A natural language dialogue manager for accessing databases

Salvador Abreu[2], Paulo Quaresma[2], Luis Quintano[1], and Irene Rodrigues[2]

[1] ljcq@sc.uevora.pt, Serviço de Computação, Universidade de Évora, Portugal
[2] spa|pq|ipr@di.uevora.pt, Departamento de Informática, Universidade de Évora, Portugal

Abstract. A logic programming based dialogue system with the capability of inferring user attitudes and accessing heterogeneous external relational databases while doing syntactic and semantic sentence parsing is presented. The system was built using a logic programming language – Prolog –, a development tool – ISCO [2] – and a language for representing actions – LUPS[4]. An application of the developed system to the Universidade de Évora's Integrated Information System (SIIUE) was developed and some examples of typical dialogues are presented.

1 Introduction

Over the last couple of years Universidade de Évora has committed itself to the development of an Integrated Information System (SIIUE) [3]. The information stored in SIIUE has been in growing demand by many university services and external entities. As more data was gathered, SIIUE has become the main source of information for Universidade de Évora faculty, students and staff. As a consequence, the development of a natural language dialogue system allowing users to interact with SIIUE directly in Portuguese was considered a major requirement.

The dialogue system needed to analyze the sentences (syntactically, semantically, and pragmatically) and to interact with the SIIUE knowledge bases in order to obtain the required information. The different modules were developed in Prolog and the interaction with the knowledge bases was done using ISCO [3, 2], a new logic-based programming framework which is able to handle relational database integration and web-based development. The pragmatic module was built using a language for describing actions, LUPS[4], which allows the system to make inferences about the user intentions and beliefs and to be able to have cooperative dialogues with the users.

The remainder of this article is structured as follows: in section 2, the ISCO language is described. In section 3, the LUPS language is briefly described. In section 4 the overall structure of the system is presented; section 5 deal with the semantic/ pragmatic interpretation. In section 6 a more extensive example is presented and, finally, in section 7 we discuss some current limitations of the system and lay out possible lines of future work.

2 ISCO

ISCO is a new Logic-Based development language implemented over GNU Prolog [5] that gives the developer several distinct possibilities, useful for the development of applications such as SIIUE:

– Gives a simple database structure description language that can help in database schema analysis. Tools are available to create an ISCO database description from an existing relational database schema and also the opposite action, i.e. to create a relational database schema from a ISCO class description.
– View relational databases as a part of a declarative/deductive object-oriented (with inheritance) database. Among other things, the system maps relational tables to classes – which may be used as Prolog predicates.
– Gives simple access to relational data through ODBC using a GNU Prolog interface with unixODBC, which has been developed within the SIIUE project.
– Creates ISCO/Prolog executables ready for use from PHP scripts [1] in web-based interfaces. The PHP extensions have also been developed specifically for use with ISCO.

The dialogue modules use ISCO's capability to establish connections from Prolog to the relational databases in an efficient way. Moreover, ISCO was directly used to access a relational database containing a fairly complete Portuguese dictionary (Polaris) [8], which is used by the syntactical and semantical analyzer module.

3 Dynamic LP and LUPS

Knowledge evolves from one knowledge state to another as a result of knowledge updates. In [4] it was introduced a declarative, high-level language for knowledge updates called *LUPS* ("Language of UPdateS") that describes transitions between consecutive knowledge states. It consists of update commands, which specify what updates should be applied to any given knowledge state in order to obtain the next knowledge state. Below, a brief description of LUPS that does not include all of the available update commands and omits some details is presented.

The simplest update command consists of adding a rule to the current knowledge state and has the form: $assert\ (L \leftarrow L_1, \ldots, L_k)$. In general, the addition of a rule to a knowledge state may depend upon some preconditions being true in the current state. To allow for that, the assert command in LUPS has a more general form:

$$assert\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \tag{1}$$

The meaning of this assert command is that if the preconditions $L_{k+1}, \ldots, L_m$ are true in the current knowledge state, then the rule $L \leftarrow L_1, \ldots, L_k$ should hold true in the successor knowledge state. The added rules are *inertial,* i.e., they remain in force from then on by inertia, until possibly defeated by some future update or until retracted.

