A dialogue manager for accessing databases

Salvador Abreu, Paulo Quaresma, Luis Quintano, Irene Rodrigues
Departamento de Informática,
Universidade de Évora,
7000 Évora, Portugal
`spa|pq|ljcq|ipr@di.uevora.pt`,
Tel: 351 266 745300; Fax: 351 266 745360

Abstract.
We present a logic programming based dialogue system that enables the access in natural language to the heterogeneous external relational databases of the Évora University.

The proposed system has the capability of inferring user attitudes and uses ISCO in order to view the University relational databases as a part of a declarative/deductive object-oriented (with inheritance) database allowing the mapping of relational tables to classes – which may be used as logic (Prolog) predicates.

Keywords: dialogue managers, natural language processing, logic programming, knowledge bases.

1 Introduction

Over the last couple of years Universidade de Évora has committed itself to the development of an Integrated Information System (SIIUE) [2]. SIIUE is not just a set of databases which have information about several aspects of the University (Academic, Research, etc.) but also a group of applications built to give that information an easy (read/write) access in conjunction with the ISCO development tool [2, 1].

The main purpose of the dialogue system presented in this article is to enable the users – eg. the administration – that are not aware of the University information structure to obtain the information they need. For instance, in the context of the University degrees evaluation the following question could be posed by the administrative staff in order to evaluate the performance of computer science professors that teach in the mathematics degree.

Que docentes de informática leccionam matemática?

(Which computer science professors teach mathematics?)

This information could be obtained in the University web pages (or with an SQL or ISCO query) if the staff knows the internal structure of the university information. The above natural language query will allow the staff to obtain the answer even when they do not know the information structure.

This query has some ambiguities that our system must resolve before it answers the user. The system is able to dialogue with the user to resolve ambiguities such as the one introduced by the noun phrase “computer science professors”. In the context of the University of Évora information system this phrase may refer to[1]:

- professors that teach computer science courses
- professors that teach courses from the computer science degree
- professors that belong to the computer science department

The phrase “mathematics” in this context may refer to:

- applied mathematic curriculum
- mathematic teachers curriculum
- mathematic courses

Note that the administrative staff may not be aware of all the possible interpretations of their query. They may not know that the university has two degrees with the word ’mathematics’ in their name but when they are confronted with those options they are able to choose the right one.

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 deals with the semantic/ pragmatic interpretation. In section 6 the dialogue manager is presented 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 that gives the developer several distinct possibilities, useful for the development of applications such as SIIUE:

- It 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.
- 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 arbitrary relational data through ODBC using a GNU Prolog interface with unixODBC, which has been developed within the SIIUE project. Whenever appropriate, native interfaces to specific databases have been developed; such is the case for PostgreSQL.

[1]This example will be detailed in the following sections where we show how is that our system deals with the problems of this question.

- By virtue of the GNU Prolog implementation base, ISCO applications may resort to Constraint Logic Programming techniques. More specifically, finite domain constraints are supported in ISCO queries.

The dialogue modules use ISCO's capability to establish connections from Prolog to the relational databases in an efficient way. For example, the following SQL table:

```
CREATE TABLE "student" (
 "number" int4 NOT NULL,
 "name" text,
 "id_card" text,
 Constraint "number_pkey" Primary Key ("number")
);
```

Maps into the following ISCO class definition, which can be automatically generated by a support tool:

```
external(sac,student) class student.
  number: int. key.
  name: text.
  id_card: text.
```

Class `student` is mapped into clauses for a set of Prolog predicates that implement the four basic operations: query, insert, update and delete.

Variables occurring in queries are mapped to SQL and may carry CLP(FD) constraints, which will be expressed in SQL, whenever possible. For example, suppose variable `X` is an FD variable whose domain is `(1..1000)`, the query `student(number = X, name = Y)` will return all pairs `(X, Y)` where `X` is a student registration number and `Y` is the student's name. `X` is subject to the constraints that were valid upon execution of the query, ie. in the range 1 to 1000.

ISCO class declarations feature inheritance, simple domain integrity constraints, global integrity constraints and a comprehensive and simple to express access-control mechanism.

3 Dynamic LP and LUPS

LUPS ("Language of UPdateS" [3]) is a declarative language for knowledge updates that describes transitions between consecutive knowledge states. It consists of commands, which specify what updates should be applied to any given knowledge state in order to obtain the next knowledge state.

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:

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

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*.

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

In order to specify *persistent update commands* (which are called *update laws*) it exists 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).

