The Architecture Of An Active Data Base Management System*

Dennis R. McCarthy
Xerox Advanced Information Technology
4 Cambridge Center
Cambridge, MA 02142
ARPANET: mccarthy@xait.xerox.com

Umeshwar Dayal
Digital Equipment Corporation Cambridge Research Laboratory
One Kendall Square
Cambridge, MA 02139
ARPANET: dayal@crl.dec.com

Abstract

The HiPAC project is investigating active, time-constrained database managment. An active DBMS is one which automatically executes specified actions when speciifed conditions arise. HiPAC has proposed Event-Condition-Action (ECA) rules as a formalism for active database capabilities. We have also developed an execution model that speicifes how these rules are processed in the context of database transactions. The additional functionality provided by ECA rules makes new demands on the design of an active DBMS. In this paper we propose an architecture for an active DBMS that supports ECA rules. This architecture provides new forms of interaction, in support of ECA rules, between application programs and thDBMS.This leads to a new paradigm for constructing database applications.

1. INTRODUCTION

Conventional database management systems are passive, in the sense that they only manipulate data in response to explicit requests from applications. The HiPAC project is investigating active, time-constrained database management [DAY88a]. An active DBMS is a DBMS that allows users to specify actions to be taken automatically, without user intervention, when certain conditions arise. Active DBMS capabilities can be traced back to the ON conditions of CODASYL [COD73]. Triggers and assertions were proposed for System R as a mechanism for enforcing integrity constraints [ESW75,ESW76]. Declarative rules for expressing relationships between data items [MOR83,STO86] are another form of active DBMS capability. Simple triggers are now appearing in commercial DBMS products [SYB87], and automatic enforcement of referential integrity constraints is included in the ANSI SQL2 standard.

*This work was supported by the Defense Advanced Research Projects Agency and by the Rome Air Development Center under Contract No. F30602-87-C-0029. The views and conclusions contained in this report are those of the authors and do not necessarily represent the official policies of the Defense Advanced Research Projects Agency, the Rome Air Development Center, or the U.S. Government.

The HiPAC project has proposed Event-Condition-Action (ECA) rules as a general formalism that subsumes most of these active DBMS functions, which previously were implemented by special purpose mechanisms. We have developed a knowledge model [DAY88b] that describes the semantics of ECA rules, and an execution model [HSU88] that specifies how these rules are processed in database transactions. We have also begun work on time-constrained scheduling of database transactions [BUC89].

ECA rules provide active database capabilities beyond what is found in a conventional, passive DBMS. The design of an active DBMS must provide support for implementing this additional functionality. This paper proposes an architecture for an active, object-oriented DBMS that implements our knowledge model and execution model.

The next two sections contain overviews of the HiPAC knowledge model and execution model, respectively. Section 4 then describes how applications interact with HiPAC in using ECA rules. Section 5 describes the internal structure of HiPAC to support rule processing, and Section 6 shows how the components of the HiPAC architecture interact to process a rule. Finally, Section 7 describes the implementation status of HiPAC, and our plans for future work.

2. KNOWLEDGE MODEL: RULES AS OBJECTS

Central to the HiPAC knowledge model is the concept of Event-Condition-Action (ECA) rules. The semantics of ECA rules are straightforward: when the event occurs (is *signalled*), evaluate the condition; and, if the condition is *satisfied*, execute the action. Integrity constraints, access constraints, derived data, alerters, and other active DBMS features can all be expressed as ECA rules. HiPAC uses an object-oriented data model (the details of which are unimportant for this paper), and rules are first-class database objects, subject to the same operations as user-defined objects (plus some special operations).

In this section, we describe the attributes of rule objects, the operations defined on rule objects, and the architectural implications of supporting ECA rules.

