AFRL-IF-RS-TR-1999-20
Final Technical Report
February 1999

AN AGENT-BASED APPROACH TO EXTENDING THE NATIVE ACTIVE CAPABILITY OF RELATIONAL DATA BASE SYSTEMS

University of Florida

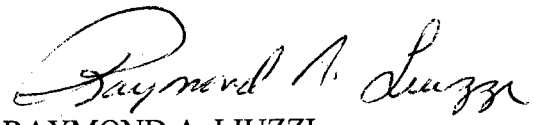
S. Chakravarthy and L. Li

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-20 has been reviewed and is approved for publication.

APPROVED: 

RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: 

NORTHRUP FOWLER, III, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1999	Final Sep 97 - Sep 98

4. TITLE AND SUBTITLE
AN AGENT-BASED APPROACH TO EXTENDING THE NATIVE ACTIVE CAPABILITY OF RELATIONAL DATABASE SYSTEMS

5. FUNDING NUMBERS
C - F30602-97-C-0306
PE - 62232N & 62702F
PR - R472
TA - 00
WU - P1

6. AUTHOR(S)
S. Chakravarthy and L. Li

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Florida
Department of Computer and Information Science
P.O. Box 116120
Gainesville FL 32611-6120

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NAVY/NCCOSE
53245 Patterson Road
San Diego CA 92152-7151

Air Force Research Laboratory/IFTB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-1999-20

11. SUPPLEMENTARY NOTES
NAVY/NCCOSE/Leah Wong/(619) 553-4127
Air Force Research Laboratory Project Engineer: Raymond A. Liuzzi/IFTB/(315) 330-3577

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Event-condition-action (or ECA) rules are used to capture active capability. While a number of research prototypes of active database systems have been built, ECA rule capability in Relational DBMSs is still very limited. In this report, we address the problem of turning a traditional database management system into a full-fledged active database system without changing the underlying system. The advantages of this approach are: transparency; ability to add active capability without changing the client programs; retain relational DBMS's underlying functionality; and persistence of ECA rules using the native database functionality.

This report describes how complete active database semantics can be supported on an existing SQL Server (Sybase, in our case) by adding a mediator, termed ECA Agent, between the SQL Server and the clients. ECA rules are fully supported through the ECA agent without changing applications or the SQL Server. Composite events are detected in the ECA Agent and actions are invoked in the SQL Server. Events are persisted in the native database system. ECA Agent is designed to connect to SQL Server by using Sybase connectivity products. The architecture, design, and implementation details are presented.

14. SUBJECT TERMS
Databases, Knowledge Base, Artificial Intelligence, Software, Computers

15. NUMBER OF PAGES
36

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

List of Figures

An Agent-Based Approach to Extending the Native Active Capability of Relational Database Systems

Abstract

Event-condition-action (or ECA) rules are used to capture active capability. While a number of research prototypes of active database systems have been built, ECA rule capability in Relational DBMSs is still very limited. In this paper, we address the problem of turning a traditional database management system into a full-fledged active database system without changing the underlying system. The advantages of this approach are: transparency; ability to add active capability without changing the client programs; retain relational DBMS's underlying functionality; and persistence of ECA rules using the native database functionality.

In this paper we describe how complete active database semantics can be supported on an existing SQL Server (Sybase, in our case) by adding a mediator, termed ECA Agent, between the SQL Server and the clients. ECA rules are fully supported through the ECA Agent without changing applications or the SQL Server. Composite events are detected in the ECA Agent and actions are invoked in the SQL Server. Events are persisted in the native database system. ECA Agent is designed to connect to SQL Server by using Sybase connectivity products. The architecture, design, and implementation details are presented in this paper.

1 Introduction

A traditional database is a passive repository of data where the DBMS only execute the explicit transactions and queries asked by a user or an application program. The DBMS uses a query-driven mechanism. This traditional view of databases as information repositories, which are used for storing and retrieving required information, works for many applications. However, the need for having a database system capable of reacting to specific situations without user or application intervention has been recognized in several newer applications. Frequently mentioned examples of such applications are network management, computer-integrated manufacturing (CIM), commodity trading, air-traffic control, plant and reactor control, tracking, monitoring of toxic emissions, workflow and process control, etc. Active database systems have been proposed as a new data management paradigm to satisfy these kinds of needs so that the system can monitor the state of the database for particular events, and trigger appropriate and timely responses when events occur. This can be done by defining event-condition-action (ECA) rules which are stored as part of the database [DAY94]. Active database systems, by the way of rule definition, event detection and action execution, are not only driven explicitly by a user or an application, but they are also able to recognize specific situations (in the database and external to the database) and react to them.

So far, a number of research prototypes of active database systems have been developed, such as HiPAC [CHA89], Ariel [HAN92], Sentinel [CHA93], Starburst [WID91], Exact [DPG91], Postgres [STO91], PEARD [JAH96], SAMOS [GAT92] etc. Most of them are developed from scratch or integrated directly into the kernel of the DBMS. The integrated approach provides the following advantages: [CHA89]

- Do not require any changes to existing applications.
- DBMS is responsible for optimizing ECA rules.
- DBMS functionality is extended.
- Modularity/maintenance of applications is better and maintenance is easier.

However, the implementation of an integrated approach requires access to the internals of a DBMS into which the active capability is being integrated. This requires access to source code, makes the cost of integrated approach very high and requires a long integration time as well. Hence, most integrated systems are research prototypes.

There are alternative approaches to support active capability, such as Embedding Situation Check in application code and Polling. However these suffer from many limitations. For example, the Embedded Situation Check approach requires extra code in all applications. Since modularity is compromised, the management and maintenance of applications is difficult. Also, constraints and business rules are not clearly separated from the application [CHA89]. The polling for a relational DBMS is discussed in Chap 2.

