www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issue 1 January 2016, Page No. 15572-15577

A Paradigm for Active Database Management Systems

Manvendra Yadav , Sonia Kumari

Atma Ram Sanatan Dharma College , University Of Delhi , India

ymanvendra@gmail.com

Shyama Prasad Mukherji College (For Women), University Of Delhi, India

soniakumari.ducs@gmail.com

ABSTRACT

Active database systems support mechanisms that enable them to respond automatically to events that are taking place either inside or outside the database
system itself. Considerable effort has been directed towards improving understanding of such systems in recent years, and many different proposals have been made and applications suggested. This high level of activity has not yielded a single agreed-upon standard approach to the integration of active functionality with conventional database systems, but has led to improved understanding of active behaviour description languages, execution models, and architectures. This paper presents the fundamental characteristics of active database systems[11], describes a collection of representative systems within a common framework, considers the consequences for implementations of certain design decisions, and discusses tools for developing active applications. Active database management systems are invoked by synchronous events generated by user or application programs as well as external asynchronous data change events such as a change in sensor value or time. In this paper gives the introduction of active DBMS[12] and discussed how it is different from passive DBMS.

Keywords
Active database management systems(ADBMS), Rule-Based Systems Application Program.

1. INTRODUCTION

Traditionally, database systems have been viewed as repositories that store the information required by an application, and that are accessed either by user programs or through interactive interfaces. In such a context, a range of different tools and systems are used together to support the requirements of the application. However, database systems are beginning to be applied to a range of domains associated with highly complex information processing, ever more substantial quantities of data, or highly stringent performance requirements, in which the conventional multi component 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[5] with more comprehensive facilities for modelling both the structural and the behavioural aspects of an application. Among the fields that have received attention in recent years with a view to enhancing the behavioural 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[5] (e.g., query, update, delete) as and when requested by the user or application program. However, some situations cannot be effectively modelled 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 all 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[11][12] by moving the reactive behaviour 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 this reactive behaviour. 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[4]. 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. At first glance, the introduction of active rules to a database system may seem like a straightforward task, but in practice proposals have been made that support widely different functionalities. Among the issues that distinguish proposals are the expressiveness of the event language, the scope of access to database states from the condition and action, and the timing of condition and action evaluation relative to the event. The functionality of a specific system will be influenced by a number of factors, including the nature of the passive data model that is being extended, and the categories of application to be supported .

2. Active DBMSs (ADBMSs) versus Passive DBMSs

DBMSs are passive in the sense that they are explicitly and synchronously invoked by user or application program initiated operations. Applications send requests for operations to be performed by the DBMS and wait for the DBMS to confirm and return any possible answers. The operations can be definitions and updates of the schema, as well as queries and updates of the data. Active Database Management Systems (ADBMSs) are event driven systems[11][12] where operations such as schema changes and changes to data generate events that can be monitored by active rules. An ADBMS can be invoked, not only by synchronous events that have been generated by users or application programs, but also by external asynchronous events such as changes of sensor values or time. When monitoring events in a passive database, a polling technique or operation filtering can be used to determine changes to data. With the polling method the application program periodically polls the database by placing a query about the monitored data. The problem with this approach is that the polling has to be fine-tuned so as not to flood the DBMS with too frequent queries that mostly return the same answers, or in the case of too infrequent polling, the application might miss important changes of data. Operation filtering is based on the fact that all change operations sent to the DBMS are filtered by an application layer that performs the situation monitoring before sending the operations to the DBMS. The problem with this approach is that it greatly limits the way rule condition evaluation can be optimized. It is desirable to be able to specify the conditions to monitor in the query language of the DBMS. By

