An SQL Query Generator for CLIPS

James Snyder and Laurian Chirica

CAD Research Unit
California Polytechnic State University
San Luis Obispo, Ca.

ABSTRACT

As expert systems become more widely used, their access to large amounts of external information becomes increasingly important. This information exists in several forms such as statistics, tabular data, knowledge gained by experts and large databases of information maintained by companies. Because many expert systems, including CLIPS, do not provide access to this external information, much of the usefulness of expert systems is left untapped. The scope of this paper is to describe a database extension for the CLIPS expert system shell.

The current industry standard database language is SQL. Due to SQL standardization, large amounts of information stored on various computers, potentially at different locations, will be more easily accessible. Expert systems should be able to directly access these existing databases rather than requiring information to be re-entered into the expert system environment. The ORACLE relational database management system (RDBMS) was used to provide a database connection within the CLIPS environment.

To facilitate relational database access, a query generation system was developed as a CLIPS user-function. The queries are entered in a CLIPS-like syntax and are passed to the query generator, which constructs and submits for execution, an SQL query to the ORACLE RDBMS. The query results are asserted as CLIPS facts.

The query generator was developed primarily for use within the ICADS project (Intelligent Computer Aided Design System) currently being developed by the CAD Research Unit in the California Polytechnic State University (Cal Poly). In ICADS, there are several parallel or distributed expert systems accessing a common knowledge base of facts. Each expert system has a narrow domain of interest and therefore needs only certain portions of the information. The query generator provides a common method of accessing this information and allows the expert system to specify what data is needed without specifying how to retrieve it.

Dr. Laurian Chirica is a Professor of Computer Science; James Snyder is a student in the Computer Science Department at the California Polytechnic State University, San Luis Obispo, California.

INTRODUCTION

Currently, Cal Poly's CAD Research Unit is developing an Intelligent Computer Aided Design System (ICADS). This system is composed of several domain expert systems running concurrently under the control of a blackboard [Pohl, Myers, Chapman, Cotton, 1989]. The current application area under development is architecture, but the system's applicability can be easily extended to other disciplines. In order for the domain expert systems to evaluate a design, a large amount of information needs to be available to the expert systems. This body of information does not remain static and therefore needs a management system. In addition, there are two major classes of information needed by an expert system: reference information and prototype information. Reference information can be described as tabular information such as a parts catalog. Each part has an identifier, a description, and a price. Another example of reference information is thermal lag times for various construction materials.

Prototype information comes from a knowledge representation scheme called Prototypical Information [Gero, Maher, Zhang, 1988]. A prototype describes the general characteristics that most objects have. For example, the ICADS project uses a Building Type Prototype Database. This database stores information about typical high-rise apartments. Some of the kinds of information stored are: owner goals and objectives, user group profiles, and designer criteria.

Prototypical information has a very complex structure unlike reference information. Complex retrieval methods are necessary for certain information and application programmers should not be concerned with the details of retrieving information. Not only is prototype and reference information needed within expert systems, but it is needed in other environments as well, such as C programs. Because common information is needed in disjoint environments, a common storage mechanism is needed, namely a Data Base Management System (DBMS). A DBMS provides a recoverable and concurrent method of storage and retrieval of data. These features are very necessary within the ICADS project because there are many independent expert systems executing, all of which could access the database. Figure 1 shows the ICADS system architecture and how expert systems, which we refer to as Intelligent Design Tools (IDT), need access to database information.

The current DBMS of choice is the Relational DBMS (RDBMS). Because of its simplicity and power, it has become the DBMS standard. RDBMSs use a Fourth Generation Language (4GL) or query language to perform operations on database objects [Korth, Silberschatz, 1986]. The *de facto* standard 4GL is the Structured Query Language (SQL). This query language is available on most hardware platforms and operating systems from PCs to super-computers.

PROBLEM DEFINITION AND REQUIREMENTS

The ICADS project uses CLIPS as its expert system shell, which in its standard version does not support RDBMS access. Because an RDBMS provides a common storage and retrieval method, we decided that relational database access within CLIPS was necessary. The solution had several general requirements which needed to be met to be useful. They were:

- To allow the use of the standard RDBMS features.
- To be easy for the expert system developer to use.
- To allow for easy integration into the CLIPS source code.

