Buletinul Stiintific al Universitatii "Politehnica" din Timisoara, ROMANIA
Seria AUTOMATICA si CALCULATOARE
PERIODICA POLITEHNICA, Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE
Vol. 47 (61), 2002, ISSN 1224-600X

Portable Ontology Query Language (POQL)

Tudor Muresan, Rodica Potolea, Alin Suciu,
Emilia Cimpian, Adrian Mocan, Radu Popovici, Horatiu Tarcea

Computer Science Department, Technical University of Cluj-Napoca,
24th Baritiu Street, Cluj-Napoca, Romania
{tmuresan, potolea, alin}@cs.utcluj.ro
{cemilia, madrian, pandrei, tiustin}@asterix.obs.utcluj.ro

Abstract *- This paper presents the definition and implementation of a query language for reusable knowledge bases, which uses the Prolog logical form. The advantage is, along with the complexity and flexibility of the allowed questions, the fact that it constitutes a theoretical interface with user friendly querying systems (i.e. NLI). Also, it makes use of the Prolog solving mechanism for an extensive search in the solution space, providing the framework for the development of theories for automated merging and alignment of existing ontologies. The current implementation represents a prototype of POQL.*

Keywords*: logical query language, knowledge base space searching, portable ontology, implementation*

I. INTRODUCTION

A large number of ontologies have been constructed taking into account the principle of generating reusable knowledge bases by adopting standard representational languages [2], [9] or by achieving portability through a translational approach [8]. The advantage of easy knowledge acquisition [1], [4] of the existing tools becomes a weakness from the querying point of view. Thus, it becomes appropriate the development of query tools independent of the ontology representation. Such query tools serve both for the development of user friendly query interfaces (i.e. Natural Language Interfaces) and for the purpose of merging and alignment of the existing ontologies. Furthermore, currently [7], there are yet extremely few theories or methods that facilitate or automate the process of reconciling disparate ontologies.

This paper presents the definition and implementation of a query language for reusable knowledge bases, which uses the Prolog logical form. The advantage is, along with the complexity and flexibility of the allowed questions, the fact that it constitutes a theoretical interface with user friendly querying systems (i.e. NLI). Also, it makes use of the Prolog solving mechanism, for generating all the solutions of a specific search, providing the framework for the development of theories for automated merging and alignment of existing ontologies.

In section 2 we give an overview of the main concepts pertaining to a reusable ontology frame which conform to the OKBC model [6]. The 3rd section constitutes a description of the syntax and semantics of the query language. Section 4 presents the architecture we chose to implement for our tool. Experimental results are shown in section 5. We conclude by presenting the conclusions and proposals for further development, in section 6.

II. OPEN KNOWLEDGE BASE CONNECTIVITY (OKBC) ONTOLOGY FRAME

An ontology is a specification of a representational vocabulary for a shared domain of discourse. The OKBC standard frame **ontology** consists of a hierarchy of **frames** [5], [6]. They are organized, according to their role, into three main categories: classes, slots and facets.

- **Classes** are concepts in the domain of discourse, collections of objects that have similar properties; they are arranged into a subclass-superclass hierarchy and allow multiple inheritance. There are two subcategories for classes: metaclasses - classes which have as instances other classes, and ordinary classes – which have ordinary instances as their materialization.
- **Slots** are named binary relations between a class and either another class or a primitive object in order to describe properties, attributes of classes or relations between classes. Slots attached to a class may be further constrained by facets.
- **Facets** are named ternary relations between a class, a slot, and either another class or a primitive object; they describe properties of slots and may impose additional constraints on a slot attached to a class.
- **Instances** are materializations of classes.

A **knowledge base** includes both the ontology and individual instances of classes with specific values for their slots. The distinction between classes and instances is not an absolute one due to the existence of metaclasses.