However, in some cases the persistence of rules by inertia should not be assumed. Take, for instance, an user utterance. This is a *one-time event* that should not persist by inertia after the successor state. Accordingly, the assert command allows for the keyword *event*, indicating that the added *rule* is *non-inertial.*

$$assert\ event\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \tag{2}$$

Update commands themselves (rather than the rules they assert) may either be one-time, non-persistent update commands or they may remain in force until canceled. In order to specify such *persistent update commands* (which we call *update laws*) there is the syntax:

$$always\ [event]\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \tag{3}$$

4 Natural Language Dialogue System

As was already stated the main goal of this work was to build a system that could get a Portuguese natural language sentence sent by a user through a web interface and respond accordingly.

To answer the question/sentence the system has to pass it from a web-based interface to a GNU Prolog/ISCO active process (A), the process must analyze the sentence accessing the relational database(s) when needed to get or check any information (B) and finally when acquiring all needed information, it has to build a comprehensive answer and pass it to the web-based interface (C).

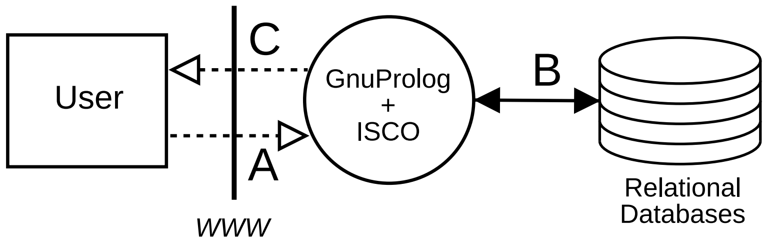

Fig. 1. Simplified SIIUE-NL system architecture

The parse that is made to the sentence is split in three different parts: Syntax, Semantics, Pragmatics

The user sends the question about the information that exists in the SIIUE. For that he/she uses a web-based interface using the scripting language PHP[1] and the tools available by the php module of ISCO.

The question is then sent to an active Prolog process that already knows all the relational database structure to be used. ISCO manages the conversion of that structure to Prolog predicates that can access the relational databases through SQL primitives as selects, inserts, updates or deletes. In our case, as we're facing a querying system we only need to use selects.

Besides all database structures, this Prolog process does all syntactic, semantic and pragmatic analysis. For that it previously had to do some pre-processing with the relational database structure to generate semantic/pragmatic database driven rules.

After analyzing the sentence received, the process has to generate an adequate answer, which will be shown to the user through the web interface.

Syntax Analysis The question reaches the syntax analysis module in form of a list to be parsed. This syntactic interpreter will identify the correct syntactic structure of the sentence.

This syntactic interpreter was built using *Chart Parsers*[6]. This is one of many techniques to build syntactic interpreters. The decision of developing the interpreter using this technique was mainly because chart parsers can parse incomplete sentences. The user can place complete or incomplete questions and the system must be able to answer them accordingly, so the need to parse incomplete sentences is essential.

The chart parser uses a set of syntactic rules that identify the Portuguese sentence structures and tries to match these rules with the input sentence(s).

The interpreter also uses a lexicon to identify the syntactic properties of the words in the sentences. For that the interpreter is connected with a relational database (Polaris) which has syntactic (and semantic) information about Portuguese words. This integration is possible through ISCO because this tool already knows the Polaris database structure and can access it through ODBC.

This module will produce an output that consists in a list with all possible syntactic representations of the sentence placed by the user.

As an example, if the user placed the following sentence as input to the system:

"Paulo lecciona Arquitectura?"

("Does Paulo teachs Architecture?")

With this question the user intends to know if Paulo is the responsible for the Architecture course.

The syntax module will return a list with the sentences' syntactic parse

```
phrase([np([n('Paulo',s+m+_)]), vp(v('teach',1+s+_)),
      args_v([np([name('Architecture',s+m+_)])])]).
```

Semantic Interpretation The syntactic parsing output will be sent to the semantic module. This module will get the syntactic structure and rewrite it in a First-Order Logic. The technique used for this parsing is based on DRS's (Discourse Representation Structures)[7].

This technique identifies triggering syntactic configurations on the global sentence structure, which activates some rewriting rules:

Proper Nouns - When a proper noun syntactic structure is found, a new discourse referent is added replacing the proper noun syntactic configuration.

Pronouns - When a pronoun syntactic structure is found, a new discourse referent (`A`) is added as a condition `A=B` in which (`B`) is a suitable discourse referent that already exists. The syntactic configuration is replaced by the new referent.

Verbs - When a verb (`verb`) is found with its arguments already rewritten (`A` and `B`), the condition `verb(A,B)` is created and replaces the syntactic configuration that activated this rule.

