RULE-BASED COMPUTATIONAL DBMSs

Ronald C. Linton
Emporia State University
Emporia, KS 66801

Abstract

Tight coupling of business information systems and the streamlining of business organizational structures are trends that demand the availability of real-time corporate information for decision making by Chief Executive Officers. In this paper, the typical Executive Information System model is shown to fail to support real-time reporting because of the batch methods commonly used to extract key business transactions. In addition, a particular database management system (DBMS) architecture is proposed that allows the database administrator to establish rules which force dynamic DBMS update of key real-time corporate values as business transactions are added to the database. Such real-time updates reduce executive information presentation to simple queries of the database.

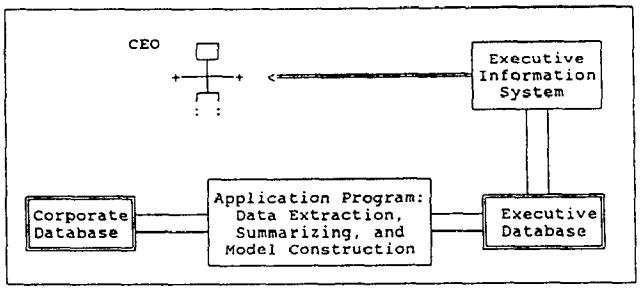
1. Introduction

If Tom Peters is correct in his prediction [PETE88] that the hierarchical structure of the typical business organization will evolve into a flat network, then the responsibilities of the successful Chief Executive Officer (CEO) will require a more direct interaction with corporate information. While interest in executive use of computers generated research activities as early as the 1960s (e.g. [BRAD67]), only recently have studies focused, in particular, on the information needs of senior management. The formal study of Executive Information Systems (EISs) was initiated by Rockart and Treacy [ROCK82] in 1982, and today there are a number of commercial EIS products available, with the high end of sophistication typified by a mix of artificial intelligence technology and superb graphics capabilities. The value of an EIS lies in its graphical presentations of both the current and projected status of factors identified as critical to corporate success. A CEO would utilize such a system in much the same manner that an operational manager would use visual displays to monitor and control some physical process. Both environments present opportunities for feedback and feedforward adjustments to control parameters. Until recently, the major difference in control in these two areas has been in the

timeliness of the response to system changes. While physical processes frequently demand instantaneous responses, those in the business environment have been slow moving, and managers have felt comfortable reviewing system status on an end-of-day, end-of-week, or in some cases, an end-of-month basis. However, if Peters is even close to the mark in his predictions, advancing technology will so tightly couple business enterprises and competition will be so fierce that (almost) real-time responses to system changes will be demanded from the outside.

Although the strength of the EIS technology is the graphical presentation of current corporate status, its weakness lies in its information generation techniques: critical transactions are extracted from the corporate mainframe database, appropriately massaged and analyzed, and then the resulting information is loaded into an executive database, frequently residing on a micro, for later CEO inquiry (see Figure 1). In the fast-paced

Figure 1
Model for a Typical EIS



business environment painted by Peters, such information gathering would be performed frequently, and would generate significant cpu cycle requirements. This processing performance could be improved by the application of standard numerical analysis methodology; in addition, the effects of data modification on statistical forecasts is a current area of research [LINT90b]. These applications would significantly reduce cpu cycle requirements by diminishing the need for complete reconstruction of summaries, model parameters, and forecasts to reflect new or modified business transactions.

Even with these quick update methods, an efficient method for recognizing and collecting important transactions remains a problem area in streamlining the EIS environment. As it now stands, depending on which factors are deemed critical to corporate success, programs would be developed to first collect copies of transactions either before or after they are added to the corporate database, and then to perform those quick updates to the Executive Database. In addition, it must be decided whether such programs would be frequently run in batch mode with appropriate database tables accessed as input files, or whether each and every application program which updates those key database tables would call such a program to perform the appropriate processing. Both approaches are unsatisfactory; the first because of performance issues, and the second because of software maintenance requirements.

A Rule-based Computational Database Management System (DBMS) is defined here as a database management system having the following features:

- a Rule Maintenance Module supporting the definition and maintenance of database update rules, and

- a "transaction" driven Inference Engine [WATE86] responsible for investigation of those update rules as user transactions update the database.

