N95-19631

EXPERT SYSTEM SHELL TO REASON ON LARGE AMOUNTS OF DATA

G. Giuffrida
Computer Science Department
The University of California Los Angeles
giovanni@cs.ucla.edu

Abstract

The current DBMSs do not provide a sophisticated environment to develop rule based expert system applications. Some of the new DBMSs come along with some sort of rule mechanism, these are *active* and *deductive* database systems. However, both of these are not featured enough to support full implementation based on rules. On the other hand, current expert system shells do not provide any link with external databases. That is, all the data are kept into the system working memory. Such working memory is maintained in main memory. For some applications the reduced size of the available working memory could represent a constraint for the development. Typically these are applications which require reasoning on huge amounts of data. All these data do not fit into the computer main memory. Moreover, in some cases these data can be already available in some database systems and continuously updated while the expert system is running.
This paper proposes an architecture which employs knowledge discovering techniques to reduce the amount of data to be stored in the main memory, in this architecture a standard DBMS is coupled with a rule-based language. The data are stored into the DBMS. An interface between the two systems is responsible for inducing knowledge from the set of relations. Such induced knowledge is then transferred to the rule-based language working memory.

INTRODUCTION

Current trends in database research area are about making smarter and more friendly current DBMS. Traditionally, a DBMS is mostly a repository of information to be later retrieved by an ad-hoc query language. If some data are currently unavailable in the system, or the query is not properly issued, the system can return either an error or a non-answer. This is not really the case when a specific question is asked to a human expert, in this case the expert digs in his or her memory looking for an answer, if no proper answer is found the expert perhaps starts looking for alternative, and approximate, responses. Actually, the expert uses his or her knowledge in a sort of "cooperative way" trying to make the user as more satisfied as possible. To answer such a query the expert employs his or her knowledge in a much more productive way than performing a simple linear scan over the known set of tuples in his or her mind: **meta-knowledge** was fruitfully employed in order to formulate the answer.

In an artificial system such meta-knowledge can be known a priori (read: Tied up to the system at database design time) or induced by an already available set of relations. The purpose of *induced knowledge* [Qu79] [CLC91] is to describe the current state of the database by means of sets of rules; a rule induction algorithm is responsible to *discover* such induced knowledge. Basically, induced rules are oriented to summarize the way the different attributes are related to each other. Artificial reasoning can be built on top of this meta-knowledge.

This paper proposes an architecture which integrates two public domain systems, namely Postgres and CLIPS, into a unique environment. The proposed architecture offers a shell for developing expert systems able to work on large amounts of data. Database relations are stored on Postgres while the artificial expert is implemented on top of CLIPS. The artificial expert can specialize the rule induction algorithm in order to focus the induction only on the necessary data.

Postgres comes along with a rule system to implement active database features. By the way, implementing an expert system using only Postgres has some drawbacks. The Postgres rule mechanism has been designed for triggering over updates on the database. We always need to perform a secondary storage update in order to fire a Postgres rule. This can be too expensive in many cases when a little fact stored in main memory can perform the same task. Moreover, the rule system in Postgres does not implement all the facilities of a special purpose rule-based language such CLIPS.

On the other side CLIPS does not provide any access to an external database. This can cause serious limitations in designing expert systems reasoning on large amounts of data stored on a database.

CURRENT SYSTEMS

Latest developments in databases are oriented to enrich traditional systems with some sort of rule-based mechanism. Novel additional features are so added to traditional DBMS by means of rules. New classes of systems have appeared on the market, they are: *Active* [St92] [HCL+90] [MD92] and *Deductive* [Mi88] [RSS92] [NT89] databases. The following sections give some details about these new systems. The suitability of those as expert system shell is also investigated. Advantages and drawbacks for each system are drawn. Also, the limitations of current rule-based languages as developing shell for expert system oriented to reason on large amounts of data are discussed.

Active Database

More and more active database systems are everyday appearing on the market. The transition of such system from research stages to the market was fast. They materialized in systems like Sybase [MD92], Postgres [St87], Starburst [HCL+90], and more. However, the active feature of databases is still under investigation. Additional research still needs to be done.