2.1 Rule Attributes

The attributes of rules are:

Event The event that triggers the rule (i.e., causes HiPAC to evaluate the rule's condition). Typed formal arguments may be defined for the event; these are bound to actual arguments when the event is detected. Event occurrences and the argument bindings are reported in an *event signal.*

Condition A collection of queries that are evaluated when the rule is triggered by its event.

Action The action that is executed when the rule is triggered and its condition is satisfied.

E-C Coupling A coupling mode that specifies when the condition is evaluated relative to the transaction in which the triggering event is signalled.

C-A Coupling A coupling mode that specifies when the action is executed relative to the transaction in which the condition is evaluated.

(Rules can have other attributes; the ones listed here are those which determine the processing.)

The event for a rule can be a primitive event, or a combination of primitive events. The primitive events are:

1. Database Operations: data definition, data manipulation, transaction control. The description of a database event specifies the operation type and its parameters. The signal includes the operation and its actual arguments (e.g., the object instances being modified, and the old and new values of the modified objects' attributes).

2. Temporal Events: absolute, relative, periodic. The description of a temporal event specifies an absolute time; some other event (the baseline) and a relative time offset from the baseline; or a baseline and a period; optional descriptive information may also be specified. The signal includes the absolute time at which teh event occurred and the optional information.

3. External Notification: application defined events. The description of an external event specifies arbitrary formal parameters from the application program. The signal includes the binding of these formal parameters to actual arguments in the execution of the application program in which the event occurs.

Primitive events can be combined using disjunction and sequence operators to specify composite events. The event specification can also be omitted from a rule definition. In this case, HiPAC derives the event specification from the condition.

The condition is a collection of queries expressed in an object-oriented DML. The queries may refer to arguments in the event signal. The condition is satisfied if all of these queries produce non-empty results. The results of these queries are passed on to the action, together with the argument bindings obtained from the event signal.

The action is a sequence of operations. These can be database operations or external requests to application programs.

The E-C coupling is the relationship, relative to transaction boundaries, between the triggering event and the condition evaluation. There are three possible coupling modes:

1. immediate: when the triggering event occurs, evaluate the condition immediately (i.e., preempt the processing of the remaining steps of the transaction) in the same transaction.

2. separate: when the triggering event occurs, evaluate the condition in a separate transaction.

3. deferred: evaluate the condition in the same transaction as the triggering event, but when that transaction terminates.

The same modes are available for the C-A coupling, which specifies the relationship, relative to transaction boundaries, between the evaluation of the condition and the execution of the action.

2.2 Rule Operations

Since rules are database objects, they are subject to the operations common to all database objects: creation, modification, deletion. In addition, there are operations specific to rules that affect rule processing:

Fire Evaluate the rule's condition and, if it is satisfied, execute the rule's action (subject to the coupling mode specifications).

Disable Disable automatic rule firing for its event (or some subset of the events that cause the rule to fire).

Enable Enable rule firing for its event.

HiPAC normally fires a rule automatically when its event occurs. A rule can also be manually fired using the "fire" operation. Automatic firing can be disabled using the "disable" operation, and re-enabled using the "enable" operation.

As database objects, rules are subject to transaction semantics. A transaction must obtain the appropriate lock on a rule before performing an operation on that rule. Firing requires a read lock. All operations that update rules (create, modify, delete, enable, disable) require write locks. (Note that we think of "enable" and "disable" as modifying a rule, insofar as they modify the ability of an event signal to fire the rule.)

2.3 Architectural Implications

To support ECA rules, HiPAC must first have the ability to determine when events have occurred. HiPAC must detect primitivedatabase events, receive signals for external and temporal events from application or system processes, infer complex events from the primitive events, and match the event signals against rule event specifications to determine which rules are triggered.

After determining that a rule has been triggered by an event, HiPAC must schedule condition evalution and action execution according to the rule's coupling modes.

Rule conditions can be complex, and rules with complex conditions can fire frequently. HiPAC must provide efficient condition evaluation, using techniques such as multiple query optimization, incremental evaluation, and materialization of derived data. (A range of techniques is described in [DAY88a].)

