Efficient Integration of PROLOG and Relational Databases in the NED Intelligent Information System

F. Maier, D. Nute, W. D. Potter, J. Wang, M. Dass, H. Uchiyama
Artificial Intelligence Center
The University of Georgia
Athens, GA, U.S.A

M. Twery, P. Knopp, S. Thomasma
Northeastern Research Station
USDA Forest Service
Burlington, VT, U.S.A

H. M. Rauscher
Bent Creek Experimental Forest
Southern Research Station, USDA Forest Service
Asheville, NC, U.S.A

Abstract
NED-2 is an intelligent information system for ecosystem management currently in development by the USDA Forest Service. Using PROLOG knowledge bases and inference engines, NED-2 evaluates forest inventories according to a set of predefined goals. It is a blackboard system with agents implemented in PROLOG. NED-2's primary data store is a set of relational databases. The present paper focuses upon an issue of central importance to the project—the integration of PROLOG programs and relational databases to form NED-2's blackboard.

Keywords: Intelligent Information System, Decision Support System, PROLOG, Relational Database System, Blackboard System

1 Introduction

NED-2 is a software system currently in development by the USDA Forest Service (in conjunction with the University of Georgia Artificial Intelligence Center) to facilitate ecosystem management [6, 7, 10]. NED-2 allows for the analysis of forest inventories to determine the degree to which they satisfy goals pertaining to timber production, water quality, aesthetics, wildlife habitat, and ecology. In addressing such diverse goals, NED distinguishes itself from the many decision support systems used in the forestry domain (the latter, it should be said, often deal only with maximizing timber production). By integrating external simulation and visualization packages, NED-2 allows the user to plan treatment schedules, predict their outcome, and assess their worth.

NED-2 is a blackboard based system with agents implemented in the semi-declarative programming language PROLOG. Most of the information used by NED, however, is stored in relational databases. Both relational databases and PROLOG clauses constitute NED's blackboard, and it is through the use of this blackboard that the PROLOG agents communicate.

As it integrates such diverse components, NED-2 is an Intelligent Information System (IIS), which may be viewed as a unified knowledge base, database, and model base [7]. The main idea behind this notion is the transparent processing of user queries. The system is responsible for "deciding" which information sources are required to fulfill a query regardless of whether this involves data retrieval, an inference, a computational method, a problem solving module, or some combination of these.

As a set of relational databases constitutes NED's primary storage medium and as NED's goal analysis modules are implemented in PROLOG, the nature of the interface between PROLOG and the databases is of paramount importance. An inefficient interface yields an unusable system. During the development of NED-2, it was found that the usual method of querying a database from PROLOG—a method exemplified by the ProData library of LPA, SICStus, and Quintus PROLOG [4]—was woefully inadequate. Combining data from multiple tables using PROLOG's depth-first search strategy proved too slow to be of any use. Furthermore, the usual method requires absolute knowledge of database schemas and requires that they remain constant. Such omniscience and constancy is not always available.

To overcome these deficiencies, a query language was created especially for NED that provides an efficient and friendly way to access databases from PROLOG. The user is not required to have full knowledge of the database schema, nor is the schema required to be static, nor is the user limited to a depth-first search of database relations. In place of the latter, the user can utilize the native query optimizer of the database system.

2 PROLOG and Relational Databases

It will be recalled that a PROLOG program consists of facts and rules. These can be viewed as logical assertions constituting a knowledge base. Execution of a PROLOG program is tantamount to searching the knowledge base for an answer to a query.

The similarity of PROLOG facts to tuples of a relational database should be fairly obvious. A PROLOG predicate is normally interpreted to be nothing more than a relation over some domain. More interestingly, a PROLOG rule may be viewed as a *join* of multiple database relations [8, 12, 2]. The conclusion of a PROLOG rule would in the relational model be a derived relation, or *view*, over base relations [5].

There are good reasons for wishing to marry PROLOG to a relational database. A PROLOG-like language is a concise and intuitive way of specifying instructions to a database (it is, at any rate, kinder than SQL) [5, 12]. Furthermore, PROLOG systems would greatly benefit from (1) the ability to store large amounts of information in secondary memory, and (2) from the optimization techniques built into most database systems [8, 3, 11].

Almost all real world systems linking PROLOG and a relational database system simply tack on a software interface between a preexisting PROLOG implementation and a preexisting relational database system. In other words, the two systems are *loosely coupled.* This is often a marriage of convenience, as it can be quickly implemented using off the shelf components. Such interfaces allow PROLOG to query the database when needed, either via the automatic translation of PROLOG goals into SQL or else by direct embedding of SQL into the PROLOG code.

It is common in this method of integration to designate certain PROLOG predicates as database predicates. Instead of attempting to unify them with clauses in an internal PROLOG knowledge base, such predicates and their arguments are translated into SQL queries that are then directed to an external database. The tuples returned by the database are treated as fully bound PROLOG facts.