The active feature extends the traditional systems with a rule-based mechanism oriented to allow implementation of *demons*. The purpose of these demons is basically to overview operations on the database and, when certain conditions are satisfied, invoke a corrective action over the database. The system can so be extended with sort of animated entities; data integrity can be enforced by means of active rules.

Typically an active rule has the following format:

```
on <event>
if <condition>
then <action>
```

This rule model is also referred as E-C-A (Event-Condition-Action) rule. Simpler rule model can utilize an E-A pattern. **event** is one of the possible operations oriented to tuple handling;

typical events are: *insert*, *delete*, *update* or, *retrieve*. The specified **condition** is a condition which should hold on the data involved in the current event in order to fire the rule. **action** is the set of actions that take place when the specified condition is satisfied.

In the following a Postgres rule is reported as example of active rule (copied from [Post]):

```
define rule r1 is
  on replace to EMP.salary
     where current.name = 'Joe'
  do replace EMP (salary = new.salary)
     where EMP.name = 'Sam'
```

Rule r1 will fire every time the salary of Joe is updated. For such update the salary of Sam will be set to the same value of the new salary of Joe. In other words, Sam will always get the same salary of Joe. Note that the inverse is not true, the same rule will not take care to update Joe's salary when Sam's salary is replaced by some new value.

Evaluation of database active feature is based upon (1) the domains of possible events which can be monitored; (2) the formalism to specify the condition; (3) the language to execute actions. Other features, like rule execution model, have also to be considered for a sound evaluation. Postgres [St87], is actually one of the most advanced active databases currently available.

Active database systems present some drawbacks as expert system shells. The most significant ones are hereafter listed:

- Firing only upon database events, no "Working memory" is available. All the data are kept in the secondary memory storage. Main memory usage might, in several cases, be preferable to a secondary memory one.
- The language to specify actions is usually restricted to the database domain. Typical rule-based languages allow on the right-hand side complex operations (print, while loop, user inputs, etc.) to be performed.
- Not well defined rule execution model is available.

Other criticisms to these systems are about the rule redundancy required in some cases. A set of sort of complementary rules might be defined in order to ensure some data integrity. For instance, let us suppose we want to make sure that each employee works for one and only one department. A set of rules needs to be defined in order to enforce this constraint: One will take care when a new employee is hired, another when the employee is transferred, another when a department is dropped, and so on. This is not really the case of an equivalent rule-based implementation where a single rule can handle all the cases. The different behavior between active DBMS and RBL is actually based on different architectures. In an active database system the rules are tied up to the database operations, so a trigger must be defined for each possible operation on the interested data. In a rule-based system, rules are fired straight from the data in the working memory. The operations which update the data are hidden to the triggered rules. This model reduces the number of required rules. Moreover, the maintenance cost for a set of rules is of course higher than the one for a single rule.

The previous considerations should convince the reader that implementing an expert system on top of an active database is not a straightforward task; they lack the power for that purpose.

Deductive Databases

Another challenging and relatively new research direction in database area is about *Deductive databases*. These systems are essentially based on a Prolog-like development language. Some of these are LDL++ [NT89], NAIL! [MUVG86] and CORAL [RSS92]. Here facts can be either stored on main memory or secondary memory storage. However, in the author's opinion, deductive databases are still not at a stable stage. Further investigation needs to be spent on those. This belief is also enforced from the lack of any commercial deductive database system on the market at the time of this writing.

Development of deductive databases raised several both theoretical and practical issues that are still looking for answers. Research is still in progress in this direction.

Some of the most significant points of deductive systems are:

- Prolog-like rule-based development environment;
- Recursive rule invocation allowed;
- Easily definable virtual relations (*views* in database jargon);
- Hybrid forward/backward chaining rule execution model. The method chosen depends on the current task to be performed. It is chosen automatically by the system itself. The user does not have any responsibility on that;
- Restricted usage of negation. Only certain classes of negation (namely *stratified*) are allowed. Basically, no negation on recursive calls can be used.
- Ability of interfacing with already available DBMSs.