3. EXECUTION MODEL: RULE FIRINGS AND TRANSACTIONS

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 [HSU88] 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 based on coupling.

3.1 Nested Transactions

In a nested transaction model [MOS85] 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 a top transaction, commit. When a transaction aborts, its effects and the effects of all of its descendants are discarded.

3.2 Coupling Modes and Transactions

When a rule fires, a transaction is created and the rule's condition is evaluated in that transaction. If the rule's E-C coupling mode is immediate or deferred, then the transaction is a subtransaction of the transaction containing the triggering event. The parent transaction is suspended while this subtransaction executes. For immediate coupling, the condition evaluation subtransaction is created and executed at the point where the triggering event occurs. For deferred coupling, the condition evaluation subtransaction is created and executed just prior to its parent transaction committing. If the rule's E-C coupling mode is separate, then condition evaluation takes place in a top level transaction that executes concurrently with the triggering transaction.

If the rule's condition is satisfied, then another transaction is created and the rule's action is executed in that transaction. The particulars of this transaction are

determined by the rule's C-A coupling mode, in a manner analogous to that described above for condition evaluation.

If an event triggers more than one rule, then a condition evaluation transaction is created for each rule. For rules with the same event and E-C coupling mode, the condition evaluation transaction will execute concurrently. Similarly, for rules with the same event and C-A coupling, the action execution transactions will execute concurrently. Thus there is no conflict resolution policy that chooses a single rule to fire, or a serial order in which to fire all of the rules. Instead, all of the rules fire concurrently as sibling transactions. The correctness criteria is serializability, and this is enforced by the HiPAC transaction manager.

When a rule fires, its action can include an operation that triggers another rule. If both rules have immediate coupling for their conditions, then the original triggering transaction will have a subtransaction for the first rule's condition evaluation, and this subtransaction will in turn have a subtransaction for evaluating the second rule's condition. Thus, cascading rule firings produce a tree of nested transactions.

3.3 Architectural Implications

An active DBMS supporting ECA rules must have concurrency control and recovery mechanisms for the nested transaction model. (Algorithms are given in [HSU88].) It must create transactions for rule firings and schedule those transactions, as described above.

4. A PARADIGM FOR ACTIVE DBMS APPLICATIONS

Even in those DBMS's that provide some form of active database facilities, both the events that trigger actions and the actions that they trigger are limited to database operations. Consider triggers in System R and Sybase. The event for a trigger is an insert, update, or delete operation on a table; the action is expressed in SQL. In contrast, HiPAC allows rule events to be defined by the application, and allows rule actions to contain requests to applications. The result is a whole new paradigm for building database applications.

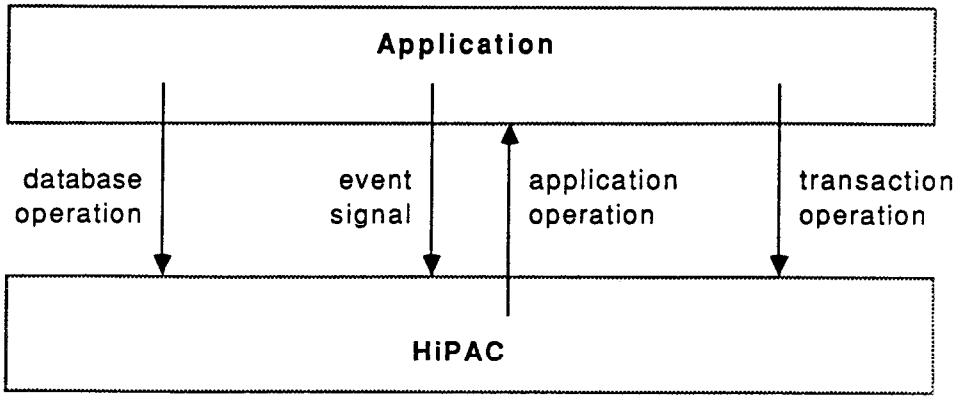

Figure 4.1 Interface Between an Application Program and HiPAC

4.1 The Interface Between Applications and HiPAC

Figure 4.1 depicts the interface between an application program and HiPAC. This interface is divided into four modules. Two of these provide the ususal DBMS functionality, and the other two are unique to HiPAC. The former are the modules that support operations on data and transactions. The latter are the modules that contain operations on events, and application-specific operations.