This paper presents an architecture for such a DBMS and shows that this architecture offers an efficient and effective solution to the real-time maintenance problem described above.

2. A Structured Language for Update Specification

If, say, the real-time maintenance of the mean value of all sales transactions were important for business system control, then the type of DBMS described above could provide dynamic update of a field MEAN for each insertion, deletion, or update of a record in the application database table SALES. Such a database update rule would take the general form

```
IF condition
THEN action.
```

Because the DBMS action to be taken is easily expressed in terms of the SQL UPDATE statement ([CHAM74], [DATE86]), the following command structure [LINT90c] completely expresses dynamic update requirements.

```
IF TABLE = table1 AND
  FUNCTION = function [AND
  ATTRIBUTE = attribute1]
THEN UPDATE table2
  SET attribute = expression [,
     attribute = expression . . .]
  [WHERE predicate],
```

where table1 denotes the database table which is the object of an application program update, function denotes the type of application I/O request that should fire the rule (effectively, one of INSERT, DELETE, or UPDATE), and attribute1 denotes the particular field in an UPDATE transaction that would cause the rule to fire. For example, the rule

```
IF TABLE = SALES AND
  FUNCTION = INSERT
THEN  UPDATE SYSTEMPOOL
  SET MEAN =
  {(COUNT-1)*MEAN+SALES.AMT}/COUNT
```

requires the DBMS to update the data field MEAN in the table SYSTEMPOOL whenever a user application inserts a new record into the SALES table. The formulation of this rule presupposes the ordered application of other rules, particularly the one in which the value of COUNT would be incremented because of the insertion. This sequencing must be controlled by the Inference Engine which evaluates rules as transactions are added.

3. System Generation

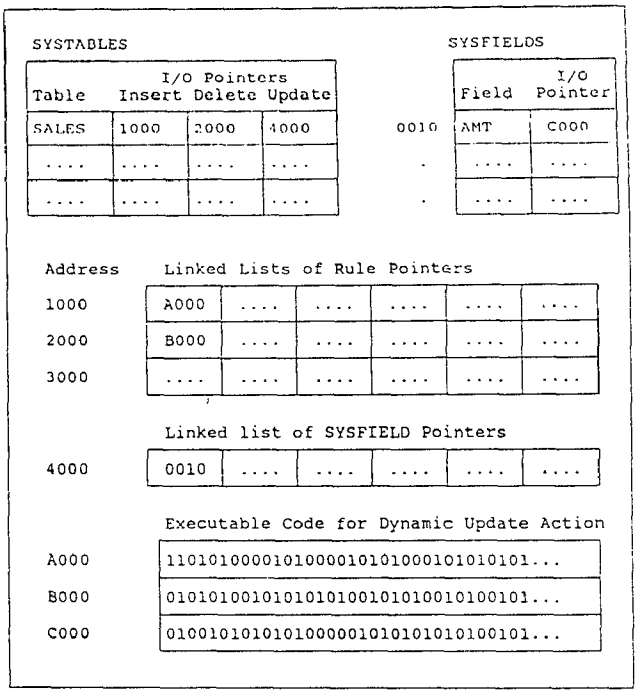
Prior to generating the DBMS environment, the Database Administrator (DBA) utilizes the Rule

Maintenance Module to build a rule base reflecting all dynamic update requirements. The system generation process consists of three phases. During the first phase, typical data dictionary entries are used to construct standard system tables, including SYSTABLES, a table of table names, and SYSFIELDS, a table containing all field names in the application database. In the second phase, each rule is examined and certain constructions are made:

- The action to be taken by the DBMS has been expressed in terms of a SQL UPDATE command. During this second phase, this command is converted to the same executable code that would be produced by a SQL interpreter.