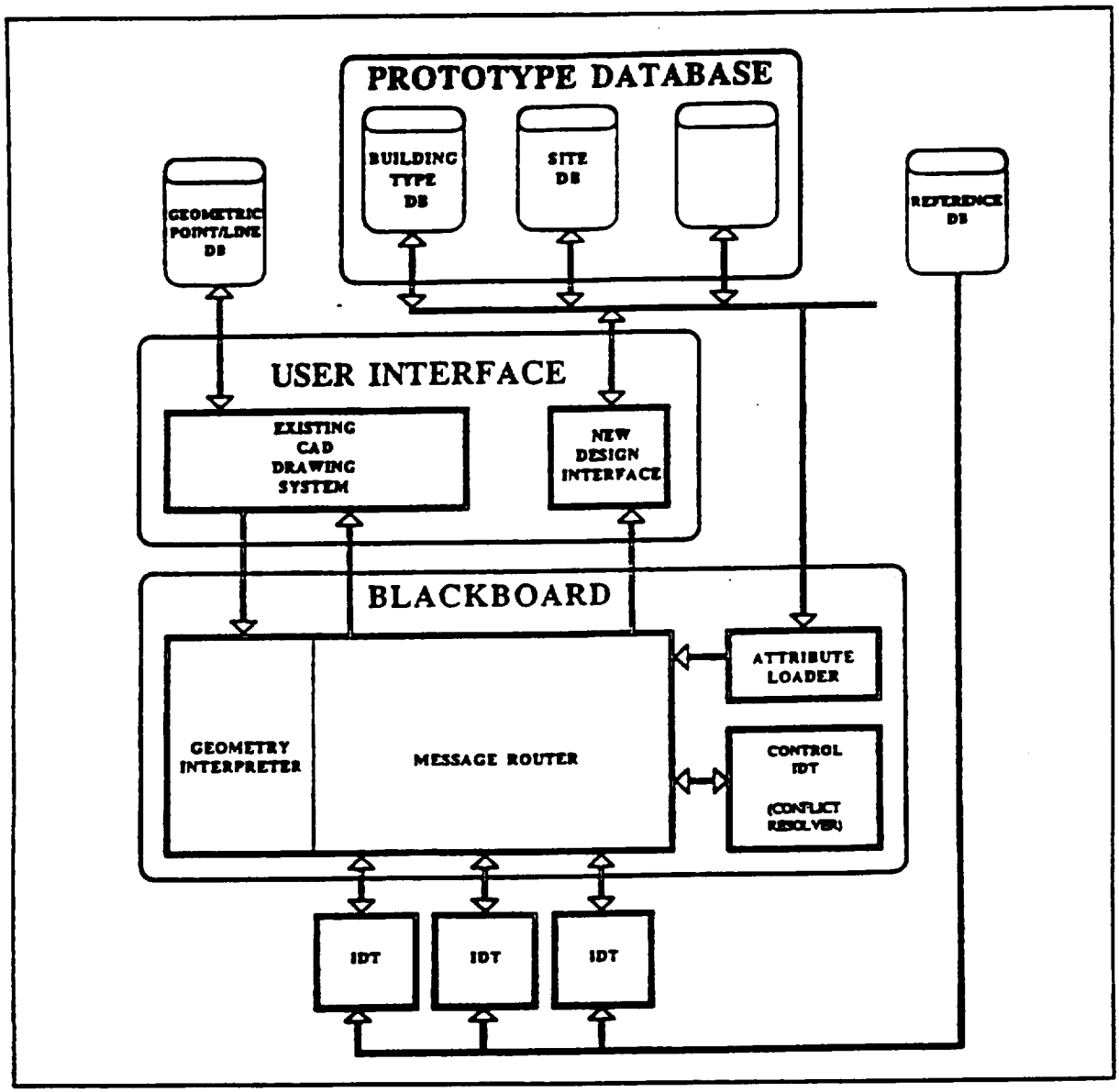

Figure 1 - ICADS system architecture

The CLIPS database access system needed to allow for most of the database queries possible. Figure 2 illustrates the general form of an SQL query. The SELECT clause defines the particular attributes to be retrieved. For example, the description of a part would be listed in this clause. The FROM clause defines which database relations the information is to come from. In this scheme, data can come from multiple relations in a single query. A user retrieves only the information needed from each relation.