The operations on data encompass data definition as well as data manipulation. Since HiPAC uses an object-oriented data model, data definition consists of defining classes (types) and operations on instances of those classes. Applications manipulate data by invoking those operations through this interface. Note that the execution of such an operation on a database object is an event that can be used in defining a rule.

The operations on transactions are create, commit, and abort. Recall that HiPAC uses a nested transaction model. Applications can create and terminate subtransactions as well as top level transactions.

The event operations module provides two operations on events: define and signal. This interface allows applications to define and signal their own events. After an application-

specific event has been defined, it can be used in creating one or more rules. Then, when the application signals the event, HiPAC will fire the rule. The definition of an event specifies the data to be included in the event signal. This data can be accessed by the rule's condition and action.

The last module, application operations, allows a reversal of roles in which HiPAC becomes the client and the application becomes the server. HiPAC allows requests to application programs to be included in the action for a rule. When the rule fires and the action is executed, HiPAC will "call" the application program the execute the operation. This provides a new medium for interaction among application programs. One program can can send a request to another program either directly (e.g., IPC message), or indirectly through a rule firing.

A mechanism must be provided for communicating requests from the Rule Manager to applications. In most systems, the DBMS and application run in different address spaces (processes), sometimes on different machines. The communication mechanism is already present for sending requests from the applications to the DBMS, and replies back to the applications. In most cases, the same underlying operating system facility can be used to reverse the direction in which requests and replies are transmitted.

4.2 An Example Application

The first application implemented over HiPAC was a Securities Analyst'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 analyt's instructions. This application is shown in Figure 4.2. It consists of programs and rules.