To the best of our knowledge, there is no commercial DBMS that supports full active capability, although the promises of active database technology is well understood and is considered significant. Currently, relational DBMS are widely used. Most users are familiar with RDBMS, and the advantages of relational DBMS are well known. Rule capability is provided in many commercial systems, but it is not sufficient as it only provides basic triggering capabilities.

The challenge is to turn a commercial database management system into a true active database system without making any changes to the underlying system. This paper introduces an approach, which adds a mediator to the Sybase SQL Server to provide ECA functionality in the SQL Server (shown in Figure 1).

Although there are some advantages to integrating the active capability to the DBMS kernel, the use of a mediated architecture outside of the SQL Server provides many benefits, including:

- Transparency: The clients do not feel the mediator.
- System functionality: None of the existing DBMS's functionality would be lost.
- Extensibility: Additional functionality and a distributed architecture can be added later.
- Scalability: The architecture is scaleable.
- Portable: Once the mediated approach has been developed, it may be ported to
- other Relational DBMSs.

The contributions of this paper are as follows:

- Present a mediated approach that significantly extends Sybase ECA functionality:

1. A client can create composite events and triggers on them.
2. Reuse of previously defined events (both primitive & composite).
3. Drop triggers associated with primitive or composite events.
4. A client can create multiple triggers on the same event.
5. Once events are created, they become persistent in the database system.
6. All primitive events and composite events can be detected, and actions are invoked within SQL Server.

- Well defined mediated approach (ECA Agent), and present the architecture of ECA Agent.
- Discuss how ECA Agent is implemented.
- Provide an API for SQL Server, and show how this architecture is also suitable for other SQL Servers.

The remainder of this paper is organized as follows. Background on sentinel and snoop are presented in Section 2. In Section 3 we discuss the architecture and functional components of the ECA agent. The implementation of the ECA agent is described in Section 4. The implementation details of triggers that use Primitive and Composite Event are described in Section 5. We then draw conclusions and discuss the future work in Section 6.

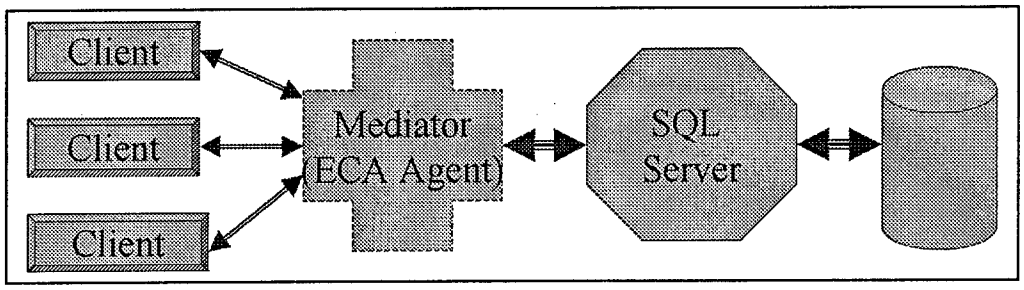


Figure 1. Architecture of Mediated Approach

2 Background

Sentinel is an Object Oriented Active Database System. It uses Open OODB as its platform, and integrates ECA rules capability into the kernel of Open OODB. The integration enhances Open OODB from a passive OODB to an active one.

One of the modules of Sentinel is the LED (or the local event detector). Parts of this component (especially the primitive event detection) are tightly integrated into the kernel of the OpenOODB. LED is responsible for primitive event detection as well as composite event detection within an application or address space. In Sentinel, a method can be specified as a primitive event, and the occurrences of the primitive events are notified to the local event detector when the method is invoked. Composite events defined within an application (using Snoop as the specification language) are detected by using a sequence of primitive events detected according to the operator semantics as well as the specified parameter context of the composite event [LEE96].

2.1 Snoop

SNOOP [CHM94] is an event specification language for Sentinel. It provides the semantics of primitive events and event operators used to express composite events. Snoop supports database, temporal, periodic, explicit and composite events. Since Snoop is model independent, we use it as our event specification language. Snoop BNF is given below. As the focus of this paper is not Snoop, we do not discuss it further. For more details on Snoop, please refer to [CHK94].

Event_exp ::= E1
E1 ::= E1 **OR** E2 | E2
E2 ::= E2 **AND** E3 | E3
E3 ::= E3 **SEQ** E4 | E4
E4 ::= **NOT** (E1, E1, E1)
| **A** (E1, E1, E1)
| **A*** (E1, E1, E1)
| **P** (E1, [*time string*], E1)
| **P** (E1, [*time string*]: *parameter*, E1)
| **P*** (E1, [*time string*], E1)
| **P*** (E1, [*time string*]: *parameter*, E1)
| [*time string*]
| E1 PLUS [time string]
| (E1)
| *event_name*
event_name::= *name*
| *Eventname*: *Objectname*
| *Eventname*:: *AppId*
AppId ::= *Sitename__Appname*
| Identifier
Name ::= Identifier
Eventname ::= Identifier
Objectname ::= Identifier

Snoop supports the following parameter contexts: *recent, continuous, cumulative* and *chronicle* [CHA94]. These contexts are precisely defined using the notion of initiator and terminator events. An initiator of a composite event is a constituent event that can start the detection of the composite event whereas a terminator is a constituent event that can detect the occurrence of the composite event. All Parameter Contexts are supported in the ECA Agent.