The WHERE clause defines constraints under which data is retrieved. For example, only the employees in department 10 should be considered. In addition, the WHERE clause can contain a join condition which tells the database system that a join between two or more relations needs to be executed. For example, an attribute of each employee is the department number they belong to. You want a list of all employees and the name of the department they work in. This information is not contained in one

```
(getsql query1 employee.name department.name
       = employee.d_no department.d_no
       = department.d_no 10
)
```

Figure 4 - Sample CLIPS SQL query

addition, each row from the database that is returned is prefixed with the user's query label and asserted as CLIPS fact by calling a C function provided by CLIPS.

The translation from the CLIPS syntax to SQL is very natural. The SELECT clause is obtained from the *<column-list>* previously defined. The FROM clause is obtained by building a list of relation names from the *<column-list>* relation names. The WHERE clause is obtained from the *<condition-list>* defined above. The operator and the first value are inverted to conform to the SQL syntax. The ORDER BY clause is implicitly built by the *<column-list>*. The data will be sorted based on the order in which the columns were listed.

SAMPLE APPLICATIONS

The Attribute Loader

In ICADS, the Attribute Loader is a special expert system which reads information from the database and asserts it into the semantic network. The ICADS project uses a frame-based representation to store information within its expert systems [Pohl, Myers, Chapman, Cotton, 1989]. The primary function of the Attribute Loader is to read the information from the database, assemble frames and assert them as facts.

The information is obtained from the ICADS Prototype Database, which contains information about typical building types and typical site locations. The structure of the prototype database is shown in Figure 4. The boxed items represent base relations; the circles represent relationship relations between base relations. Currently, objects, attributes, and values are retrieved from the database and asserted. The query generator allows the expert system source to remain constant even if a new database management system is used.

DBRESOLVE

Because we use frame-based knowledge, expert systems have frames as patterns in rules. Some frames can be quite complicated and can contain typographical errors. To increase programmer productivity, we created a program which resolves frames in an expert system with the information contained within the database. The DBRESOLVE programs function is very similar to a cross-referencing tool but is applied to frames.

The DBRESOLVE program scans the expert system source and identifies the occurrences of frame information which needs to be verified. This information is then checked against the database information. Any frames which are not contained in the database are flagged as incorrect.

place. It resides in two relations: employees and departments. A join condition specifies that the employee's department number must match a department number in the department relation.

The ORDER BY clause sorts the data in a specific ordering. If this clause is not specified, the data is returned in a system-dependent order which may not remain constant over time.

```
SELECT      <list of column names>
FROM        <list of relation names>
WHERE       <list of boolean conditions>
ORDER BY    <list of columns names>
```

Figure 2 - General form of an SQL query

From the point of view of an expert system developer, the database access should be intuitive and easy to use. The ideal solution would allow the user to specify the desired information in a CLIPS-like syntax. This considerably reduces the learning curve of the database access system.

Placing the database access system within the CLIPS environment should be as simple as adding any other user-defined CLIPS function. The database access system should be as small and fast as possible.

AN SQL QUERY GENERATOR FOR CLIPS

Our solution to the above problem was to develop a query generator for CLIPS. The function of the query generator is to take a CLIPS database query within a rule, translate it into SQL and submit the query for execution. The results of the query are then asserted as CLIPS facts.

The implementation of the query generator can be divided into three areas: the CLIPS interface syntax, the SQL interface, and the translation process. The general CLIPS syntax is defined in Figure 3. A *<label>* defines a unique label to prefix the facts when they are asserted. A *<column-list>* is a list of columns prefixed with a relation name to eliminate any ambiguous references to columns. For example, two different relations may have a column named "description". There must be a way to differentiate between each column, so they are prefixed with their relation names. A *<condition-list>* is a relational operator followed by constants or column names.

```
(getsql <label> <column-list> <condition-list>)
```

Figure 3 - General CLIPS - SQL syntax

Figure 4 shows a complete example. The query label is "query1". The employee names and department numbers from department 10 will be asserted as facts. Notice the join condition between the employee and department relations.

