A Systematic Architecture of An Active Database Management System

Thodeti Srikanth

1Research Scholar, Department of Computer Science, Dravidian University, A.P., India

Abstract—Database Management Systems (DBMSs) are actually a ubiquitous and critical part of contemporary computing, and the outcome of years of study and development in each academia and industry. We've also created an execution model that says show the rules are actually prepared in the context of data source transactions. The extra functionality provided by ECA rules creates brand new demands on the layout of an energetic DBMS. In this particular paper we suggest an architecture for an active DBMS which supports ECA rules. This architecture provides brand new types of interaction, in support of ECA rules, between application plans and also the DBMS. This results to a new paradigm for constructing database applications.

Index Terms—Contemporary computing.

I. INTRODUCTION

However, database systems are beginning to be applied to a range of domains associated with highly complex information processing, evermore substantial quantities of data, or highly stringent performance requirements, in which the conventional multicomponent environment has proved to be unsatisfactory. This has resulted in a trend in database research towards more of the functionality required by an application being supported within the database system itself, giving rise to database systems with more comprehensive facilities for modeling both the structural and the behavioral aspects of an application. Among the fields that have received attention in recent years with a view to enhancing the behavioral facilities of database systems are database programming, temporal databases, spatial databases, multimedia databases, deductive databases, and active databases. This survey focuses upon the last mentioned. Traditional database management systems (DBMSs) are passive in the sense that commands are executed by the database (e.g., query, update, delete) as and when requested by the user or application program. However, some situations cannot be effectively modeled by this pattern. As an example, consider a railway database where data are stored about trains, timetables, seats, fares, and so on, which is accessed by different terminals. In some circumstances (e.g., public holidays, cultural events) it may be beneficial to add additional coaches to specific trains if the number of spare seats a month in advance is below a threshold value. Two options are available to the administrator of a passive database system who is seeking to support this requirement. One is to add the additional monitoring functionality to the booking programs so that the preceding situation is checked each time a seat is sold. However, this approach leads to the semantics of the monitoring task being distributed, replicated, and hidden among different application programs. The second approach relies on a polling mechanism that periodically checks the number of seats available. Unlike the first approach, here the semantics of the application is represented in a single place, but the difficulty stems from ascertaining the most appropriate polling frequency. If too high, there is a cost penalty. If too low, the reaction may be too late (e.g., the coach is added, but only after several customers have been turned away).

Active databases support the preceding application by moving the reactive behavior from the application (or polling mechanism) into the DBMS. Active databases are thus able to monitor and react to specific circumstances of relevance to an application. The reactive semantics is both centralized and handled in a timely manner. An active database system must provide a knowledge model (i.e., a description mechanism) and an execution model (i.e., a runtime strategy) for supporting its reactive behavior.

A common approach for the knowledge model uses rules that have up to three components: an event, a condition, and an action. The event part of a rule describes a happening to which the rule may be
able to respond. The condition part of the rule examines the context in which the event has taken place. The action describes the task to be carried out by the rule if the relevant event has taken place and the condition has evaluated to true.

Most active database systems support rules with all three of the components described; such a rule is known as an event-condition-action or ECA-rule. In some proposals the event or the condition may be either missing or implicit. If no event is given, then the resulting rule is a condition-action rule, or production rule. If no condition is given, then the resulting rule is an event-action rule.

II. RELATED WORKS

Probably the most mature and commonly used database devices in production these days are actually relational database management methods (RDBMSs). These methods may be discovered with the center of a great deal of the world's application infrastructure including e-commerce, health records, billing, human resources, payroll, customer connection management and supply chain management, to name just a few. The arrival of web-based commerce and community-oriented sites has just improved the volume as well as breadth of the use of theirs. Relational systems work as the repositories of record behind almost all internet transactions and most internet content management methods (blogs, social networks, wikis, as well as the like). Along with being critical application infrastructure, relational data source methods work as a well understood point of reference for brand new extensions as well as revolutions in database devices that could develop in the future. Given these different choices, a regular DBMS needs to be compatible with a number of different connectivity protocols employed by a variety of customer owners as well as middleware methods.

At base, nonetheless, the duty of the DBMS' client communications supervisor in all the protocols is roughly the same: to set up and remember the hookup state for the caller (be it a client or maybe a middleware server), to respond in order to SQL commands from the caller, as well as to return both information as well as balance emails (result codes, mistakes, etc.) as adequate. In the simple example of ours, the communications manager would set the security qualifications of the prospect, set upstate to recall the details of the brand new interconnection and the today's SQL command across calls, along with advanced the client's very first request deeper into the DBMS to be dealt with.

Upon getting the client's very first SQL command, the DBMS must designate a "thread of computation" to the command. It should also make certain that the thread's information and control outputs are actually connected via the communications director to the client. As soon as admitted and allocated like a thread of command, the gate agent's query could start executing. It does so by invoking the code in the Relational Query Processor.