This method of access is sometimes called *relational access*, in that only one relation is involved in each query [1]. Tuples in the database would be retrieved in PROLOG's depth-first search fashion. Database predicates are re-entrant (meaning that separate calls to a database predicate do not interfere with each other—each is bound to tuples in a top-down manner) and cuttable—meaning that they do not backtrack through a PROLOG cut [9]. In other words, they look and feel exactly like any other PROLOG predicate.

With the exception of the routines needed to implement the transparent use of database predicates, relational access requires no changes to either PROLOG or the database system. PROLOG thus gains the use of secondary storage and concurrent access and otherwise escapes unscathed.

Though relational level access is easy to implement, it is profoundly inefficient. This fact cannot be overemphasized. In restricting itself to a depth-first search of tables—a *blind* search—relational access cannot utilize any of the database system's query optimization mechanisms. A given query posed to a database system in isolation might require less than a second to answer. The same query posed to a database+PROLOG system might take literally hours or days to answer. Under no circumstances would this be acceptable.

3 PROLOG/RDBMS Integration in NED

The PROLOG components of NED-2 do not make use of relational access routines. Instead, a query in PROLOG is first matched against metadata about databases registered with NED. The query is then translated into SQL and directed to the appropriate database system. An important benefit of this technique is that one need not know a schema in its entirety in order to query a database successfully, nor is the schema required to be static. Metadata about each database can be gathered dynamically, and the metadata is used to fill in any blanks left by the user (Significantly, NED uses the metadata to automatically generate join constraints). In translating the query into SQL, one is not limited to querying a database one relation at a time. The query optimizer of the database system can be utilized.

3.1 An Example Query

A simple query in NED-2 is

```
  ?- known(‘STAND_AREA‘(
[‘STAND_ID‘ = ’patch-cut’], Value)).
```

This might be interpreted in English as "What is the area of the stand called 'patch-cut'."[1] The process of answering this query proceeds in the following manner:

1. A list of Attribute-Value pairs is formed from the original query:
   `[‘STAND_AREA‘ = Value,`
   `‘STAND_id‘ = ’patch-cut’].`[2]

2. Each term in the list is then examined for references to database attributes. If any are found, PROLOG attempts to determine, by exploring stored or dynamically created metadata, the database and table associated with each attribute.

   If attributes are associated with multiple tables, PROLOG will present multiple solutions to the query upon backtracking.

3. Attributes paired with unbound variables are set apart from the rest of the list; these will be used in the `SELECT` part of the SQL query.

4. A list of the tables associated with the attributes is kept and is used in the `FROM` part of the SQL query.

5. The remaining elements of the list—which constitute constraints on the attributes to be selected—will be used in the formation of the `WHERE` part of the SQL statement. Particularly, metadata about the relationships between tables in each database is used to create join constraints. Were it not for these, any query involving multiple relations would return attributes from the entire Cartesian product of these relations. This is obviously an unacceptable state of affairs.

[1]For present purposes, a stand is just a piece of land containing trees.

[2]Attributes are syntactically distinguished from values and variables by the use of backward quotation marks. Values are indicated with forward quotation marks. Variables, as is usual in PROLOG, are unquoted and begin with an uppercase letter.

At this point, the attributes to be selected, the list of tables, and the list of constraints are fed to a definite clause grammar which translates them into SQL. In the example, the resulting SQL query is:

```
SELECT
      ‘STAND_HEADER‘.‘STAND_AREA‘
FROM
      ‘STAND_HEADER‘
WHERE
      ‘STAND_HEADER‘.‘STAND_ID‘ =
      'patch-cut'
```

Both attributes were found to be in the ‘Stand_header’ table of a particular database. The query is directed to this database. If the query succeeds, a single value corresponding to the area of the patch-cut stand is returned. Other solutions, if they exist, would be returned upon backtracking.

4 Related Work

The translation of PROLOG expressions into SQL is absolutely vital if information is to be retrieved in a timely fashion. Indeed, in informal tests comparing databases tables containing 10,000 tuples to PROLOG knowledge bases containing 10,000 facts stored in RAM, we found that a RDBMS can outperform PROLOG even though the RDBMS is handicapped by retrieving its information from a hard disk. While it might take the RDBMS a fraction of a second to evaluate a fairly complex query, it might take PROLOG several minutes or even longer to produce the same results via its normal fetch, check, and backtrack search mechanism.

NED’s querying technique is quite similar to a language called TREQL (Thorton Research Easy Query Language) developed some time ago [5]. The intention behind that language parallels the development of NED’s language in at least one respect—namely, TREQL permits meaningful queries to be posed to databases despite some ignorance of the underlying database schema. As in NED, the poser of the query need not specify join constraints. TREQL provides these automatically. TREQL, however, is translated directly into PROLOG predicates attached in ProData fashion to database relations. This, as has been said, is an unacceptably inefficient means of querying a database.

Draxler, in [1], describes a PROLOG to SQL translator. Queries can be any complex PROLOG query involving: predicates linked to database relations; the PROLOG equivalents of `AND`, `OR`, and `NOT`; the existential quantifier `'^'`; arithmetical comparators; and aggregate functions (with the exception of the existential quantifier, the language designed for NED-2 allows all of these as well). Unlike relational level access, the system described in [1] is intended to facilitate passing complex queries to a database system, allowing the database system to do the sort of work it was designed to do (and to which PROLOG is ill suited). The routines described in [1] are used in both Ciao and XSB implementations of PROLOG. The use of Draxler’s technique is telling. Relational access is not a viable solution.