This module returns two lists, one with the new sentence rewritten and the other with information about the referents that were created in this analysis.

For instance, if this module receives the syntax module output presented in the previous sections it will return the semantic representation of the sentence:

name(A, 'Paulo'), name(B, 'Architecture'), teaches(A,B)

and a list with information about the discourse referents:

[ref(A,s+m+_),ref(B,s+f+_)]

5 Pragmatics Interpretation

The pragmatics module receives the sentence rewritten (into a First Order Logic form) and tries to interpret it in the context of the dialogue and in the context of the user model.

In order to achieve this behavior it is necessary to recognize the speech act associated with the sentence (in this domain it can be an $inform$, a $request$, or a $askif$ speech act), to model the user attitudes (intentions and beliefs), and to represent and to make inferences over the dialogue domain. After having interpreted the sentence, the pragmatic module establishes the connection with the databases to fetch data which will be used to give a coherent answer to the user.

As a first step, it was necessary to represent the knowledge conveyed by the database schemas as logic programming rules. Using the ability that ISCO has to describe external relational databases, it was possible to generate these rules according to the existent relations (classes) and its attributes. This process was described in detail in [11] and, in this paper, we will present only one example:

```
teaches(A,B) <-                                           (4)
  all_ids(si_teacher,A),
  all_ids(si_course,B),
  abduct(si_teaches(course = B, lecturer = A)).
```

The verb `teaches` receives a teacher (`A`) and a course (`B`). It checks both of them against the database information and abducts all possible relations between teachers and courses lectured. The `all_ids` returns variable `A` and `B` restricted to all known teachers and courses identifiers. This is implemented using FD constraints.

After having defined these domain representation rules it is necessary to pragmatically interpret the sentence. In order to achieve this goal the system needs to model the speech acts, the user attitudes (intentions and beliefs) and the connection between attitudes and actions. This task is also achieved through the use of logic programming framework rules and the LUPS language (see [10, 9] for a more detailed description of these rules).

For instance, the rules which describe the effect of an inform, a request, and a ask-if speech act from the point of view of the receptor are:

always bel(A,bel(B,P)) *when* inform(B,A,P)
always bel(A,int(B,Action)) *when* request(B,A,Action)
always bel(A,int(B,inform-if(A,B,P))) *when* ask-if(B,A,P)

In order to represent collaborative behavior it is necessary to model how information is transferred between the different agents:

always bel(A,P) *when* bel(A,bel(B,P))
always int(A,Action) *when* bel(A,int(B,Action))

These two rules allow beliefs and intentions to be transferred between agents if they are not inconsistent with their previous mental state.

There is also the need for rules that link the system intentions and the accesses to the databases:

always yes(P) ← query(P), one-sol(P) *when* int(A, inform-if(A, B, P))
always no(P) ← query(P), no-sol(P) *when* int(A, inform-if(A, B, P))
always clarif(P) ← query(P), n-sol(P) *when* int(A, inform-if(A, B, P))

These three rules update the system's mental state with the result of accessing the databases: yes, if there is only one solution; no, if there are no solutions; and clarification, if there are many solutions (the predicates that determine the cardinality of the solution are not presented here due to space problems, but there implementation is quite simple).

After accessing the databases, the system should answer the user:

always confirm(A,B,P) *when* yes, int(A, inform-if(A, B, P))
always not int(A, inform-if(A,B,P)) *when* yes, int(A, inform-if(A, B, P))
always reject(A,B,P) *when* no, int(A, inform-if(A, B, P))
always not int(A, inform-if(A,B,P)) *when* yes, int(A, inform-if(A, B, P))
always ask-select(A,B,C) ← cluster(P,C) *when* clarif(P), int(A, inform-if(A, B, P))

The first two rules define that, after a unique solution query, the system confirms the answer and terminates the intention to answer the user. The next two rules define that, after a no solution query, the system rejects the question and terminates the intention to answer the user. The last two rules define that, after a multiple solution query, the system starts a clarification answer, asking the user to select one of the possible solutions. In order to collaborate with the user we have defined a cluster predicate that tries to aggregate the solutions into coherent sets, but its definition is outside the scope of this paper.

6 A Practical Example

For a better comprehension and to give a generic view of the system implementation we will present a complete example starting from the user input:

"Does Paulo teachs Architecture?"