Deductive databases are definitively a proper environment for developing expert reasoning on large amounts of knowledge. However, as already said, they still need some time before reaching a steady stage.

Expert System Shells

Somewhat current expert systems shells present the opposite problem of the previously discussed database systems. In this case all the facts are kept into the system working memory. No connection with any external storage system is provided. This can be a constraint difficult to be taken around when huge amount of knowledge is employed as basis for the reasoning. In this case the main memory cannot be big enough to accommodate this large number of tuples. In another possible scenario, reasoning has to be developed on data which are already stored on some database systems. Perhaps these data are still evolving while the artificial reasoning is performed. The only possible solution to this problem is represented by a periodic transfer of the new updated data from the database to the expert system shell.

Current solutions to this problem are oriented to create an ad-hoc interface between an expert system shell and a DBMS. Such solutions are often properly tailored for the application itself. They do not provide a generic solution to the problem.

KNOWLEDGE INDUCTION ALGORITHM

The process of knowledge induction [Qu79] [CL90] aims to build useful semantic data starting from the available relations in a database. In this section, an overview of the knowledge induction algorithm is given while keeping an eye open on our specific implementation.

Somehow the knowledge induction process aims to "capture" the semantics of the stored data and model it by means of *induced rules*. Different forms of rule may exist, we are mainly interested in rules which correlate two attributes between themselves. The correlation of interest for us is the definition of a domain for the first attribute which identifies a unique value for the second attribute. We are interested to the *range form* which is: "*if* $x_1 \leq X \leq x_2$ *then* $Y = y$ *with Probability P and Coverage C.*" The **probability** value expresses the *certainty degree* of the given rule while the **coverage** refers to the number of tuples used to build such rule. Additional details can be found in [Qu79], [CL90] and [CLC91]. Several different forms and attributes can be combined to express induced rules.

The set of induced rules represents part of the knowledge about the given databases. Storing these rules most probably requires less space than the corresponding table format. Induced rules have to change dynamically reflecting the database updates. Every update operation on the database has to affect the set of induced rules accordingly.

In the proposed system induced rules are stored as CLIPS facts whose form is:

```
(ik <arg1> <lowerBound> <upperBound> <arg2> <value> <prob> <cover>)
```

The first atom is to classify the set of facts describing the induced knowledge. The remaining atoms are the information about the specific induced rule. The set of all these facts forms the induced knowledge. Semantics of most of the arguments is straightforward. **prob** is the probability factor and **cover** is the number of tuples covered by the rule (look at [Ch94] for a complete description of these parameters.) For instance, the rule "*if 62000 ≤ SALARY ≤ 85000 then TYPE = 'MANAGER' with Probability=0.6, Coverage=3*" is stored in the following CLIPS fact:

```
(ik SALARY 62000 85000 TYPE MANAGER 0.6 3)
```

As an example, let us suppose to have the following table, namely *emp*, in our Postgres database:

EMP

NAME	TYPE	SALARY
John	STAFF	30000
Mary	MANAGER	62000
Eva	STAFF	32500
Bob	MANAGER	85000
Mark	MANAGER	76000
Judy	MANAGER	83000
Kia	STAFF	56000
Tom	MANAGER	50000
Phil	STAFF	54000

By applying the induction algorithm on these tuples the set of CLIPS facts will be:

```
(ik emp.SALARY 62000 85000 emp.TYPE MANAGER 0.8 4)
(ik emp.SALARY 50000 50000 emp.TYPE MANAGER 0.2 1)
(ik emp.SALARY 30000 32500 emp.TYPE STAFF 0.5 2)
(ik emp.SALARY 54000 56000 emp.TYPE STAFF 0.5 2)
```

The induced rule schema is only a first attempt which is used more as framework for this paper. A real design should take into more consideration other parameters to be included in the schema.