III. QUERIES: SYNTAX AND SEMANTICS

A query consists of one or more linked atomic queries. The syntax of such connections follows the Prolog logical form syntax, with conjunctions and disjunctions between expressions.

```
<query> ::= <atomic query> |
       <atomic query> <logical-op> |
       <query> |
       'not(' <query> ')'

<atomic query> ::=
       <term>  <poql-op>  <term>  |
       <path term> |
       <path  term>  <relational-op>
       <path    term>    |    <axiom-
       predicate> | '(' <query> ')'

<path term> ::=  <term> |
       <term> '.' <path term>

<term> ::= '<frame name>' |
    ('?' | '?_') <Prolog variable> |
       <Prolog constant>

<poql-op> ::= 'isa' | 'sub' | ':' |
       '::'

<relational-op> ::= '=' | '>=' |
       '=<' | '<' | '>' | '\=='

<logical-op> ::= ',' | ';'
```

The `<poql-op>` operators correspond to the relations between frames:

isa direct instance-class relation;
: transitive closure of *isa* relation;
sub direct inheritance relation;
:: transitive closure of *sub* relation.

The name of a frame may be simple, referring directly a frame of the knowledge base, or it may be a path. A path is a concatenation of slots s1, s2, ..., sn, written o.s1.s2....sn, where o is the frame slot s1 belongs to (class or instance), o.s1 refers the frame slot s2 belongs to and so forth. Such expression has within itself a truth value given by the (non)existence of the path.

The queries' semantic is specified by a meta-interpreter [3] for Prolog with freeze, in a compositional manner [11]. For this computational model in [3] it is proven the soundness and the safety of negation as failure (see Appendix A). For the `<logical-op>` we define:

```
semantic(Q1 <l-op> Q2) →
sem_freeze(Q1) <l-op> sem_freeze(Q2)
```

where

```
sem_freeze(Q) →
       semantic(Q), sem_queue.
```

For the atomic query Q_A with `<poql-op>` we define:

```
semantic(QA) → postpone(QA)
```

where

```
postpone(X <poql-op> Y) →
        freeze((X,Y), X <poql-op> Y)
```

From the above definition follows that the atomic queries are postponed through the freeze predicate until at least one of its variables becomes instantiated. Postponed atomic queries are resumed by the `sem_queue` predicate.

The following equivalences hold true:

```
a <poql-op> B ↔
       forall(X, a <poql-op> X , Lx),
       member(B, Lx)
forall(X,a <poql-op> X,Lx) ↔
       api(<poql-op>)(a,Lx)
```

where `api(<poql-op>)(a,Lx)`represents an API call specific to the ontology representation. These equivalences allow the definition of the semantic for the postponed atomic queries through the correspondent ontology program interface:

```
semantic(a <poql-op> B ) →
  api(<poql-op>)(a,Lx),member(B, Lx)
```

This renders the Prolog search strategy independent of the actual representation of the queried ontology (Fig. 1).