- The condition under which the rule should fire determines an application database table name, an I/O type, and perhaps a data field name (only for UPDATE transactions). In SYSTABLE there is generated, for each application table, three linked lists - one for each type of I/O (see Figure 2). For the I/O types INSERT and DELETE, the only entry made in the linked list is a pointer to the instructional equivalent of the required UPDATE action. For the I/O type UPDATE, the only entry in the linked list is a pointer to a linked list of entries in SYSFIELDS representing the names of the particular data fields whose modification would trigger the action. In addition, for each data field name in SYSFIELDS, there is generated a linked list of pointers:

Figure 2
I/O Related Linked Lists



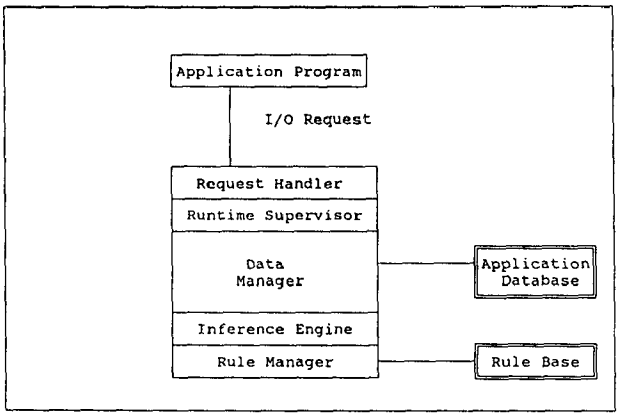
a pointer to an UPDATE (action) instruction set is added to the list that corresponds to the particular data field triggering the action.

- During the third phase, the final load module version of the DBMS is produced along with appropriate secondary storage files.

4. System Architecture

In general, the system consists of five components, namely the I/O Request Handler, Runtime Supervisor, Data Manager, Inference Engine, and Rule Manager (see Figure 3).

Figure 3
System Architecture



The functions of these components are:

- The I/O Request Handler evaluates the parameters passed by the application program in its call to the DBMS. As it visits SYSTABLES and SYSFIELDS when validating table and field names, the Request Handler initializes, for the Inference Engine, a (perhaps null) pointer to the head of the appropriate I/O related linked list found on SYSTABLES.

- After successful editing of the request, control is passed to the Runtime Supervisor which in turn invokes the Data Manager. Upon completion of the requested I/O, the Supervisor invokes the Inference Engine if the pointer to the head of the I/O linked list is not null.

- The Data Manager is responsible for managing the physical database and invokes lower-levelcomponents to perform functions such as buffering, locking, and journaling.

- The Inference Engine is responsible for the ordered execution of the dynamic update machine code. After the address of next set of update instructions is determined, an appropriate pointer in the I/O Request Handler is modified, the pointer to the I/O Linked List is incremented, and control is passed to the Request Handler so that it may handle this dynamic request as if it were an application initiated request. The role of the Rule Maintenance Manager is to provide, prior to system generation, on-line entry and editing of DBA constructed update rules. In addition, this Manager provides integrity checks to avoid execution time anomalies such cycling.

5. The Inference Engine

In the case where a dynamic update of the database itself triggers one or more other such updates, the Inference Engine maintains a stack of current pointers to I/O Linked Lists. Whether a stack is constructed, or not, is determined by the initial value of the lowest level pointer at the time that the Request Handler updates its value during the editing of request table and field names. If the pointer is not null, the Handler assumes a nesting of updates and initializes the pointer at the next level up. At the end of the handling of the stack, the Inference Engine sets the lowest level to null.

6. EISs Revisited

With the technology of a Rule-based Computational DBMS available, all real-time computations required by an EIS can be guaranteed, and the graphics software need only query the database to get summarized values, projected values, and any other model parameters. Current research [LINT90d] is focusing on opportunities presented by this technology to develop EIS software in which the CEO can choose to either continuously monitor critical factors or to allow the system, using appropriate expertise, to monitor these factors itself and then to notify the executive when an anticipated problem is detected. It is reasonable to expect future EIS products to recommend alternative responses and to graphically demonstrate consequences of potential CEO decisions.