2.2 Limitations of Triggers in commercial systems

Trigger capability is supported in most commercial Relational database systems. Generally, this capability only supports primitive events. Following is a list of the trigger restrictions of Sybase (which is also true for most of the commercial relational database systems):

- Definition of complex data types is not allowed.
- There is no direct access to C, to other programs, or to the underlying operating
- system.
- Only atomic values (and not tables) may be passed as parameters to stored
- procedures.
- A trigger cannot be applied to more than one table.
- Each new trigger on a table for the same operation (insert, update, or delete)
- overwrites the previous one. No warning message is given before the overwrite occurs.
- An event cannot be named and reused
- Composite events cannot be specified

Some of the restrictions make the system inflexible. For example, a trigger cannot be applied to more than one table, or each new trigger on a table for the same operation overwrites the previous one, etc. The ECA Agent overcomes some of these restrictions so that it can provide SQL Server a complete Active Database engine.

3 ECA Agent Architecture

ECA Agent is a multithread program. It is positioned between clients and the SQL Server so that the SQL Server can provide ECA capabilities with full transparency. From the users'point of view, the ECA Agent is a Virtual Active SQL Server. Not only can it provide all functions of the native SQL Server, but it also provides the ECA functions that active database requires. From a system's point of view, the ECA Agent is a middleware/mediator that connects the client and SQL Server and provides ECA service if necessary.

The architecture of ECA Agent is shown in Figure 2. There are seven functional modules in the ECA Agent:

- *General Interface (GI)*: GI (which is added to the Gateway Open Sever or GOS) is responsible for the connection between a client and SQL Server. It provides the same interface as SQL Server to accept client commands and returns the results to the client. GOS also accepts SQL requirements from the other parts of ECA Agent and forwards them to SQL Server. Results are returned to the corresponding part. Fully transparency is provided here.

- *Language Filter*: All client commands flow through the Language Filter. The ECA commands are separated and sent to the ECA Parser while other SQL commands are sent back to the Gateway Open Server. Language Filter is also responsible for filtering different types of ECA commands. (i.e. primitive event command, composite event command, drop trigger command, etc.)
- *ECA Parser*: ECA commands are filtered into the ECA Parser from the Language Filter. The ECA Parser scans and parses the commands. If there is no syntax error, the ECA Parser will create corresponding events and rules which depend on the Local Event Detector, send corresponding SQL to the Gateway Open Server, and send specifications of ECA rules to the Persistent Manager for persistent storing events and rules. If a parse error occurs, an error message is returned to Language Filter.
- *Local Event Detector (LED):* Since the SQL Server (trigger) detects only primitive events, LED is mainly responsible for composite event detection.
- *Persistent Manager*: All events and rules defined by a client need to be persistent. The information is stored in the following system tables using the SYBASE database:

  1. SysPrimitiveEvent: It is used to store information of primitive events.
  2. SysEcaTrigger: All triggers are stored here.
  3. SysCompositEvent: Information of composite events is stored in this table.

  Persistent Manager stores all ECA information into these tables when it receives information from ECA Parser. On ECA Agent starting or recovery, Persistent Manager restores and creates all events and rules from these tables.

- Event Notifier: As soon as a primitive event occurs, SQL Server sends notification to the Event Notifier. The Event Notifier is responsible for receiving the notification, formatting it and sending the formatted notification to the LED.

- *Action Handler*: Once an event occurs, it calls the actions defined on this event. The Action Handler processes these actions, changes them into corresponding SQL commands (call corresponding stored procedures), and send SQL commands to the Gateway Open Server. The Gateway Open Server will send them to the SQL Server, get the results from the Gateway Open Server, and send them to the client as appropriate

