all the rule suboperators are non valid, the rule has the minimum score. Otherwise, the rules scores are between the interval [0.0..1.0]. A large amount of valid suboperator causes a higher score. This is useful to determine the best rule, to take in the next move.

```
<VALIDATION_RULES>
  <ASCENDING prob='0.081' >
        <SLOT_VALUE prob='0.3' slotKey="4" />
        <SLOT_VALUE prob='0.3' slotKey="5" />
  </ASCENDING>
</VALIDATION_RULES>
```

Figure 7: Validation rule sample after user's utterance and problem solving act processing.

4.1.2 Example of processing a user utterance

Let us suppose the user asserted the intention to leave Lisbon, having as departure time 15:00 and arrival time 10:00. Figure 5 represents an instantiated classification rule that results from processing the previous affirmations. Since we already have the knowledge about the start city, departure time and arrival time, the PSA is simplified to contain only what we do not know, i.e., the type of request the user intends to do and the destination city. The score value of the classification rule shown in Figure 4 has increased from 0.21 to 0.27 (Figure 5). This simple method of calculating the score values coherently reflects the progression towards the completion of a goal.

Table 1 - Operator heuristic values.

Operator Name	Heuristic Value
EQUAL	0.8
ASCENDING	0.9
SLOT_VALUE	0.3
SLOT_FILLED	0.9

Nevertheless, the user's supplied data is against the validation rule that rejects any arrival time occurring before the departure time. The PSA will have a recognition score of 0.27, and answer score 0.081 (see Figures 5 and 7), meaning that we are approaching a goal, but at the same time invalidating the frame. These values will also be used in the Interpretation Manager (see section 2).

PROTOTYPE

To evaluate the capabilities of frames we developed a dialog manager prototype, based on the described architecture, to answer inquires about bus travel. When we decided to introduce two new domains, cinema tickets and celebrities, it took us only one week to develop the new frames.

5 CONCLUSIONS

Because frames encapsulate all the necessary mechanisms to deal with a domain, they are a valuable resource when it is necessary to introduce a new domain into the dialog system. They also make it possible to simultaneously manage several domains.

Frames provide some of the basic services needed by a dialog manager: (i) communication with the domain application (e.g., a database) through the description of the accepted requests; (ii) definition of the restrictions that help maintaining coherent the information conveyed by the user.

Since all the relevant information about any domain is represented in a frame, the decision to support a new domain only implies the design of a new frame.

REFERENCES

M. F McTear, 1998. Spoken Dialogue Technology: Enabling the conversational user interface, Ulste Univ.

J. Allen, G. Ferguson, A. Stent, 1999. An Architecture for more realistic conversational system, Univ. of Rochester.

D. K. Byron, 1999. Improving Discourse Management in TRIPS-98. Proceedings of the 6th European conference on Speech Communication and Technology, Budapest, Hungary.