The query language described in [1] is more expressive than the language described here. However, since database relations there are specified explicitly by PROLOG predicates, a knowledge of the database schema is necessary. Furthermore, for databases containing tables with large numbers of attributes, writing them as PROLOG predicates is tedious and makes uneconomical use of space. Referring to attributes by name is far easier. It is for these two reasons that a technique more closely resembling that proposed in [1] was not used in NED-2.

5 Conclusion

The technique described above is an essential component of NED-2. What was needed was a means of retrieving information from a database both quickly and without requiring the programmer to possess complete knowledge of the database’s schema. Furthermore, what was required was that these queries be succinctly posed from within PROLOG. Though it certainly could be expanded and improved upon, the language described accomplishes this. Though it does not allow transparency (in that database relations are not treated as PROLOG predicates), this is not considered a horrible loss. Since the databases of NED-2 involve many attributes and underwent frequent changes in their developmental phases, maintaining transparency would have only caused further and unnecessary delays in development. Every change to a database schema would require a corresponding change in the PROLOG routines designed to access that database.

References

[1] Draxler, Christophe. 1993. "A powerful PROLOG to SQL compiler." Technical report, CIS Centre for Information and Speech Processing, Ludwig-Maximilians-University, Munich. http://www-2.cs.cmu.edu/afs/cs/project/ai-repository /ai/lang/prolog/code/io/pl2sql/pl2sql.tgz (accessed April 2, 2002).

[2] Gray, P. M. D., and R. J. Lucas, eds. 1988. *PROLOG and Databases: Implementations and New Directions*. Chichister, West Sussex: Ellis Horwood Limited.

[3] Irving, T. 1988."A generalized interface between PROLOG and relational databases." In *PROLOG and Databases: Implementations and New Directions*, edited by P. M. D. Gray and R. Lucas. Chichister, West Sussex: Ellis Horwood Limited.

[4] Lucas, Robert, and Keylink Computers, Ltd. 1997 *ProData Interface Manual*. Kenilworth, UK: Keylink Computers Ltd.

[5] Lunn, K; and I. G. 1988. "TREQL (Thorton Research Easy Query Language: An intelligent front-end to a relational database." In *PROLOG and Databases: Implementations and New Directions*, edited by P.M.D. Gray and R. Lucas. Chichister, West Sussex: Ellis Horwood Limited.

[6] Nute, Donald, Geneho Kim, Walter D. Potter, Mark J. Twery, H. Michael Rauscher, Scott Thomasma, Deborah Bennett, and Peter Kollasch. 1999. "A multi-criterial decision support system for forest management." In *Environmental Decision Support Systems and Artificial Intelligence*, AAAI-99, Technical Report WS-99-07, Menlo Park CA: AAAI Press. 74-81.

[7] W. Potter, D. Nute, F. Maier, M. Twery, M. Rauscher, P. Knopp, S. Thomasma, M. Dass, and H. Uchiyama. 2002. "The NED IIS Project—Forest Ecosystem Management," to appear in *Proceedings of the IFIP World Computer Congress WCC2002—Intelligent Information Processing (IIP-2002)*, August 25-30, 2002, Montreal, Canada.

[8] Sciore, Edward, and David S. Warren. 1986. "Toward an integrated database-PROLOG system." In *Expert Database Systems, Proceedings From the First International Workshop*, October 24-27, 1984, Kiawah Island, SC. Edited by Larry Kerschberg. Menlo Park, CA: Benjamin/Cummings Publishing Company, Inc.

[9] Singleton, Paul, and Pearl Brereton. 1993. "Storage and retrieval of first-order terms using a relational database." In *Advances in Databases, Proceedings of the 11th British National Conference on Databases*, July 7-9, 1993, Keele, U.K. Edited by Michael F. Worboys and A. F. Grundy.

[10] Twery, Mark J., H. M. Rauscher, D. J. Bennett, S. Thomasma, S. Stout, J. Palmer, R. Hoffman, D. DeCalesta, E. Gustafson, H. Cleveland, J. M. Grove, D. Nute, G. Kim, and R. P. Kollasch. 2000. "NED-1: Integrated analysis for for-

est stewardship decisions." *Computers and Electronics in Agriculture*. 27:167-193.

[11] Venken, R., and A. Mulkers. 1988. "The interaction from PROLOG to a binary realtional database." In *PROLOG and Databases: Implementations and New Directions*, edited by P.M.D. Gray and R. Lucas. Chichister, West Sussex: Ellis Horwood Limited.

[12] Zaniolo, Carlo. 1986. "PROLOG: A database query language for all seasons." In *Expert Database Systems, Proceedings From the First International Workshop*, October 24-27, 1984, Kiawah Island, SC. Edited by Larry Kerschberg. Menlo Park, CA: Benjamin/Cummings Publishing Company, Inc.