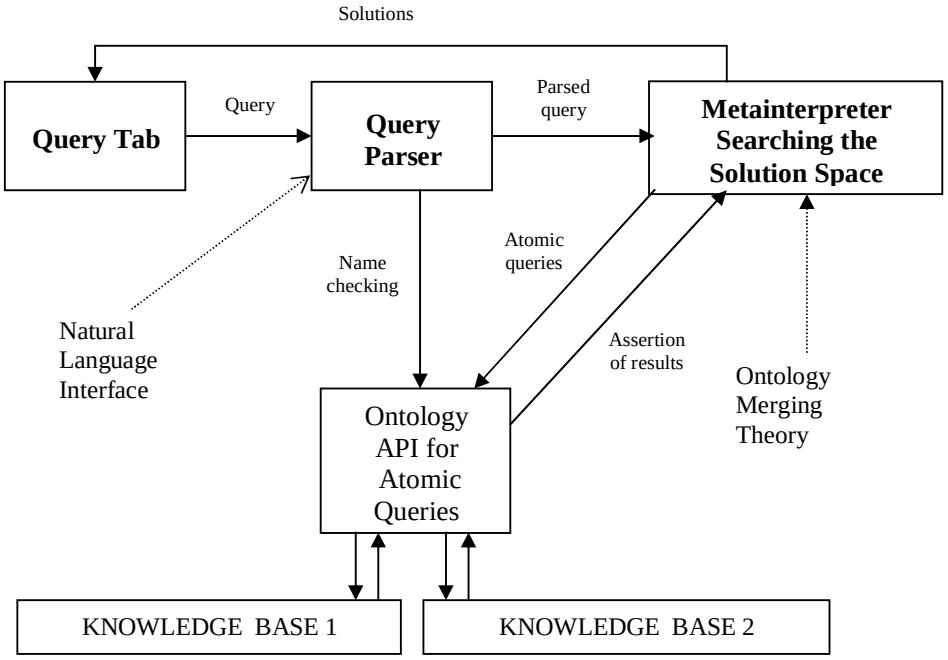

Fig. 1 System Architecture

IV. POQL ARCHITECTURE AND IMPLEMENTATION

The architecture of POQL is shown in Fig.1.
A query is entered in a Prolog like logical form, using the interface we have developed. Subsequently, a parser performs syntax and name checking, converting the query to a string of our convenience which is further passed to the metainterpreter of Prolog with freeze. It ensures, among other, the correct order of execution for the atoms of complex queries. We have used a Prolog like strategy of searching through the entire solution space, thus obtaining all the solutions for our query.

The resolution of the atomic queries is handled by methods specific to the ontology representation (API). The result of each such atomic query is asserted as a Prolog fact and further used by the solving algorithm.

The current implementation of POQL ensures queries' independence of the representation of queried ontologies. Furthermore, the system is subject to further developments, so that it may simultaneously query two distinct ontologies with different representations. This feature will eventually make possible the integration of an ontology merging theory [10]. Meanwhile, the user interaction is to be enriched with a NLI.

V. EXPERIMENTAL RESULTS

We chose to test POQL with ontologies built in Protégé 2000 [5], [9], a widespread knowledge base creational environment. Our decision was based on the fact that, among other, Protégé 2000 presents the advantage of integrating the OKBC model.

For the implementation, we have used two programming languages: Java and XSB Prolog. The reason for using Java is that Protégé is a Java based environment which provides an API for easy access to both the representation of the ontology and interface development. The implementation of the metainterpreter is written in XSB Prolog, which gives a direct mapping between the logical form of the query and the solving strategy.

In Appendix B are shown some results for three existing ontologies and various complexity queries, obtained on Athlon XP 1800+.

VI. CONCLUSIONS AND FURTHER DEVELOPMENT

We have described in this paper our approach regarding the development, implementation and usage of POQL, a new language for describing investigations of a portable knowledge base. It presents several advantages over other querying tools:

- it allows a query syntax that follows the Prolog logical form, therefore enabling the further development of interfaces that would communicate with our tool (i.e. Natural Language Interfaces, which permit users that are not familiar with the given ontologies to ask them queries.);
- the use of a Prolog meta-interpreter brings the possibility of generating complex queries; the solutions are computed taking advantage of the backtracking mechanism and postponing technique;
- queries are introduced in the interface in a Prolog like manner, without being restricted to certain patterns, which leads to increased flexibility in searching the solution space;
- queries are parsed before sending them to the Prolog resolution mechanism, thus eliminating syntactic errors and ensuring the use of frame names belonging to the knowledge base space, before calling the Prolog solver;
- it contains a user friendly interface, integrated in the Protégé environment which takes full advantage of the possibilities of the described query language.

POQL conforms to the Prolog logical form, thus being independent towards any specific knowledge base creational environment. Its independence renders it fit for usage in other such environments and for further developments of interfaces that would communicate with POQL. Our implementation allows further developments for ontologies merging theory support (by using axiom-predicate as defined in section 3).

ACKNOWLEDGEMENTS

This paper is part of the research joined project of DaimlerChrysler AG and Computer Science Department of Technical University of Cluj-Napoca.