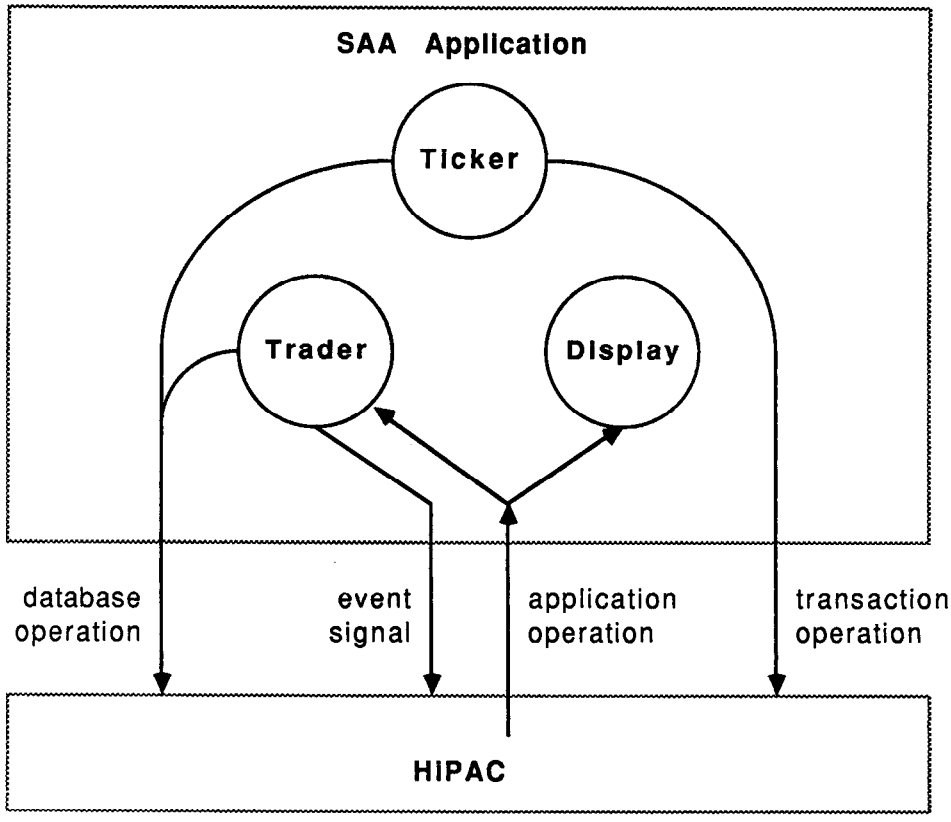

Figure 4.2 An Example of an Active Database Application

The SAA consists of three application programs:

Ticker Updates the current prices of securities in the database based on price quotes read from a wire service.

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 each 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 analyt'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 A
Coupling:	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 HiPAC 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 flows from one application program to another through the DBMS. With ECA rules, application defined events, and calls to application programs in rule actions, HiPAC 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.

5. FUNCTIONAL COMPONENTS

To implement the HiPAC knowledge model and execution model, a DBMS must provide object-oriented data managment and nested transactions. It must support the semantics of the rule object class. This includes detecting events, determining which rules to fire when events are reported, scheduling condition evaluation and action execution according to rule coupling modes, and performing these activities in nested transactions.

Figure 5.1 depicts the functional components of the HiPAC architecture. These are:

Object Manager	Provides object-oriented data management.
Transaction Manager	Provides nested transactions.
Event Detectors	Detect primitive events and signal them to the Rule Manager.
Rule Manager	Maps events to rule firings, and rule firings to transactions.
Condition Evaluator	Evaluates rule conditions.

The Object Manager and Transaction Manager together provide the functionality of an object-oriented DBMS, plus nested transactions. ECA rules are implemented by the Event Detectors, Rule Manager, and Condition Evaluator.

The overall flow of control and data is as follows. First, an event relevant to some rule (e.g., a database operation) is detected by an Event Detector and reported to the Rule Manager. The Rule Manager determines which rule to fire, and schedules condition evalution for these rules based on their condition coupling modes. The Rule Manager calls on the Transaction Manager to create a transaction for condition evalution, and calls on the Condtion Evaluator to evaluate the rule's condition. If the condition is satisfied, the Rule Manager calls on the Transaction Manager to create a transaction for executing the action.

5.1 Object Manager

The Object Manager provides object-oriented data management. It supports the definition of object types and operations on instances of those types, and is responsible for executing those operations. In the course of executing database operations, the Object Manager calls on the Transaction Manager to obtain locks, and acts as an event detector, reporting database operations to the Rule Manager.

The interface to the Object Manager consists of a single operation:

Execute Operation Execute a database operation (DDL or DML) on one or more database objects. The parameters are the database objects and the transaction in which to perform the operation.

This interface is used by applications, the Rule Manager, and the Condition Evaluator.