The induced rules need to be dynamically updated reflecting operations performed over the database. For instance, let us suppose the following new tuple is inserted into the system:

```
Betty  MANAGER  60000
```

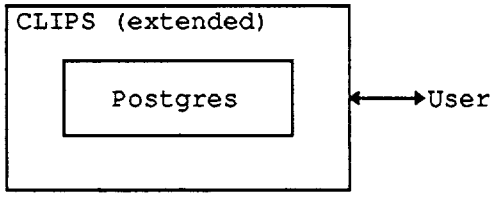
The knowledge base has to be updated to the following set of facts:

```
(ik emp.SALARY 60000 85000 emp.TYPE MANAGER 0.83 5)
(ik emp.SALARY 50000 50000 emp.TYPE MANAGER 0.16 1)
(ik emp.SALARY 30000 32500 emp.TYPE STAFF 0.5 2)
(ik emp.SALARY 54000 56000 emp.TYPE STAFF 0.5 2)
```

The new tuple affected the values in the first fact, the lower bound, probability and coverage values changed accordingly. Similar changes will take place in case of deletion or updating.

PROPOSED SYSTEM OVERVIEW

In the proposed system architecture, Postgres and CLIPS are being integrated. The interface between them represents the key point for the design. An expert system can be developed on top of CLIPS. This latter is extended with features to make available summarized data stored in Postgres. From an expert system shell's point of view the new system can be seen as in the following figure:



Postgres presence is actually hidden to the end-user. The application built on top of CLIPS takes advantage of the meta-knowledge induced from the stored database. The expert system being developed on CLIPS has to be aware of the following:

- Database schema;
- Database statistics: number of tuples, number of induced rules, rule coverage, popularity, probability, etc.

CLIPS has to be extended with dedicated constructs to inquire for such information. Depending on both the database information and the application being implemented, the meta-knowledge extraction process is properly tailored and executed. From CLIPS several actions can be taken in order to model the induction algorithm to best fit the current application.

The knowledge induction process executes in a dynamic fashion, that is, once the rule schema has been defined, any update on the monitored Postgres relations will be reflected on the induced rules. The rule induction manager then propagates these induced rules to the CLIPS working memory in the fact form previously discussed. Eventually, these facts trigger some CLIPS rules.

The interface between CLIPS and Postgres then can be defined by the following set of functions available on the CLIPS side:

- Inquire for the database schema;
- Inquire for database statistics;
- New rule induction schema definition: $R_1.X \rightarrow R_2.Y$, where R_1 and R_2 are relations while X and Y are attributes;
- Modify rule induction parameters (cut-off factor, etc.);
- Retrieve current rule induction schema;
- Remove induced rule;

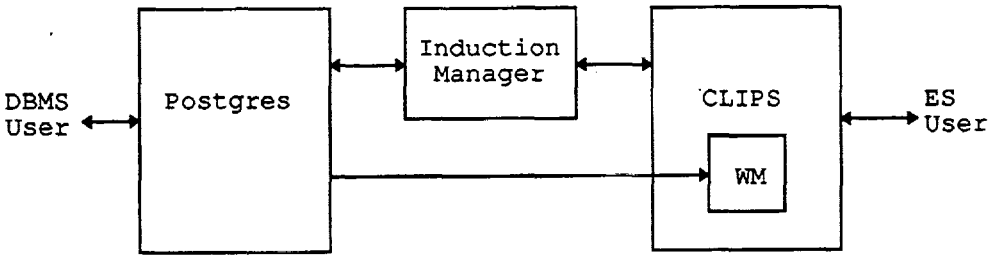
The system extends the set of built-in CLIPS functions to include these new services.

DESIGN

The system will be presented to the expert system developer as a more featured shell which includes ability to access summarized data. The key points of all design are essentially the following:

- Interface between CLIPS and Postgres;
- Induced knowledge management.