The SQL interface is invisible to a CLIPS user. The interface pertains only to the RDBMS that is used. In the ICADS system, we used the Embedded SQL option which allows C programs to submit queries for execution [Korth, Silberschatz, 1986]. Embedded SQL naturally falls in line with CLIPS, which is also written in C. We designed the system to take any SQL query and submit it for execution. In

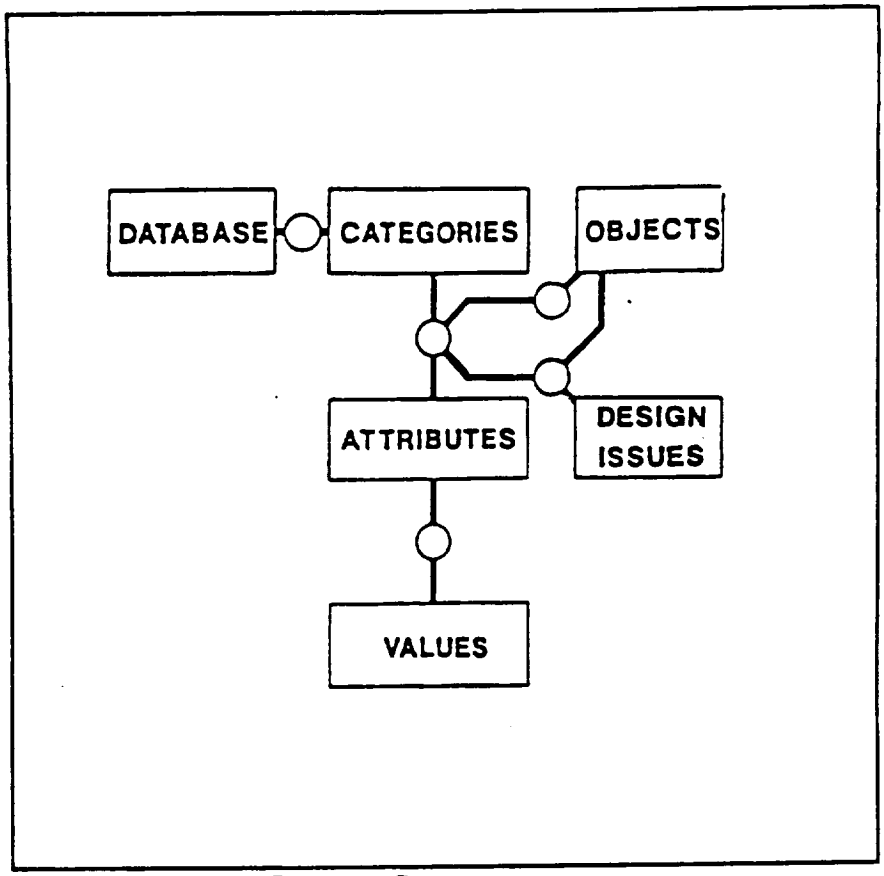

Figure 5 - Prototype Database Structure

CONCLUSIONS

Our initial hopes for decreased development time were easily met. Because of system's simplicity, queries can be easily written. Using a RDBMS allows the application programmer to only retrieve the information they need, which is much better than storing information in hard coded facts or reading information from disk files. In addition, other environments can access the same information.

Initially, we had concerns that query times would be too large. This proved to be quite the opposite. Because of the buffer management of the RDBMS, many queries execute faster than if the same information were read from a disk file.

Perhaps the most important feature of using a RDBMS is the concurrency, integrity and reusability of data in many orthogonal environments. Within ICADS, many programs and expert systems access the same relations. If any of the data within a relation changes, every system which accesses it retrieves the current and correctly updated values. Concurrency and integrity control would be extremely complicated to add to CLIPS, but it comes automatically by using a RDBMS.

The above factors make an RDBMS a superior method of information storage and retrieval. We have not yet encountered any drawbacks to using this approach.

BIBLIOGRAPHY

[Gero, Maher, Zhang 1988] Gero,J., M. Maher and W. Zhang; 'Chunking Structural Design Knowledge as Prototypes'; Working Paper, Architectural Computing Unit, Department of Architectural Science, University of Sydney, Australia, January, 1988.

[Korth and Silberschatz 1986] Korth H.F. and A. Silberschatz; 'Database System Concepts'; McGraw-Hill, 1986.

[Pohl, Myers, Chapman, Cotton 1989] Pohl, J., L. Myers, A. Chapman, J. Cotton; 'ICADS: Working Model Version 1'; Technical Report, CADRU-03-89, CAD Research Unit, Design Institute, Cal Poly, San Luis Obispo, Calif., USA, December, 1989.