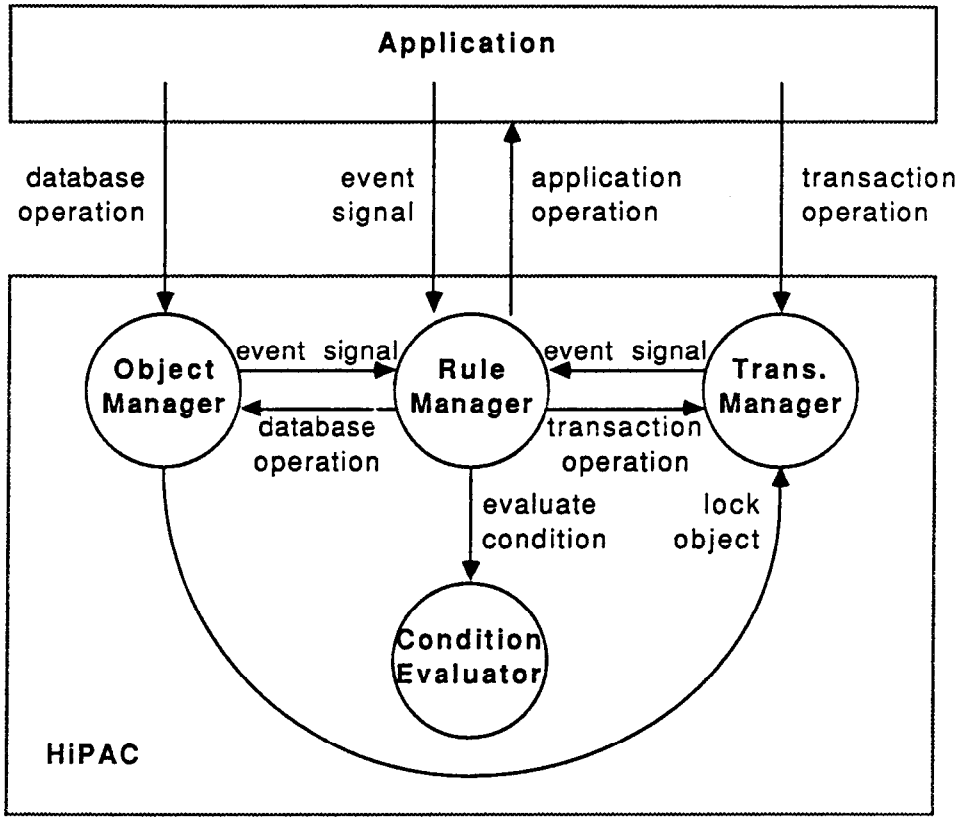

Figure 5.1 the HiPAC Functional Components

5.2 Transaction Manager

The Transaction Manager implements the HiPAC nested transaction model. It is responsible for creating and terminating transactions, and for concurrency control. In addition, it acts as an event detector, reporting transaction termination to the Rule Manager.

The interface to the Transaction Manager consists of three operations:

Create Transaction Create a (top level or nested) transaction.

Commit Transaction Commit a (top level or nested) transaction.

Abort Transaction Abort a (top level or nested) transaction.

This interface is used by applications and the Rule Manager.

5.3 Event Detectors

Event Detectors are responsible for reporting the occurrence of primitiveevents to the Rule Manager. There are event detectors for database events (in the Object Manager and Transaction Manager), for temporal events, and for application-defined events. Particular event detectors differ in the type of events that they detect, how these events are described when programming the event detector, and the contents of the event signal that is passed to the Rule Manager when an event is reported.

When a rule is created, the appropriate event detector(s) is (are) programmed to detect and report the primitive events that can trigger the rule. The event detector(s) is (are) instructed to cease detecting the event when the rule is deleted (if there is no other rule with the same event). When a rule is disabled, event detection for the rule is

disabled. When the rule is enabled again, event detection is enabled.

The interface to an Event Detector consists of the following operations:

Define Event Program the event detector to report the occurrence of a primitive event. The parameter is a description of the event.

Delete Event Cease detecting and signalling an event.

Enable Event Suspend the detection and signalling of an event, specified as a parameter.

Disable Event Resume the detection and signalling of an event, specified as a parameter.

This interface is used by the Rule Manager.

5.4 Rule Manager

The Rule Manager is responsible for firing the appropriate rules when an event is detected. That is, it determines which rules to fire, and schedules condition evaluation and action execution for those rules according to their coupling modes. The Rule Manager calls on the Transaction Manager to create the transaction used for condition evaluation and action execution, and it calls on the Condition Evaluator at the appropriate points to determine which conditions are satisfied. The Rule Manager is also responsible for suspending triggering transactions until all of their subtransactions (for immediate and defrred rule firings) have terminated.