7. Consideration of an Alternative

It may seem that, rather than developing a new DBMS architecture, it would be far simpler to enhance application development language or SQL preprocessors so that appropriate dynamic update commands would be inserted as a reflection of the DBA constructed rules. This approach would move all rule interpretation and code generation from sysgen time to compile time, but it would also create some additional processing requirements as each inserted update would require a call to and a return from the DBMS. In addition, a complication occurs in the case that the environment requires the adjustment of some data field whenever some other particular database field is modified. For example, suppose that stored in the table PROD is a collection of records containing product information: NUMB, DESC, and QOH. Suppose further that the table INTX is used to store inventory transactions and that each record contains the fields TNUMB, PNUMB and QTY. Now, if for each product number NUMB, QOH is a summary of all associated values of QTY, then we could charge the DBMS with the responsibility for updating these QOH values. It would be difficult to develop an SQL preprocessor algorithm that would convert the following update requirement to a standard SQL command: "If, for some product, the value of QTY is modified, then the associated value of QOH should be appropriately modified". The difficulty lies in not knowing in advance the name used by the application program to hold the new value of QTY.

In the case of the Update Specification Language described above, this requirement can be expressed as follows (see [LINT90c] for more details) :

```
IF TABLE = INTX AND
  FUNCTION = UPDATE AND
  ATTRIBUTE = QTY
THEN UPDATE PROD
  SET PROD.QOH =
   PROD.QOH - INTX.QTY..O + INTX.QTY..N
  WHERE PROD.NUM = INTX.PNUM,
```

where INTX.QTY..O, and INTX.QTY..N denote, respectively, the old and new values of QTY. The advantage of the new architecture introduced here is that the DBMS can be instructed to maintain the "before" and

"after" images of the modified data record, whose constructions are required for journaling, anytime the Linked List pointer, as initialized by the Request Handler, is not null.

8. Other Applications

The inventory control example above demonstrates application potential beyond the real-time maintenance of computational data items. In particular, it suggests that the DBMS could be charged with any updating of "parent" records which reflect the insertion, deletion, or modification of corresponding "child" records. In addition, the DBMS can be required to make certain "child" updates that would maintain database consistency. For example, the rule

```
IF   TABLE = PROD AND
     FUNCTION = UPDATE AND
     ATTRIBUTE = PROD.NUM
THEN UPDATE INTX
     SET INTX.PNUM = PROD.NUM..N
     WHERE INTX.PNUM = PROD.NUM..O
```

will cause an automatic adjustment to all "child" INTX records whenever a product number is changed on the "parent" record in PROD.

9. Future Research

The author is currently involved in an implementation of both the DBMS architecture and the companion update specification language described in this paper. Beyond the study of system performance in a multiuser environment and an application of this technology in streamlining EIS applications, there is a need to identify other opportunities for applications of this technology.

10. References

[BRAD67] Brady, Rodney H., "Computer Use in Top-level Decision Making," Harvard Business Review, 45, (July - August), 1967.

[CHAM74] Chamberlin, D. D. with R.F.Boyce, "SEQUEL: A Structured English Query Language," Proceedings ACM SIGMOD Workshop on Data Description, Access, and Control, May, 1974.

[DATE86] Date, C. J., An Introduction to Database Systems, Addison-Wesley, 1986.

[LINT90a] Linton, Ronald C., "Applications of Rule-based Computational DBMSs", Proceedings Southwest Business Symposium, Edmond, OK, 1990.

[LINT90b] Linton, Ronald C., "Effects of Data Modification on Short Term Forecasts", presented at the Kansas-Western Missouri Chapter of the American Statistical Association, Kansas City, MO, 1990.

[LINT90c] Linton, Ronald C., "A Structured Language for Rule-based Computational DBMSs", Proceedings Decision Sciences Institute Annual Meeting, San Diego, CA, 1990.

[LINT90d] Linton, Ronald C. with Jacob R. Wambsganss, "Expert Executive Information Systems - Key to Small Business Success?", Proceedings Decision Sciences Institute Annual Meeting, San Diego, CA, 1990.

[PETE88] Peters, Tom, Thriving on Chaos, Alfred A. Knopf, New York, 1988.

[ROCK82] Rockart, John F. with Michael E. Treacy, "The CEO Goes On-Line," Harvard Business Review, 60, (January - February), 1982.

[WATE86] Waterman, Donald A., A Guide to Expert Systems, Addison-Wesley, 1986.