At the base of the gate agent's query program, one or maybe more operators can be found to request details from the database. These Calls are made by operators to fetch information from the DBMS. At this stage in the example of this query's life, it's started to access information records, and is actually prepared to make use of them to calculate outcomes for the customer. This is accomplished by "unwinding the stack" of things we described up to this stage. The access methods return management to the query executor's operators, which orchestrate the computation of outcome tuples from database data; as outcome tuples are actually produced, they're placed in a buffer for the prospect marketing communications manager, which ships the results back again to the caller. For substantial result sets, the client generally will make extra calls to fetch a lot more details incrementally from the query, resulting in several iterations through the communications supervisor, query executor, and storage space supervisor. The query the transaction is actually completed as well as the hookup closed; this results in the transaction supervisor cleaning up express for the transaction, the procedure manager freeing any management constructions for the query, as well as the communications supervisor cleaning up communication express for the connection.

III. THE PROPOSED APPROACHES

When a rule fires, database operations are performed as part of condition evaluation and action execution. These database operations are executed concurrently with application transactions and other rule firings. The HiPAC execution model [HSUSS] describes how rules fire in database transactions, and the relationships among these transactions. The execution model
consists of a nested transaction model, and an assignment of condition evaluation and action execution to transactions base don coupling.

In a nested transaction model IMOS851 there are two types of transactions: top level transactions and nested transactions (also called subtransactions). A nested transaction is wholly contained within another transaction, called its parent. The parent of a nested transaction can be a top level transaction or another nested transaction. A transaction can have more than one subtransaction, and sibling subtransactions can execute concurrently. Our model assumes that a parent transaction is suspended while its subtransactions execute.

Top level transactions, like the usual database transactions are atomic, serializable, and permanent.

Nested transactions are atomic. Concurrently executing sibling subtransactions are serializable. The effects of a subtransaction do not become permanent until it, and all of its ancestors through top transaction, commit. When a transaction aborts, its effects and the effect of all of its descendants are discarded.

The first application implemented over HiPAC was a SecuritiesAnalyst’s Assistant (SAA). The purpose of this application is to deliver information to an analyst’s display, and to automatically execute trades according to the analyst’s instructions. This application is shown in Figure 1. It consists of programs and rules.

Display Displays prices, trades, portfolios and other information on an analyst’s workstation.
Trader Executes trades by transmitting requests to a trading service and updating the client’s portfolio when the reply is received.

There would be several copies of a program running: one ticker for each source of price quotes (e.g., NYSE), one display for each analyst using the application, and one trader for each trading service.

The rules for the SAA application are divided into two groups, display and trading, according to the application operations invoked in their actions. Display rules contain requests to a display program in their action. For example, each analyst’s display includes a window that scrolls price quotes from right to left, like the stock ticker seen at a brokerage.

This ticker window is driven by the following rule:

Event: update stock price
Condition: true
Action: send “display price quote” request to display program
Coupling: condition and action together in a separate transaction.

There is a rule of this form for each display program running. The actions for trading rules contain request to trading programs. For example, an analyst might instruct the application to buy 500 shares of Xerox for a client when the price reaches 50.

This is expressed as a rule:

Event: update Xerox price
Condition: where new price = 50
Action: send request to buy 500 shares for client
ACoupling: condition and action together in a separate transaction

The execution of a trade is an event defined by SAA and signalled by a trading program. There is a display rule that causes the trade to be displayed and the portfolio updated on the analyst’s screen and trading rules.

After implementing the SAA using HiRAC rules, we noted the following: There are no direct interactions between the application programs. All interactions take place through rules firing. The application programs tended to be quite simple servers. The control logic was encoded in the rules.

Most of our rules contained requests to application programs in their actions, rather than database operations. To modify the behavior of the application, we would change the rules rather than the software.

In traditional database applications, data

![Figure 1: An Example of an Active Database Application](image-url)
flows from one application program to another through the DBMS. With ECA rules, application defined events, and calls to application programs in rule actions, HiF’AC provides a medium for flow of control as well as data. The high level logic for the application can be encoded in rules rather than software, making the application more modular and easier to modify.

IV. CONCLUSIONS

We’ve presented a structure for HiPAC, an active DBMS with ECA guidelines. This architecture supports the knowledge model, execution version, and condition monitoring algorithms discussed in the HiPAC research papers. In doing so, it specifies 2 brand new types of interaction between an application and the DBMS. Application programs signal functions, along with HiPAC producing requests to program programs in executing principle actions. This particular leads to a brand new paradigm for building programs over an energetic DBMS. Control logic is actually encoded in rules rather than software. HiPAC becomes a moderate for the flow of influence, as well as data, between application plans.

REFERENCES


Author:

Thodeti Srikanth, Research Scholar, Department of Computer Science in Dravidian University and working as HoD, Department of Computer Science in N.S.V. Degree College, Jagityal, Telangana, India. I received Master of Computer Applications (M.C.A.) from Kakatiya University in April 2004.