The interface to the rule manager consists of a single operation:

Signal Event Report the occurrence of an event. The parameters are the event, its signal, and the transaction in which the event occurred.

This interface is used by the Event Detectors and the Transaction Manager.

5.5 Condition Evaluator

After an event has been detected, the Condition Evaluator is responsible for efficiently determining which rule conditions are satisfied (among the rules triggered by the particular event). The Condition Evaluator uses techniques such as multiple query optimization and view materialization to do this. The data structure used for this purpose is called a *condition graph*. The Condition Evaluator can be thought of as having two functions relative to these condition graphs. One is to maintain the condition graphs as rules are created, deleted, enabled, and disabled. The other is to use the condition graphs to determine which rule conditions are satisfied when an event occurs.

The interface to the Condition Evaluator consists of the following operations:

Add Rule Add a rule to the condition graph. The message includes the event, the condition, and the coupling mode for condition evaluation. The output includes the events that must be signaled.

Delete Rule Remove a rule from the condition graph.

Evaluate Conditions Determine which rules have been satisfied. The input is the event signal, coupling mode, and previous database state (deferred and separate condition evaluation only).

This interface is used only by the Rule Manager.

6. RULE PROCESSING

In this section we describe how the HiPAC components described above interact in carrying out operations on rules. The operations described here are creating a rule, signalling an event, and committing a transaction.

6.1 Rule Creation

Rule creation is initiated when an application issues a request for the rule creation operation. The request is handled by the Object Manager. The Object Manager creates the rule object, obtains a write lock on it, and signals the "create rule" event to the Rule Manager. The Object Manager then waits for a reply from the Rule Manager.

First, the Rule Manager issues an "add rule" request to the Condition Manager. Then it issues "define event" requests to the appropriate Event Detectors. Finally, it adds the rule and events to its mapping from events to rules. At this point, the Rule Manager replies to the event signal, and the Object Manager resumes processing the original request.

6.2 Event Signal Processing

When the event for a rule occurs, an Event Detector issues an event signal to the Rule Manager (because the Rule Manager programmed the Event Detector to do so when the rule was created). The operation that triggered the event signal is suspended. The Rule Manager then determines which rules to fire by consulting the mapping that it maintains between events and rules. The Rule Manager divides these rules into three groups according to their coupling mode for condition evaluation.

For each rule firing with separate condition evaluation, the Rule Manager obtains a new top level transaction (from the Transaction Manager) and calls on the Condition Evaluator to evaluate the rule's condition in that transaction. All of these transactions execute concurrently, each in its own thread of execution. If a rule's condition is satisfied, the thread of execution will process the rule's action. Meanwhile, the Rule Manager continues with the processing of the original event signal.

For each rule firing with deferred condition evaluation, the Rule Manager saves adds the rule and event signal to a set of deferred rule firings that is associated with the transaction in which the triggering event occurred. This set of rule firings is processed later, when the transaction commits (as described in the next section).

For each rule firing with immediate condition evaluation, the Rule Manager obtains a subtransaction (of the triggering transaction (from the Transaction Manager) and calls on the Condtion Evaluator to evaluate the rule's condition in that subtransaction. When all conditions have been evaluated, actions are executed for those rules whose conditions were satisfied.

When all immediate condition evaluation and action execution is completed, the Rule Manager replies to the Event Detector. At this point the operation that originally caused the event signal resumes.

6.3 Transaction Commit Processing

Transaction commit is initiated by a "commit transaction" request to the Transaction Manager. As part of commit processing, the Transaction Manager issues an event signal to the Rule Manager. This event signal identifies the transaction that is terminating.

The Rule Manager maintains a set of deferred rule firings for each transaction. When the Rule Manager receives the commit event signal, it first gets the corresponding set of deferred rule firings. This set is divided into two subsets according to whether it was the condition or action that was deferred. For each of the former, the Rule Manager calls on the Condition Evaluator to evaluate the rule's condition. For the latter, the Rule Manager simply executes the action.