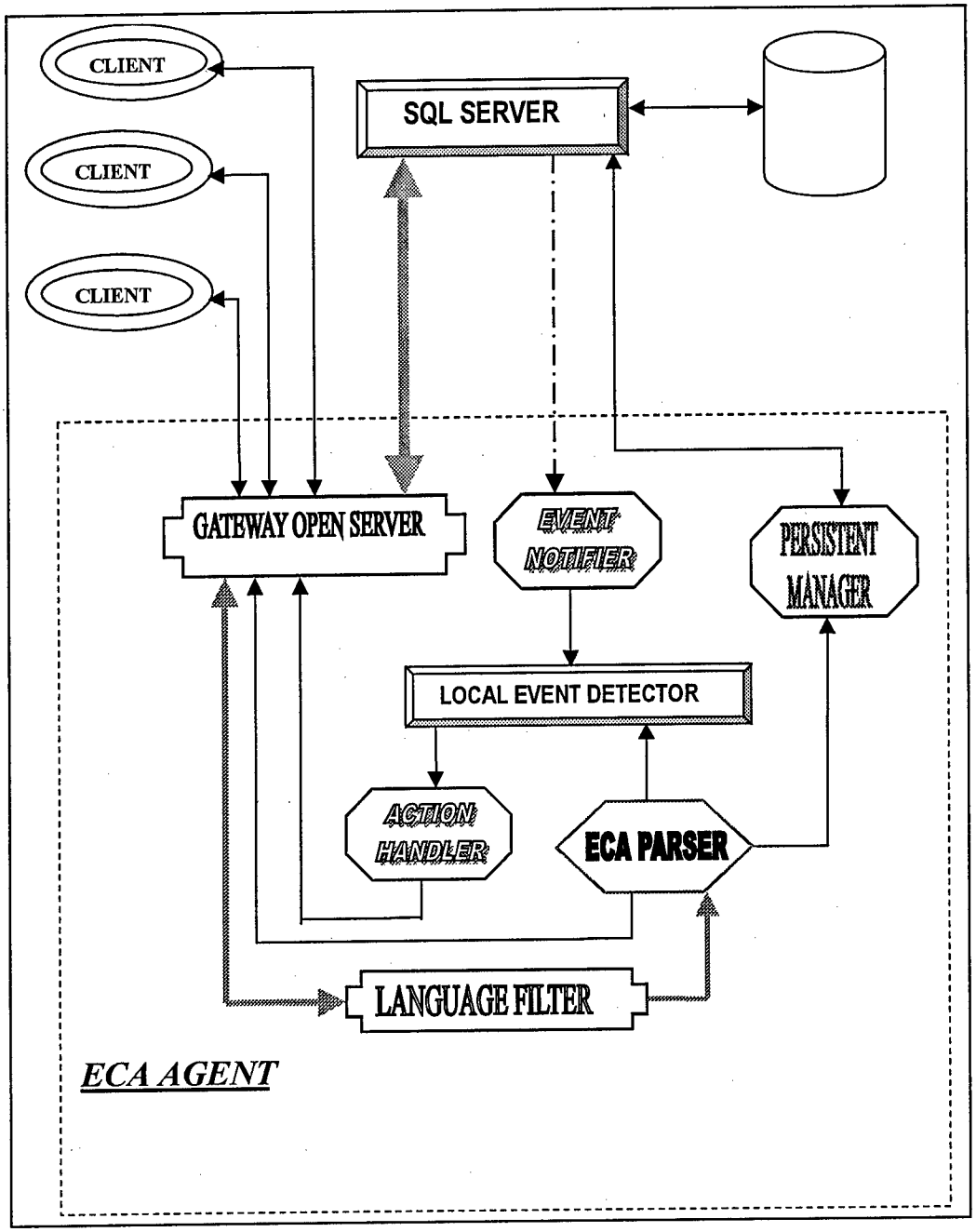


Figure 2. Architecture of an ECA agent

3.1 Module Interaction

The ECA Agent is responsible for two major functions: "create ECA rules" and "event notification and action."

The workflow of "create ECA rules" is shown in Figure 3. It includes seven steps:

1. The command goes to the Gateway Open Server.

2. The Gateway Open Server forwards the command to the Language Filter.
3. The Language Filter checks if the command is an ECA command. If so, the Language Filter scans the command and sends the command to the ECA Parser. Otherwise, the command is returned to the Gateway Open Server. And the Gateway Open Server Client provides input for creating a new ECA rule.
4. forwards the command to SQL Server and returns the result to the client.
5. The ECA Parser analyzes the command and checks for errors. If an error is detected, a message is returned to the Gateway Open Server. If no error is found, the ECA Parser creates event graphs using the LED, sends the new SQL commands to the Gateway Open Server, and forwards persistent requirements to the Persistent Manager.
6. if necessary, then returns the results to the client.
7. If the Persistent Gateway Open Server sends SQL commands to the SQL Server and Persistent Manager receives the persistent requirement, it persistently stores ECA rules.

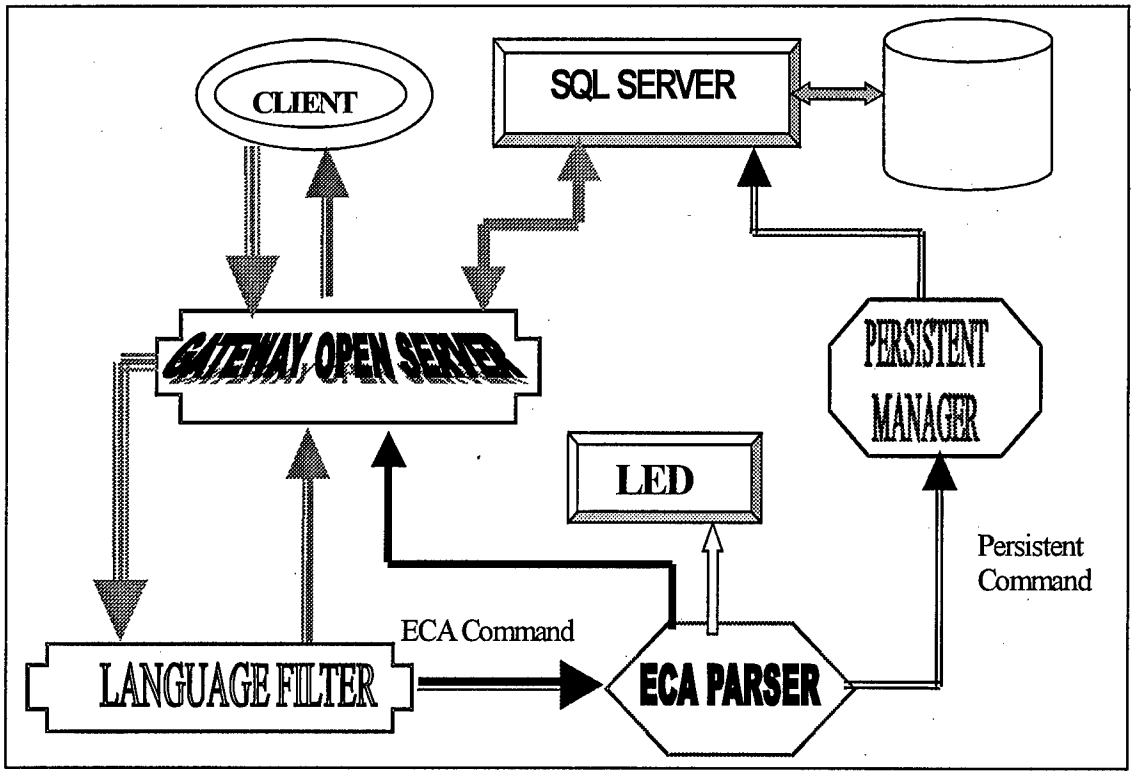


Figure 3. Control Flow for Creating ECA Rules

Figure 4 shows the workflow of "event notification and action." There are six steps for event notification and action:

- The client sends SQL commands to the Gateway Open Server. If the commands are not ECA commands, the commands will pass through to the SQL Server.
- If the commands invoke triggers, the SQL Server sends a notification to the Event Notifier.