checking the conditions outside the database the complete queries representing the conditions will have to be sent to the DBMS. Many DBMSs allow precompiled stored procedures that can update the database. The effects of calling such a procedure cannot be determined outside of the database. If the condition monitoring is used to determine inconsistencies in the database, it is questionable whether this should be performed by the applications, instead of by the DBMS itself. In an integrated ADBMS condition monitoring is integrated into the database. This makes it possible to efficiently monitor conditions and to notify applications when an event occurred that caused a rule condition to become true and that is of interest to the application. Monitoring of specific conditions, represented as database queries, can be performed more efficiently since the ADBMS has more control of how to evaluate the condition efficiently based on knowledge of what has changed in the database since the condition was last checked. It also lets the ADBMS perform consistency maintenance as an integrated part of the data management. Internal ADBMS functions that can use data monitoring includes, for example, constraint management, management of long-running transactions, and authorization control. In constraint management, rules can monitor and detect
inconsistent updates and abort any transactions that violate the constraints. In
some cases compensating actions can be performed to avoid inconsistencies instead of performing an abortion of the complete transaction. In the management of long-running transactions, rules can be used to efficiently determine synchronization points of different activities and also whether if one transaction has performed updates that have interfered with another [2]. In authorization control rules can be used to check that the user or application has permission to do specific updates or schema changes in the database. Applications which depend on data monitoring activities such as CIM, Telecommunications Network Management, Medical [6] and Financial Decision Support Systems [7] can greatly benefit from integration with ADBMSs.

3. Temporal Logic in Active Databases

In ADBs, dynamic integrity constraints formulated in temporal logic are converted to ECA rules[4]. An architecture is proposed for implementing temporal integrity constraints by compiling them into a set of active DBMS rules. This compiler allows automatic translation of integrity constraints formulated in past temporal logic into rules of an active DBMS. During compilation, the set of constraints is checked for the safe evaluation property. The result is a set of SQL statements that includes all the necessary rules needed for enforcing the original constraints. When the rules are activated, all updates to the database that violate any of the constraints are automatically rejected (meaning the corresponding transaction is aborted). This method converts past temporal logic formulae into a set of SQL statements. For past temporal logic formulas, the truth of the formula in state n depends only on the finite history D_0 ,D_1 ,D_2..... Dn of the database. [9] uses past temporal logic for specifying conditions and events in the rules for active database system. An algorithm is presented for incremental evaluation of temporal conditions and a new framework for processing temporal constraints. This method stores the history of the database. In [10], monitoring schemes for dynamic integrity constraints are developed. Generating triggers for monitoring integrity from dynamic constraint formulae are addressed. Refering to the “job” description of a database from the first section, we see that all of these approaches go beyond what is meant to be supported by a database. In fact, they address precisely some of the issues that are required for modeling the interactive behavior of information systems that are defined by their service descriptions.

4 . Using Rules as a Complement to Traditional Coding

Active rules can serve as a complement to traditional coding techniques where all the functionality of the system is specified in algorithms written in modules and functions. Active rules provide a more dynamic way of handling new situations and are often better alternatives to modifying old functions to cope with new situations. Great care has to be taken, however, when using active rules to avoid introducing unanticipated behaviour into the system. Misuse of rules, such as using too many levels of rules that can affect each other in

unpredictable ways or attempts to use rules where traditional functions are more suited is one reason why the use of active rules in software development has only had limited success. A common technique that is used is to use rules for specifying parts of the system during the design phases and to use these rules as guidelines Introduction for the actual coding phases or to compile the rules into corresponding functions to simplify the coding. This last technique is sometimes found directly supported in some programming languages where pre- and post-conditions on data can be specified. If the conditions are violated an error is generated. Rules can, however, specify pre- or post-conditions that should apply in many different situations not just in one piece of code. The rules can signal to the user or some application that a condition has been violated. Rules can also specify actions to be taken, such as removing inconsistencies by changing illegal values of data. In most programming languages and query languages such as SQL fault or signal handlers can be defined that catches error signals. Rules in an ADBMS can be seen as having similar behaviour, but catches database events such as updates. Rules can also be used for monitoring changes to data. These are often specified as conditional expressions (if-then-else, or case expressions) in traditional coding. the rules can be dynamically changed. New situations can also be monitored by adding new rules.

5. Rule-Based Systems and Active Database Systems

In rule-based systems the rules can be used for different purposes. In fig. 1 the distinction is made between using rules for monitoring, control, and reasoning. We here make a distinction between active database systems [3] and other rule based systems such as reactive systems (sometimes called real-time expert systems) and knowledge-based systems [1] (often just referred to as expert systems).