REFERENCES

[1] H. Eriksson, R. W. Fergerson, Y. Shahar, & M. A. Musen. (1999). Automatic Generation of Ontology Editors. *Twelfth Banff Knowledge Acquisition for Knowledge-based systems Workshop, Banff, Alberta, Canada*

[2] H. G. Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, & J. Widom (1997). The TSIMMIS approach to mediation: Data models and Languages. *Journal of Intelligent Information Systems.*

[3] T. Muresan, R. Potolea, S. Muresan, (1998), Amalgamating CCP with Prolog, *Scientific Journal of Technical University Timisoara, Vol.43,57, no.4, 1998, Special Issue Dedicated to Third International Conference on Technical Informatics, CONTI'98, Romania*, pag.47 - 58

[4] M. A. Musen, R. W. Fergerson, W. E. Grosso, N. F. Noy, M. Crubezy, & J. H. Gennari. (2000). Component-Based Support for Building Knowledge-Acquisition Systems. *Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress (WCC 2000), Beijing, 2000.*

[5] N. F. Noy, R. W. Fergerson, & M. A. Musen. (2000). The knowledge model of Protege-2000: Combining interoperability and flexibility. *2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France*

[6] Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P. D. and Rice, J. P. (1998), OKBC: *A Programmatic Foundation for Knowledge Base Interoperability. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI – 98), Madison, Wisconsin, AAAI Press*

[7] N. F. Noy, M. A. Musen. An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support. *Sixteenth National Conference on Artificial Intelligence (AAAI-99), Workshop on Ontology Management, Orlando, FL, 1999.*

[8] Gruber, T.R. (1991). *A translation approach to portable ontology specifications. Knowledge Acquisition, 5*, 199-220.

[9] W. Grosso, H. Eriksson, R. Fergerson, J. Gennari, S. Tu and M. Musen. *Knowledge Modeling at the Millenium (The Design and Evolution of Protégé-2000), Stanford University.*

[10] N. F. Noy, M. A. Musen *Anchor-PROMPT: Using Non-Local Context for Semantic Matching* In the *Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, August 2001.

[11] T Janssen. *Compositionality*. In J. van Benthem and A. ter Meulen, editors, Handbook of Logic and Language,and Linguistics, pages 417-473. Elsevier Science, 1997.

APPENDIX A

SOUNDNESS FOR PROGRAMS WITH FREEZE

A SLD refutation procedure for a logic program P and a goal G uses a computation rule and a search strategy (rule). The computation rule chooses a subgoal from the sequence of goals to perform the derivation step. A SLD derivation is (said to be) *fair* if it ensure any subgoal selection in a finite number of steps (the depth first search strategy of Prolog is unfair).

The soundness and completeness of a fair SLD refutation has been proved, that is the equivalence between the logical consequence $(P \models G)$ and SLD refutation $(P \vdash G)$.

$$(P \models G) \leftrightarrow (P \vdash G)$$

If the negation as failure is taken into consideration, a SLDNF computation rule is said to be *safe* if it selects only ground negative literals and it does not interrupt the coresponding SLDNF finite failure subtree building.

If $comp(P)$ is the Clark completion of a program P, and the SLDNF rule of computation is safe, the soundness and completeness of SLDNF refutation hold true.

$$(comp(P) \models G) \leftrightarrow (P \vdash G)$$

Moreover, the soundness and completeness of a SLD refutation is independent of the chosen computation rule (e.g. the current subgoal selection).

In this paper we consider P_f the logic program obtained from P, by enclosing any subgoal G_i of a clause into a $freeze(Var,G_i)$ predicate, where Var belongs to the set of G_i variable, $Var \in SetVar(G_i)$.

If

$$H:- B_1,..,B_i ,..,B_n \in P,$$

then

$$H:-B_1,..,freeze(Var, B_i),..,B_n \in P_f .$$

A subgoal $freeze(Var, B_i)$ is not selected as long as Var is unbound. On it's selection the equivalence

$$freeze(Var, B_i) \leftrightarrow B_i$$

takes place.