- Event Notifier receives the notification from SQL Server, decodes the notification message, and notifies the LED.
- LED, after receiving the notification detects if occurrences of one or more events. If so, LED sends event information to the Action Handler.
- Action Handler processes event information, changes it into SQL commands, and sends it to Gateway Open Server.
- The Gateway Open Server sends the commands to the SQL Server and returns the results to the client.

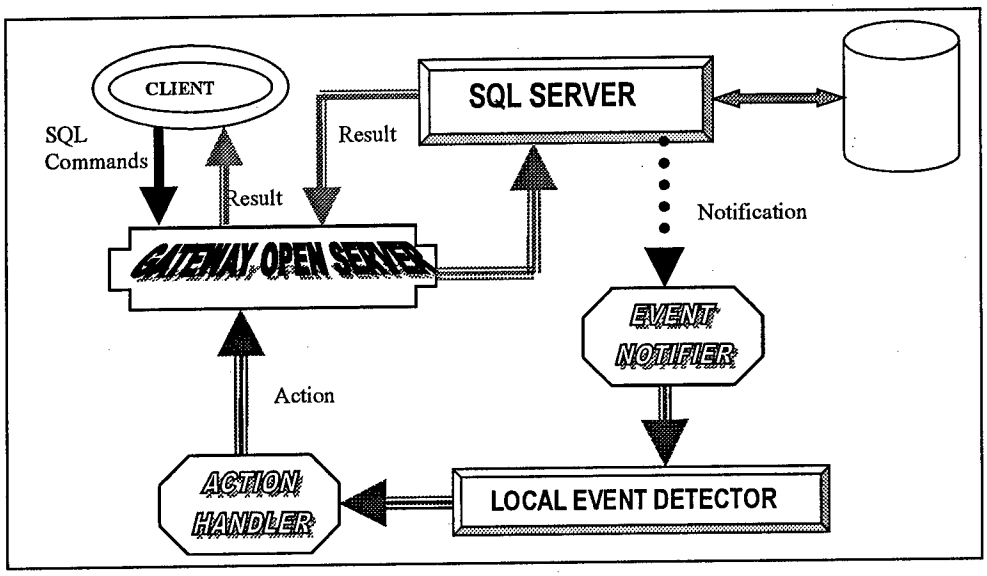


Figure 4. Control Flow of Event notification and Action

4 Implementation of the ECA Agent for Sybase

Sybase provides an OpenServer Library named Open Server™ Server-Library/C [SYB96]. The library is a non-preemptive multithread C library available from Sybase upon which the SQL Server is based. In fact, it is the basis for most Sybase server and middleware products including OmniSQL Server, all Sybase gateways, and Replication Server. The library allows for a C program to become a server to multiple Sybase client programs. This Server is called Open Server. It allows the programmer to authenticate logins, receive language requests, receive procedure calls, and return results in Tabular Data Stream (TDS) format, which all Sybase clients can receive. The following libraries may be used in Open Server:

- Open Client Client-Library/C
- Open Server™ Server-Library/C

Depending on its function, an Open Server's position in the client/server architecture is different. There are three functional categories for OpenServer: standalone, auxiliary, or gateway. Gateway Open Server is used as General Interface for the ECA Agent. By using Gateway Open Server, the mediator is truly transparent to the client. And there are no limitations on this architecture.

Once the Gateway Open Server starts, it generates a thread that connects to the SQL Server directly by using Client-Library. The thread is a Persistent Manager that will control and manage all persistent ECA rules. The connection with the SQL Server should be granted high privilege, for example DBA, so that it can have higher privilege to create or delete system tables than ordinary users.

The Persistent Manager is responsible for the management of ECA rule database. Listed below are functions of the Persistent Manager:

- Maintain ECA Agent system tables.
- Recovery of ECA rules.
- Persist ECA rules.
- On system startup, restore all ECA rules.
- Add or delete ECA rules from ECA Agent system tables.
- Keep track of the occurrence of each event.

To implement these functions, three system tables are created for the ECA Agent. Figure 5 shows the structure of table SysPrimitiveEvent that is used to store all primitive events. The structure of table SysCompositEvent is shown in Figure 6. This table is used to store all composite events. Table SysEcaTrigger whose structure is shown in Figure 7 stores all triggers.

Column_name	Type	Length	Nulls
dbName	varchar	30	NULL
userName	varchar	30	NULL
eventName	varchar	30	NULL
tableName	varchar	30	NULL
operation	varchar	20	NULL
timeStamp	datetime	8	NULL
vNo	int	4	NULL

Figure 5. Schema of SysPrmitiveEvent Table

Column_name	Type	Length	Nulls
dbName	varchar	30	NULL
userName	varchar	30	NULL
eventName	varchar	30	NULL
eventDescribe	text	text	NULL
timeStamp	datetime	8	NULL
coupling	char	10	NULL
context	char	10	NULL
priority	char	10	NULL

Figure 6. Schema of SysCompositeEvent Table

Column_name	Type	Length	Nulls
dbName	varchar	30	NULL
userName	varchar	30	NULL
triggerName	varchar	30	NULL
triggerProc	text	text	NULL
timeStamp	datetime	8	NULL
eventName	varchar	30	NULL

Figure 7. Schema of SysEcaTrigger Table

Since system tables are created in the SYBASE database system, they are maintained in the SYBASE database system based upon SYBASE storage management. Persistent Manager is responsible for managing the data in these tables and providing the persistent service to ECA Agent.