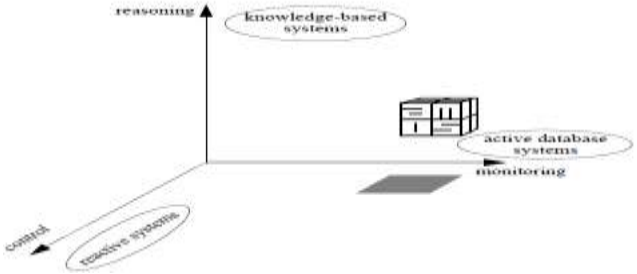

Figure 1 The relation between active database systems and other rule based

Active database systems are primarily database management systems with the main task of storing large amounts of data and providing efficient access to this data through a query language. In active database systems the rules are primarily used for monitoring changes to the data stored in the database. In reactive systems the rules are used for reacting to changes of some external environment and performing actions on (controlling) the environment in response to the changes. In knowledge-based systems the rules are usually used for reasoning using stored facts and by deducing new facts by using the rules. As can be seen in fig. 1 there is no sharp distinction between the three different kinds of rule systems. An active database system can do limited reasoning by using rules with more complicated rule conditions and which store new data in the database as new facts that signify that the rules have triggered. Control of the environment represented by the data in the database itself can also be performed, e.g. with constraint rules that modify the database to remove any inconsistencies. By allowing the active database manager to access an external environment that can be both accessed and updated, the rules in the active database can be used for control of an external environment as well. The primary use of active rules in an active database system as presented in this thesis is to monitor changes to the data that can be accessed in the database.

6. EXECUTION MODEL

The execution model specifies how a set of rules is treated at runtime, the execution model of a rule system is closely related to aspects of the underlying DBMS (e.g., data model, transaction manager), there are a number of phases in rule evaluation, illustrated in Figure 2, that transcend considerations that relate to specific software environments.

- The signaling phase refers to the appearance of an event occurrence caused by an event source.
- The triggering phase takes the events produced thus far, and triggers the corresponding rules. The association of a rule with its event occurrence forms a rule instantiation.
- The evaluation phase evaluates the condition of the triggered rules. The rule

conflict set is formed from all rule instantiations whose conditions are satisfied.

- The scheduling phase indicates how the rule conflict set is processed.
- The execution phase carries out the actions of the chosen rule instantiations.

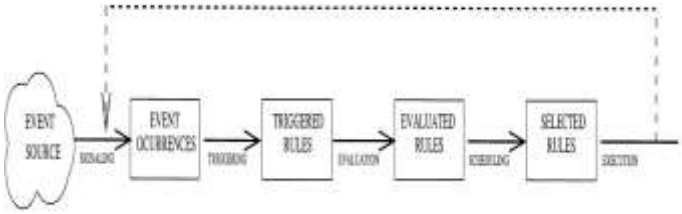

Figure 2: Principal steps that take place during rule execution

During action execution other events can in turn be signalled that may produce cascaded rule firing. These phases are not necessarily executed contiguously, but depend on the Event-condition and Condition-action coupling modes. The former determines when the condition is evaluated relative to the event that triggers the rule. The Condition-action coupling mode indicates when the action is to be executed relative to the evaluation of the condition. A further aspect to be considered is how Error handling is supported during rule firing. Most systems simply abort the transaction, as this is standard behaviour in databases. However, other alternatives may be more convenient that is to ignore the rule that raised the error and to continue processing other rules to backtrack to the state when rule processing started and either restart rule processing or continue with the transaction to adopt some contingency plan that endeavours to recover from the error state, possibly using the exception mechanism of the underlying database system.