A fair SLD refutation for a program P_f and a goal G is achieved if the empty clause may be derived in a finite number of steps. This means that all the subgoals $freeze(Var, B_i)$ have been actually selected. Taking into consideration the independence of the choice of the computation rule and the logic equivalence between $freeze(Var, B_i)$ and B_i in the moment of the selection, we have:

Lemma :

$$(P_f \vdash G) \rightarrow (P \vdash G)$$

and

Corollary: (Soundness of SLD refutation for programs with freeze.)

$$(P_f \vdash G) \rightarrow (P \models G)$$

The soundness of P_f programs makes possible the use of Prolog with freeze as target language for queries interpretation.

Even if the Prolog strategy is an unfair and incomplete one, freeze does not introduce new exceptions from the theoretical model (comparing with those of standard Prolog). Moreover, freeze may improve the Prolog programs behaviour, making safe the negative literals selection (safety of SLDNF refutation). However, the Prolog strategy makes incompatible the use of freeze together with the cut ('!'), without imposing special restrictions.

APPENDIX B

EXPERIMENTAL RESULTS

Ontology & Query	No. of Atomic Queries	No. of Variables	No. of Solutions	Time (s)
I. Private Ontology: 331 frames				
?A sub 'EveryThing', 'smallDieselMotor-1' : ?A	2	1	1	0.045
?A sub 'EveryThing', 'smallDieselMotor-1' : ?A, ?B : 'EveryThing'	3	2	6	0.065
(?A sub 'EveryThing', 'smallDieselMotor-1' : ?A) , (?B = 'drive' . ':hasPart')	3	2	1	0.040
?A sub 'EveryThing', ?B sub ?A, ?C sub ?B, ?D sub ?C, 'SaloonBody-1' isa ?D	5	4	1	0.290
?A :: ':DCX_CF_SYSTEM_CLASS', ?B :: ':DCX_CF_SYSTEM_CLASS', ?A \==?B, ?C sub ?A, ?D sub ?B	5	4	10	2.450
?A :: ':DCX_CF_SYSTEM_CLASS', ?B :: ':DCX_CF_SYSTEM_CLASS', ?A \==?B, ?C sub ?A, ?D sub ?B, ?A sub ?E, ?B sub ?E	7	5	1	4.950
II. Newspaper-querie Ontology: 187 frames (http://protege.stanford.edu/ontologies.html)				
?A : 'Employee', ?B = ?A.'responsible_for'.'current_job_title', ?B =="sports reporter"	3	2	1	0.110
?A=?_B.'sections', ?_B=?C.'responsible_for', ?C isa 'Editor'	3	2	4	0.060
(?A isa 'Personals_Ad'; (?B isa 'Article', ?B.'containing_section'=?C, ?C.'section_name'\=="Lifestyle")), ?B.'published_in'=?A, ?A.'number_of_pages' >35	4	3	4	0.220
III. Organizational_Model Ontology: 163 frames (http://protege.stanford.edu/ontologies.html)				
?A sub 'Organizational_Model_Entity', ?A.'participant_name'	2	1	5	0.040
?A sub 'Diagram_Entity', ?B isa ?A, ?B.'upper_left_corner' =?C, ?C \==[]	4	3	8	0.261
?A isa 'Point', ?A.'x'=130, ?A.'y'=?B, ?C isa 'ObjectLocation', ?C.'location'=?D, (?D.'lower_right_corner'=?A ; ?D . 'upper_left_corner '=?A)	7	4	2	0.250
IV. REA sample: 138 frames (http://protege.stanford.edu/ontologies.html)				
?A sub 'ExchangeElement', ?B isa ?A, ?B.'association'=?C, ?C : ?D	4	4	30	0.380
?A isa 'AgentType', ?A.'participates'.'classifies'=?B	2	2	4	0.050
?A isa 'Agent', ((?A.'custody'=?B, ?B isa ?C);(?A .'association'=?B, ?B isa ?C))	5	3	8	0.110
?A sub ':THING', ?B sub ?A, ?C sub ?B, ?D sub ?C, 'REA_00002' isa ?D	5	4	1	0.570