When all deferred rule firings have completed, the Rule Manager replies to the commit event signal, and the Transaction Manager resumes commit processing.

7. IMPLEMENTATION STATUS AND FUTURE WORK

We have presented an architecture for HiPAC, an active DBMS with ECA rules. This architecture supports the knowledge model, execution model, and condition monitoring algorithms described in the HiPAC research papers. In doing so, it specifies two new forms of interaction between an application and the DBMS. Application programs signal events, and HiPAC makes requests to application programs in executing rule actions. This leads to a new paradigm for building applications over an active DBMS. Control logic is encoded in rules rather than software. HiPAC becomes a medium for the flow of control, as well as data, between application programs.

We are currently implementing a HiPAC prototype and applications using Smalltalk-80. Initially, we are concentrating on the knowledge model and the execution model. The Rule Manager and Transaction Manager are implemented. The Object Manager and Condition Evaluator are being designed. The Object Manager will support the Probe data model and algebra [MANN86]. In the interim, rule conditions and actions are expressed as Smalltalk-80 blocks. Concurrent execution of transactions is implemented using Smalltalk-80 processes, which are "light weight".

In our paradigm for active database applications, many control functions are implemented in ECA rules rather than software. While this tends to simplify the software, it also produces large sets of rules. As the rule base for an application grows, problems due to unexpected interactions among rules become more likely. Tools and techniques have evolved for dealing with large, complex programs (e.g., modularity, data abstraction, debuggers). Future research will produce the tools and techniques needed to develop large, complex rule bases.

8. REFERENCES

[BUC89] A. Buchmann et al. "A Framework For Integrating Time-Critical Scheduling and Database Transaction Scheduling." To appear in *Proc. 5th International Conference on Data Engineering*, February 1989.

[COD73] CODASYL Data Description Lnaguage Committee. *CODASYL Data Description Language Journal of Development* June 1973. NBS Handbook 113 (1973).

[DAY88a] U. Dayal et al. "HiPAC: a Research Project in Active, Time-Constrained Database Management, Interim Report." Technical Report XAIT-88-02, Xerox Advanced Information Technology, June 1988.

[DAY88b] U. Dayal, A. Buchmann, and D. McCarthy. "Rules Are Objects Too: A Knowledge Model For An Active, Object-Oriented Database System." *Advances in Object-Oriented Database Systems*, September 1988, pp. 129-143.

[ESW75] K. P. Eswaran and D. D. Chamgerlain. "Functional Specifications of a Subsystem for Data Base Integrity." *Proc. 1st Int'l Conf. on Very Large Data Bases* (September 1975).

[ESW76] K. P. Eswaran. "Specifications, Implementations, and Interactions of a Trigger Subsystem in an Integrated Data Base System." IBM Research Report RJ1820 (August 1976).

[HSU88] M. Hsu, R. Ladin, D. McCarthy. "An Execution Model For Active Database Management Systems." *Proc. 3rd International Conference on Data and Knowledge Bases*, June 1988, pp. 171-179.

[MAN86] F. Manola and U. Dayal. "PDM: An Object-Oriented Data Model." *Proc. Int'l Workshop on Object-Oriented Database Systems*, (1986).

[MOR83] M. Morgenstern. "Active Databases as a Paradigm for Enhanced Computing Environments." *Proc 9th Int'l Conf. on Very Large Data Bases, Florence*, pp. 34-42 (October 1983).

[MOS85] E. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*, MIT Press (1985).

[STO86] M. Stonebraker et al. "A Rule Manager For Relational Database Systems." *The POSTGRES Papers*, Univ. of California, Berkley, Ca. Electronics Research Lab, Memo No. UCB/ERL M86/85 (1986).

[SYB87] Sybase, Inc. *Transact-SQL User's Guide* (1987).