The system architecture is the following:



The *Induction Manager* is responsible for the communication between CLIPS and Postgres. CLIPS utilizes services offered by this module to tailor the induction algorithm to its needs. Responses from Postgres to CLIPS are also passed through the induction manager. The link between Postgres and WM (the CLIPS working memory) stands for a straight access from Postgres to CLIPS working memory. That is, once the rule schema has been defined, updates on Postgres will result in updates on the induced rules. These rules are then written straight to CLIPS' working memory without requiring the intervention of the induction manager. In other words, the induction manager can be seen as a set of function to be invoked from CLIPS, while

the link from Postgres to WM represents an asynchronous flow of data toward CLIPS. Once the rule induction schema has been defined, this flow of data will be dynamically maintained, that is, any update to any monitored Postgres relation can cause some rule induction instances updates, these will be promptly propagated to CLIPS through this link. As better explained in the following, actually the induction manager is scattered between Postgres and CLIPS. Some Postgres triggers and CLIPS rules (together with some external Postgres procedures) form the complete block.

The entire system works in a dynamic fashion. That is, at the same time the CLIPS application is running people can keep on working on Postgres. Postgres updates affect the set of induced rules on the CLIPS side. Such updates can trigger some CLIPS rules to execution. Under this light the entire system can be seen as an extension of the rule-based mechanism of Postgres: Some Postgres rule activations eventually trigger some CLIPS rules.

CONCLUSIONS

This paper discussed the design of a system where expert shell techniques are combined together with knowledge discovering techniques. Two public domain systems, namely CLIPS and Postgres, have been combined into a unique one. Purpose of this design is to provide an expert system shell with ability to reason over a large amount of data. Current expert system shells lack the ability of accessing data stored on external devices. On the other side database technology is not yet well refined to provide a featuring shell to develop expert systems. Proper combinations of these techniques may lead to interesting results.

The possible applications for such a system are those where reasoning on large volume of data are required. For instance, think about the complexity of reasoning on the millions of customers of a phone company or a frequent flyer program, in this case the complexity is due to the enormous amount of data to reason on.

REFERENCES

[Ch94] W.W.Chu, "Class notes CS244-Spring 1994", University of California in Los Angeles, 1994.

[CL90] W.W.Chu, R.Lee, "Semantic Query Processing Via Database Restructuring," Proceedings from the 8th International Congress of Cybernetics and Systems, 1990.

[CLC91] W.W.Chu, R.Lee, Q.Chen, "Using Type Inference and Induced Rules to Provide Intensional Answers," Proceedings of the 7th International Conference on Data Engineering, 1991.

[Fo82] C.L.Forgy, "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence 19, 1982.

[Gi89] J.C.Giarratano, "CLIPS User's guide," Artificial Intelligence Section, Lyndon B. Johnson Space Center, June 1989.

[HCL+90] L.Haas, W.Chang, G.M.Lohman, J.McPherson, P.F.Wilms, G.Lapis, B.Lindsay, H.Pirahesh, M.Carey, E.Shekita, "Starburst midflight: as the dust clears," IEEE Transactions on Knowledge and Data Engineering, March 1990.

[MD92] D.McGoveran, C.J.Date, "A Guide to SYBASE and SQL Server," Addison-Wesley Publishing Company, 1992.

[Mi88] J.Minker "Foundation of Deductive Databases and Logic Programming," Morgan-Kaufmann, Los Altos, CA, 1988.

[MUVG86] K.Morris, J.Ullman, A.Van Gelder, "Design overview of the NAIL! system," proceedings of the 3rd Int. Conference on Logic Programming, Springer-Verlag LNCS 225, New York, 1986.

[NT89] S.Naqvi, S.Tsur, "A Logical Language for Data and Knowledge Bases," Computer Science Press, 1989.

[RSS92] R.Ramakrishan, D.Srivastava, S.Sudarshan, "CORAL: A Deductive Database Programming Language," Proc. VLDB '92 Int. Conf. 1992.

[Post] J. Rhein, G. Kemnitz, POSTGRES User Group, The POSTGRES User Manual, University of California, Berkeley.

[St87] M. Stonebraker, "The POSTGRES Storage System," Proc. 1987 VLDB Conference, Brighton, England, Sept. 1987.

[St92] M. Stonebraker, "The Integration of Rule Systems and Database Systems," IEEE Transactions on Knowledge and Data Engineering, October 1992.

[Qu79] Quinlian, J.R., "Induction Over Large Data Bases," STAN-CS-79-739, Stanford University, 1979.