After the syntax and semantic module analyses we will have:

name(A, 'Paulo'), name(B, 'Arquitecture'), teaches(A,B)

and a list with information about the discourse referents:

[ref(A,sin+masc+_),ref(B,sin+masc+_)]

The semantic/pragmatic interpretation will give rise to following expression:

si_ teach(lecturer=A,course=B),

with A a variable constrained to values of all the database identifiers that are named Paulo; and B a variable constrained to the values of the identifiers that are courses with the word Architecture in their name.

After having the sentence re-written into its semantic representation form, the speech act is recognized and we'll have:

ask-if(user, system, [si_teaches(lecture=A,course=B)]).

Using the "ask-if" and the transference of intentions LUPS rules we'll have:

int(system,inf-if(system, user, [si_teaches(lecture=A,course=B)])).

Now, using the rules presented in the previous section, the system accesses the databases (using the ISCO modules). Suppose there are two possible solutions (one having Paulo Quaresma as the lecturer and the other having Paulo Santos as the lecturer). We'll have:

int(system,inf-if(system, user, [si_teaches(lecture=A,course=B)])).

clarif([si_teaches(lecture=A,course=B)]).

with variable A constrained to the set $100, 120$ and $name(100,'PauloQuaresma')$, $name(120,'PauloSantos')$.

The $cluster(P, C)$ rule will obtain two possible selections and the correspondent ask-selection speech act will be performed:

ask-select(system,user,[(name(A, 'Paulo Quaresma') ; name(A, 'Paulo Santos')]).

"Is the lecturer Paulo Quaresma or Paulo Santos?"

Now, suppose the user answers:

"Paulo Quaresma."

inform(user, system, [lecturer(A), name(A, 'Paulo Quaresma')]).

Using the inform and the transference rules, the system is able to start a new belief and to add it to the constraints of the current question:

bel(system, [name(A, 'Paulo Quaresma')]).

int(system,inf-if(system, user, [si_teaches(lecture=A,course=B)])).

A is constraint to the identifier of Paulo Quaresma.

With this new constraint the query will have only one solution (yes, for instance) and the system will perform a confirm speech act.

7 Conclusions and Future Work

The dialogue system described in this paper is still in an experimental stage, but we intend to make it available to all users using the University's internal web interface, via Universidade de Évora's web page (`http://www.uevora.pt/`) in a short period of time.

Clearly, and due to its complexity, all modules have aspects that may be improved:

- The syntactical coverage of the Portuguese grammar
- The coverage of the semantic analyzer (plurals, quantifiers, ...)
- The capability of the pragmatic module to take into account previous interactions and the user models

However, we believe that probably the major positive aspect of the described system is its modularity and the integration of several AI techniques under a logic programming paradigm.

References

1. Php hypertext processor - http://www.php3.org.
2. Salvador Abreu. Isco: A practical language for heterogeneous information system construction. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.
3. Salvador Pinto Abreu. A Logic-based Information System. In Enrico Pontelli and Vitor Santos-Costa, editors, *2nd International Workshop on Practical Aspects of Declarative Languages (PADL'2000)*, volume 1753 of *Lecture Notes in Computer Science*, pages 141–153, Boston, MA, USA, January 2000. Springer-Verlag.
4. J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Preliminary exploration on actions as updates. In M. C. Meo and M. Vilares-Ferro, editors, *Procs. of the 1999 Joint Conference on Declarative Programming (AGP'99)*, pages 259–271, L'Aquila, Italy, September 1999.
5. D. Diaz. http://www.gnu.org/software/prolog, 1999.
6. Gerald Gazdar and Chris Mellish. *Natural Language Processing in PROLOG*. Addison-Wesley, 1989.
7. H. Kamp and U. Reyle. *From Discourse to Logic*. Kluwer, Dordrecht, 1993.
8. J. Lopes, N. Marques, and V. Rocio. Polaris, a portuguese lexicon acquisition and retrieval interactive system, 1994.
9. P. Quaresma and J. G. Lopes. Unified logic programming approach to the abduction of plans and intentions in information-seeking dialogues. *Journal of Logic Programming*, 54, 1995.
10. Paulo Quaresma and Irene Rodrigues. Using logic programming to model multi-agent web legal systems – an application report. In *Proceedings of the ICAIL'01 - International Conference on Artificial Intelligence and Law*, St. Louis, USA, May 2001. ACM. 10 pages.
11. Luis Quintano, Irene Rodrigues, and Salvador Abreu. Relational information retrieval through natural lanaguage analysis. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.