7. ADBMS in Large Complex Systems

The introduction of a DBMS into a system provides a good platform for designing rules that access data from different parts of the system. Rules are dependent on the fact that all the information that is needed to check the rules is available. In a system without a general mechanism for storing data the rules have to be compiled into each module or function that can affect the rule condition. This limits the rule to just relating to data available in that module or function. In an ADBMS active rules are managed by the ADBMS and the rules can thus directly access data stored in the database. Rules specified in a database can have conditions that span over data belonging to several modules of the system. The active rules can be used for directly supporting various applications with monitoring of changes to data in the database, with synchronizing activities in the system, and with maintaining the integrity of data in the database. Care has to be taken when designing these systems to not introduce unwanted or unspecified communication between modules through the database. A common and successful technique in designing large systems has been to carefully design the interaction between different modules or processes by special interfaces, i.e. by exported interface functions or by inter-process communication. The design phase [8] now has to take into consideration what data that is going to be stored in the database for each module and what data is going to be visible to other modules. By storing information about the state of the system in the database, e.g. the state of different hardware and software components, active rules can be used to monitor the state of the system itself. If the system is interacting with some external environment, e.g. a telecommunication network or a manufacturing plant, state information of these environments can be made available in the database as well. This could be done by mapping sensor data into the database and making it available in queries and rules. This does not have to mean that the sensor data is always stored permanently in the database. It may be the case that the sensor data is available to read as if it was stored directly in the database and that the ADBMS is informed when the sensor data changes. In many cases it makes no sense to store the data permanently since it changes quite frequently. The sensor data can sometimes be stored for logging purposes, but this might already be done in some other system that is part of the external environment. Allowing access to the state of the external environment through the database makes it possible to use active rules to monitor changes in the external Environment.

8. Conclusions

Active databases have been traditionally considered as data transformation systems, with research methods that borrow from traditional database arsenal. This is despite the fact that one of the design issues in active databases is in bringing application-level integrity concerns to the database level. As a result, designing the rule system of ADBMS is a challenge and the mapping between application requirements
and the system of rules remains complicated. We argue for a change in viewpoint, so active databases are embraced as a special (restricted) type of an information system rather than a special (augmented) type of a database. As such, they are interactive service-providing systems rather than mere data transformation engines. This change in perspective offers a promising approach for addressing ADBMS shortcomings, and reveals a roadmap of additional features for ADBMS.

REFERENCES :

1. Hedberg S. and Steizner M. "Knowledge Engineering Environment (KEE) System" Summary of Release 3.1, Intellicorp Inc. July 1987.

2. Dayal U., Hsu M., and Ladin R. "Organizing Long-Running Activities with Triggers and Transactions", Proceedings of the ACM SIGMOD International Conference on Management of Data, Atlantic City, May 1990.

3. Morgenstern M. "Active Databases as a Paradigm for Enhanced Computing Environments" , Proceedings of the 9th VLDB Conference, Florence, Nov. 1983.

4. N. W. Paton (Ed.), "Active Rules in Database Systems". Springer, New York 1999, pp. 81-102.

5. ABITEBOUL, S. AND HULL, R. 1987. "IFO: A formal semantic database model", ACM Trans. Database Syst. 12, 4 (Dec.), 525–565.

6. Hayes-Roth D., Washington R., Hewett R., Hewett M., and Seiver A. "Intelligent Monitoring and Control", Proceedings of the 1989 International Joint Conference on Artificial Intelligence, 1989.

7. Chandra R. and Segev A , "Active Databases for Financial Applications", RIDE ´94, Houston, Febr., 1994, Pages 46-52.

8. Widom J. and Ceri S. (ed.), "Active Database Systems - Triggers and Rules for Advanced Database Processing", Morgan Kaufmann Publishers, Inc., ISBN-1-55860-304-2, 1996.

9. A.P. Sistla, O. Wolfson, "Temporal Conditions and Integrity Constraints in Active Database Systems" , Proc. 1995 ACM SIGMOD Int'l Conf. on Management of Data, pp. 269– 280, 1995.

.

10. M.Gertz, U.W.Lipeck. "Deriving optimized monitoring triggers from dynamic integrity constraints". IEEE Proc. Of Data and Knowledge Eng., Vol. 20(2), pp. 163-193, 1996.

11. F. Casati, S. Castano, and M. Fugini, "Managing Workflow Authorization Constraints through Active Database Technology" . Information Systems Frontiers 3:3, Sep. 2001.

12. N. W.Paton, Os. Diaz. " Active Database Systems", ACM Computing Surveys, Vol 31, No 1, March 1999.