The functions of Persistent Manager are discussed in the previous sections. Figure 8 shows how Persistent Manager works and how it is implemented.

When the ECA Agent starts, it creates a thread that connects to the SQL Server by using functions in the Client-Library. The Persistent Manager runs in this thread (dark square in Figure 8). It creates ECA rules from the system tables of the ECA Agent.

After the ECA Agent starts, the Persistent Manager waits for the requirement from ECA Parser. If there is an add or delete ECA Rules requirement, Persistent Manager will persist the ECA Rules to the system tables or delete them from system tables.

5 Primitive and Composite Trigger Implementation

To provide user transparency, the syntax of an ECA Rule definition in the ECA Agent is almost the same as a trigger definition in SQL except for the concept of an event. Figure 9 shows the syntax of the Primitive Event definition. We can see the only difference here is the keyword **event**.

A primitive event is defined on a table for an operation (delete, update, and insert). A trigger can be defined on any event. The default coupling mode is **RECENT**, and the default parameter context is **IMMEDIATE**. The action function is written in SQL code, "**as**" is the keyword of action. The action is invoked in the SQL Server. Once an event is defined, the user can define additional triggers on the event. Figure 10 shows the Syntax of defining triggers on a previously defined event.

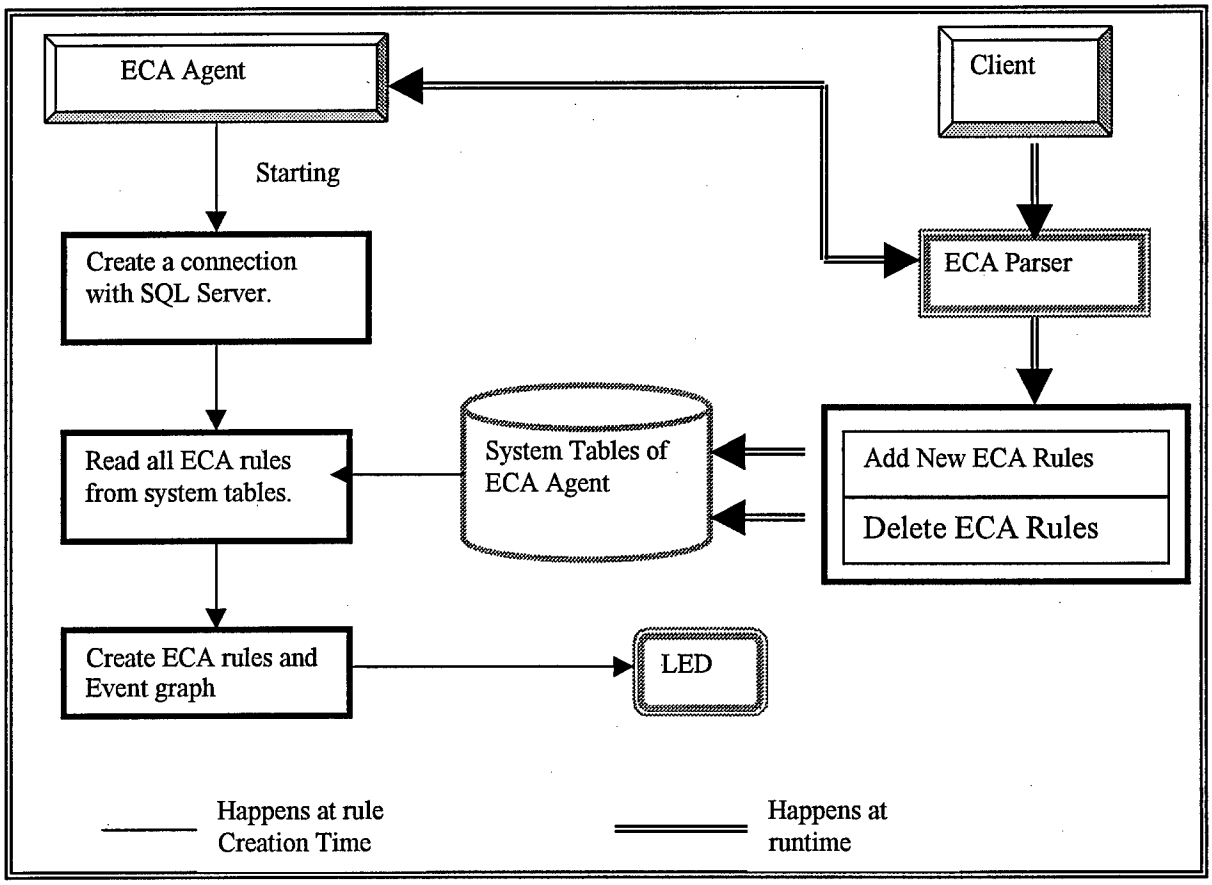


Figure 8. Implementation of the Persistent Manager

5.1 Naming

A user can assign a name for an object (a trigger or an event) in the system. Since SYBASE supports a multi-user, multi-database environment, we cannot use the name that the user assigns as the internal system wide identifier. All user-defined names are changed into an internal name that is unique across user and database-names. A user need only be concerned with the names assigned by him/her. This naming scheme is consistent With the way Sybase expands user-defined object names. If a user assigns an object ***objectName***, this name will be changed into the following system-wide internal name:

DatabaseName.userName.objectName

create trigger *[owner.] trigger_name*
on *[owner.] table_name*
for operation
[event *event_name [coupling_mode] [parameter_context] [priority]*]
as *SQL_statements*

operation := insert | delete | update
parameter_context := RECENT | CHRONICLE | CONTINUOUS
| CUMULATIVE
coupling_mode := IMMEDIATE | DEFERED | DETACHED
priority := positive integer

Figure 9. Syntax of Primitive Event Definition

create trigger *[owner.] trigger_name*
event *event_name [coupling_ mode] [parameter_context] [priority]*
as *SQL_statements*

operation := insert | delete | update
parameter_context := RECENT | CHRONICLE | CONTINUOUS
| CUMULATIVE
coupling_mode := IMMEDIATE | DEFERED | DETACHED
priority := positive integer

Figure 10.Syntax of Defining a Trigger on Existing Event

5.2 Code Generation for a Primitive Event (Example 1)

Suppose the command is:

create trigger t_addStk on stock for insert
event addStk
as print " trigger t_addStk on primitive event addStk occurs"
select * from stock

If there is no syntax error, all object names are replaced by the internal system names, as shown below:

create trigger *sentineldb.sharma.t_addStk* on stock for insert
event *sentineldb.sharma.addStk*

```
as
    print " trigger t_addStk on primitive event addStk occurs"
        select * from stock
```

SQL code generated for a primitive event includes the creation of two tables, if they do not already exist, for processing the parameter context. One table is created for storing the deleted tuples, whose internal system name is:

DatabaseName.userName.tablename_deleted

The other table is for storing the inserted tuples, whose internal system name is:

DatabaseName.userName.tablename_inserted

These two tables are created using the name of the table on which the primitive event is created. The schema is almost the same as the table with an additional attribute **vNo** (for recording the unique event occurrence value). The value of this attribute will be used for composing parameters for the parameter context specified. For the example shown above, the following tables are created:

sentineldb.sharma.stock_inserted and ***sentineldb.sharma.stock_deleted.***

The code is generated for the example 1 is shown Figure 11. This code will be sent to the Gateway Open Server, which will pass it on to the SQL Server, and the result will be returned to the client.

In addition t the SQL code generated, additional SQL statements are generated to persist the Rule in the system tables: SysEcaTrigger and SysPrimitiveEvent. These SQL statements are sent to the Persistent Manager.

Two **insert** statements are generated for example 1:

```
insert SysEcaTrigger values ("sharma", "t_addStk",
"sentineldb.sharma.t_addStk__Proc", getdate(), "addStk")
insert SysPrimitiveEvent values("sharma", "addStk", "stock", "insert",
getdate(), 0)
```

Furthermore, a primitive event is created in the LED (using the API of LED). For the above example, the event name is **sentineldb.sharma.addStk.**

```
PRIMITIVE *eventPoint=new PRIMITIVE(sentineldb.sharma.addStk
"Sybase_Event","begin", sentineldb.sharma.addStk);
```

5.3 Composite Event trigger implementation (Example 2)

The objective of the ECA Agent is to expand the functionality of the SQL Server as well as provide user transparency. To provide a composite event mechanism for SQL Server, the syntax of the trigger definition in SQL is minimally expanded.

Figure 12 shows the syntax of a composite event definition where the event expression corresponds to any event expression using the Snoop syntax shown in Section 2.2.

The event specification language, SNOOP, is used by Sentinel to specify composite events in the ECA Agent. The keyword "**create trigger**" is the same as that of Sybase SQL Server. The action function is written in SQL and is invoked within SQL Server. The result is returned to the client.

The command from the client goes through the Gateway Open Server and is then forwarded to Language Filter. If the command is a composite event definition command, it is sent to Composite Event Parser. The Composite Event Parser parses the command, creates the composite event in the LED and generates the appropriate SQL code.

```
/* create two tables. */
select * into sentineldb.sharma.stock_inserted from stock where1=2
alter table sentineldb.sharma.stock_inserted add vNo int null

/*create stored procedure*/
1.1.1.1  create procedure sentineldb.sharma.t_addStk__Proc as
print "* trigger_addStk on primitive event addStk occurs"
select * from stock

/* create trigger*/
create trigger sentineldb.sharma.t_addStk
on stock
1.1.1.2  for insert
as
 insert sentineldb.sharma.stock_inserted
        select * from inserted,Version
 select syb_sendmsg("128.227.205.215", 10006, " sharmastockinsert
begin sentineldb.sharma.addStk") /* Notification */

/* Get and change occurence Number */
update SysPrimitiveEvent set vNo=vNo+1 where eventName
= "sentineldb.sharma.addStk "
delete Version insert Version select vNo from SysPrimitiveEvent where eventName="
sentineldb.sharma.addStk"

/* action function */
execute sentineldb.sharma.t_addStk__Proc
```

Figure 11. Code Generation for the Primitive Trigger

create trigger *[owner.] trigger_name*
event *event_name [= Snoop_Event_exp]*
[coupling_ mode] [parameter_context] [priority]
as *SQL_statements*

Figure 12. Syntax of Composite Event Definition

Below, we illustrate, with an example, the creation of a composite event:

```
create trigger t_and
event addDel = delStk ^ addStk
RECENT
as
   print " trigger t_and on composite event addDel = delStk ^ addStk"
   select symbol, price from stock.inserted
```

Syntax checking: The command is scanned to check syntax. If there is no syntax error, all object names are replaced with internal system names. Otherwise, an error message is returned to the Gateway Open Server.