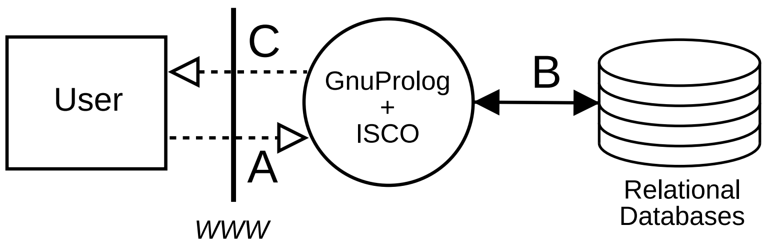

Figure 1: Simplifi ed SIIUE-NL system architecture

The processement of a natural language sentence is split in four subprocesses: Syntax, Semantics, Pragmatics, Dialogue manager

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 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. 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 syntactic interpreter was built using *Chart Parsers*[4]. 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 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:

"Que docentes de informática leccionam à matemática?"

("Which computer science professors teach mathematics?")

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

```
phrase([np([det(que,_+_+_), n('docente',3+p+m),
            pp(de,np([name('informática',3+s+f)]))]),
        vp(v('leccionar',3+p+_)),
        args_v([np([name('matemática',3+s+f)])])]).
%(phrase([np([det(which,_+p+_), n('professor',_+p+m),
%              pp(of,np([name('computer science',_+s+m)]))]),
%         vp(v('teach',3+p+_)),
%         args_v([np([name('mathematic',_+p+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)[6].

This technique identifies triggering syntactic configurations on the global sentence structure, which activates the rewriting rules. We always rewrite the pp's by the relation 'rel(A,B)' postponing its interpretation to the semantic pragmatic module.

This module returns a DRS build with 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 as semantic representation of the sentence the expression:

```
professor(A), name(B, 'computer science'),
name(C,'mathematics'),
rel(A,B), teaches(A,C).
```

and the list with the discourse referents:

```
[ref(A,p+_+_,which),ref(B,s+_+_,undef),ref(C,p+_+_,undef)]
```

5 Semantic/Pragmatic Interpretation

The semantic/pragmatic module receives the sentence rewritten (into a First Order Logic form) and tries to interpret it in the context of the SIIUE databases information.

In order to achieve this behavior the system tries to find the best explanations for the sentence logic form to be true in the knowledge base for the semantic/pragmatic interpretation. This strategie for interpretation is known as "interpretation as abduction" [5].

The knowledge base for the semantic/pragmatic interpretation is built from the ISCO description of the SIIUE databases. The Kb rules are generated from the databases descriptions. This process was described in detail in [9].

From the description of the relation *si_teaches* the KB has rules for the interpretation of the predicates: *teaches(A,B)* and *rel(A,B)*.

```
class si_teaches.
  lecture: si_individual.id.
  degree: si_degree.code.
  course: si_course.code.
  year: int.
```

The rules in semantic/pragmatic Knowledge base are like the one below and they enable the interpretation of a sentence like "lecture teaches course" as the ISCO expression:
si_teaches(course = B, lecturer = A).

```
  rel(A,B) <-
    si_lecturer(A),
    si_course(B),
    abduct(si_teaches(course = B, lecturer = A)).
```

In the semantic/pragmatic interpretation the evaluation of a predicate like "si_course(A)" is done by an access to the relational databases through ISCO. The result of such an evaluation is the constraint of variable A to database identifiers of objects from class course.

The interpretation of names (eg. name(A,mathematics)) is done by accessing the SIIUE in order to collect in (constraint) A all entity identifiers that have in their name the word 'mathematics'.

The result of interpreting the sentence represented by:

```
professor(A), name(B, 'computer science'),
name(C,'mathematics'),
rel(A,B), teaches(A,C).

[ref(A,p+_+_,which),ref(B,s+_+_,undef),ref(C,p+_+_,undef)]
```

is the following ISCO expressions:

1. teaches(lecturer=A,course=B,degree=C)
   - `A=_#(7001...7852)` – A is constraint to all lectures
   - `B=_#(1046..1049:1345:1456..1457)` – B is constraint to the courses that have in their name the expression 'computer science'

- `C=_#(3046:3123)` – C is constraint to degrees with the word 'mathematics' in their name.

2. teaches(lecturer=A:,course=C:,degree=B:)

   - `A=_#(7001...7852)` – A is constraint to all lectures
   - `B=_#(3012)` – B is constraint to degrees with the word 'computer science' in their name.
   - `C=_#(1265..1281:1431:1454..1455:1784:1791)` – is constraint to the courses that have in their name the word 'mathematics'.

3. department(key=B:, lecturer=A:), teaches(lecturer=A:,degree=C:)

   - `A=_#(7001...7852)` – A is constraint to all lectures
   - `B=_#(101)` – B is constraint to departments with the word 'computer science' in their name.
   - `C=_#(3046:3123)` – C is constraint to degrees that have in their name the word 'mathematics'.

4. department(key=B:, lecturer=A:), teaches(lecturer=A:,course=C:)

   - `A=_#(7001...7852)` – A is constraint to all lectures
   - `B=_#(101)` – B is constraint to departments with the word 'computer science' in their name.
   - `C=_#(1265..1281:1431:1454..1455:1784:1791)` – C is constraint to courses that have in their name the word 'mathematics'.

The above ISCO expression contains the possible interpretations of the sentence in the context of the University information.

6 Dialogue Manager

The Dialogue Manager must disambiguate the sentence possible semantic pragmatic interpretations. It has 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.

This task is achieved through the use of a logic programming framework rules and the LUPS language (see [8, 7] 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 databases accesses:

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 knowledge bases: 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 reject(A,B,P) *when* no, 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 rule defines that, after a unique solution query, the system confirms the answer. The next rule defines that, after a no solution query, the system rejects the question. The last rule defines 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 complete definition is outside the scope of this paper.

Considering the question:

Which computer science professors teach mathematics?

As we presented in the previous section the semantic/pragmatic interpretation will give rise to the following expression:

```
R=[ref(A,p+_+f_,which),ref(B,s+_+_,undef),ref(C,p+_+_,undef)]

V=[(A1,B1,C1),(A2,B2,C3),(A3,B3,C3),(A4,B4,C4)]

I=[si_teaches(lecturer=A1, course=B1, degree=C1),
   si_teaches(lecturer=A2, course=C2, degree=B2),
   (si_department(key=B3, lecturer=A3),
         si_teaches(lecturer=A3, degree=C3)),
   (si_department(key=B4, lecturer=A4),
         si_teaches(lecturer=A4, course=C4))]
```

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

```
ask-if(user, system, [R,V,I])
```

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

```
int(system,inform-if(system, user, [R,V,I])).
```

Now, using the rules presented in the previous section, the system may access the knowledge bases (using the ISCO modules). The first step is to decide what is the meaning of the user sentence since there are four possible interpretations for each discourse referent (the V list has four elements). Using the following rule, the system is able to detect if there are many possible interpretations, and to obtain, for each referent variable, its respective classes. Then, it will ask the user to disambiguate the question:

> *always* ask-select(A,B,[VC,R,V,I]) ← int(A, inform-if(A, B, [R,V,I])), cardinality(V,N), N > 1, get_classes(V, VC).

$get\_classes$ is a predicate that obtains the set of possible classes for each variable referent. For instance, in our example we have:

> get_classes([(A1,B1,C1),(A2,B2,C2),(A3,B3,C3),(A4,B4,C4)], [([lecturer], [course, degree, department], [degree, course])])

The ask_select predicate chooses the first referent variable which has more than one possible class (B in the example) and, as a consequence, the dialogue system is able to ask if the user wants 'computer science' to refer to:

- course
- degree
- department

Suppose the user selects the option $department$. In this situation, there are two possible options for variable C: degree and course. The system will ask again the user to select the class of variable C. Suppose the user selects $degree$. Now, there is only one possible interpretation.

```
I = [teaches(lecturer=A, degree=C),
      department(key=B, lecturer=A)]
```

After having disambiguated the question interpretation, the system will use the rule presented previously and it will access the databases: $query(I)$

Suppose there are two possible solutions for the variable referent C: one for the degree of "Applied Mathematics" and another for "Mathematic Teachers". In this situation the n_sol predicate holds and the system will start a clarify interaction (using the ask_select rule and the cluster predicate):

```
ask-select(system, user, ["Applied Mathematics",
                          "Mathematic Teachers"]).
```

Now, suppose the user answers "Applied Mathematics". Using the inform and the transference rules, the system is able to add the information to the constraints of the current question and to, finally, answer the user question.

7 Conclusions and Future Work

The dialogue system described in this paper is still in an experimental stage and it has not been tested in "real" situations. We intend to have 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 dialogue manager to take into account previous interactions and the user models

References

[1] Salvador Abreu. Isco: A practical language for heterogeneous information system construction. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.

[2] 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.

[3] 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.

[4] Gerald Gazdar and Chris Mellish. *Natural Language Processing in PROLOG*. Addison-Wesley, 1989.

[5] Jerry Hobbs, Mark Stickel, Douglas Appelt, and Paul Martin. Interpretation as abduction. Technical Report SRI Technical Note 499, 333 Ravenswood Ave., Menlo Park, CA 94025, 1990.

[6] H. Kamp and U. Reyle. *From Discourse to Logic*. Kluwer, Dordrecht, 1993.

[7] P. Quaresma and J. G. Lopes. Unifi ed logic programming approach to the abduction of plans and intentions in information-seeking dialogues. *Journal of Logic Programming*, 54, 1995.

[8] 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.

[9] Luis Quintano, Irene Rodrigues, and Salvador Abreu. Relational information retrieval through natural lanaguage analysis. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.