In example 2, since there is no syntax error, the command is changed to:

```
create trigger sentineldb.sharma.t_and
event sentineldb.sharma.addDel = sentineldb.sharma.delStk ^
                                 sentineldb.sharma.addStk
RECENT
as
   print " trigger t_and on composite event addDel = delStk ^
         addStk"
   select symbol, price from sentineldb.sharma.stock.inserted_tmp
```

Name checking: New object names should not be duplicates and all associated objects should exist. If these conditions are not satisfied, an error message is returned to the Gateway Open Server. Otherwise, the event specification string is sent to the Snoop Parser.
For example 2, new object name ***sentineldb.sharma.t_and*** and ***sentineldb.sharma.addDel*** are not duplicates in the system and event ***sentineldb.sharma.delStk*** and ***sentineldb.sharma.addStk*** are currently defined, so string "***sentineldb.sharma.addDel = sentineldb.sharma.delStk ^ sentineldb.sharma.addStk***" is sent to the Snoop Parser.

Snoop Parser: Snoop Parser parses the event specification string. If there is no error in the string, it creates the composite event in the LED and generates a node in the **eventContext** list to process context. In example 2, event ***sentineldb.sharma.addDel*** is created in the LED using the constructor of the composite event operator:

AND *sentineldb.sharma.addDel = new AND (sentineldb.sharma.delStk, sentineldb.sharma.addStk)

Code generation: The following code is generated: A rule is created in LED. Once the rule is triggered, the action function is called, since a function can only be called in C++ in LED, function **SybaseAction** is defined as an interface of all SQL actions. In this function, the SQL action is sent to the Gateway Open Server to run the action. All information associated with the SQL action function is packed into a structure NotiStr (Figure 13) so that when **SybaseAction** is invoked, it can call the corresponding SQL function.

```
/* Notify Parameter Structure*/
struct NotiStr
{
  char store_proc[MAX_PARA_LENGTH];//stored procedure
name
  char eventName[EVENT_NAME_LENGTH];// event name
 char context[CONTEXT_LEN];//context
  SRV_PROC     *spp; // Thread control structure for connection
}
```

Figure 13. Structure of NotiStr

SQL code is generated to create a stored procedure in the SQL Server as an Action function. The code is sent to SQL Server through the Gateway Open Server.

SQL code is generated to make the event and the rule object persistent. In example 2, rule **sentineldb.sharma.t_and** is generated in the LED:

RULE * sentineldb.sharma.t_and = new RULE(sentineldb.sharma.t_and, sentineldb.sharma.addDel, condition, SybaseAction,(void *) ActionPara, RECENT);
ActionPara is a pointer of structure NotiStr:
ActionPara->store_proc = "sentineldb.sharma.t_and__Proc";
ActionPara->eventName= "sentineldb.sharma.addDel";
ActionPara->context="RECENT";
ActionPara->spp=spp;
// spp is a pointer of Thread Conrol Structure for client in Open Server

The code in Figure 14 is generated for action in the SQL Server and is sent to the SQL Server through the Gateway Open Server. Also the following SQL code is generated for inserting tuples into the tables **SysCompositEvent** and **SysEcaTrigger:**
insert SysPrimitiveEvent values ("sharma", "addDel", "delStk ^ addStk", getdate(), "IMMEDIATE", "RECENT", 0)

insert SysEcaTrigger values ("sharma", "t_and", "sentineldb.sharma.t_and__Proc", getdate(), "addDel")

The above code is sent to the Persistent Manager.

```
create procedure sentineldb.sharma.t_and__Proc
as

/* context processing */
        delete sentineldb.sharma.stock_inserted_tmp
        insert sentineldb.sharma.stock_inserted_tmp
              select *
                    from sentineldb.sharma.stock_inserted, sysContext
                   where   sysContext.context="RECENT"
                              and
                              sysContext.tableName=" sentineldb.sharma.stock"
                              and
                           sentineldb.sharma.stock_inserted.vNo=
                              sysContext.vNo

/*action function*/
  print "trigger t_and on composite event addDel = addStk ^ delStk"
        select symbol, price from sentineldb.sharma.stock.inserted_tmp
```

Figure 14. Stored procedure for Example 2.

5.4 Event Notifier

The Event Notifer is a Light Weight Thread, which runs in the ECA Agent. It is generated when the ECA Agent starts. Once it is created, it will wait for notifications from SQL Server. If Event Notifier receives a notification from the SQL Server, it notifies the LED that an event has occurred.

There are two parts in the Event Notifier. The first is the Notification Listener, which catches any notification from the SQL Server. The other is the Notifier, which sends notifications to LED. Figure 15 shows the workflow of the Event Notifier.

Primitive Event Parser adds a built-in function call in the trigger definition. The built-in function uses the UDP protocol to send the message to the destination. When a primitive event occurs, a trigger is invoked in the SQL Server. The UDP socket is created and the message is sent to the Notification Listener in the Event Notifier. As soon as the Notification Listener receives notification, it calls the Event Notifer. At this point the Event Notifier unpacks the message and sends the notification to LED. After LED detects an event, it invokes an action interface

"**SybaseAction**." "**SybaseAction**" calls an action function (a stored procedure) stored in the SQL Server through the Gateway Open Server.

5.5 Action Handler

Sybase Action actually is a function in ECA Agent. From the LED's point of view, it is an C++ action function. From the point view of SQL Server, it is an interface for an action in the SQL Server. Since many events may occur at the same time, and each action function should run independently, new thread is generated for each call to **SybaseAction**. Each thread calls an action function within the SQL Server (stored procedure) through the Gateway Open Server. The final results are returned to clients through the Gateway Open Server. Figure 16 shows how Sybase Action works.

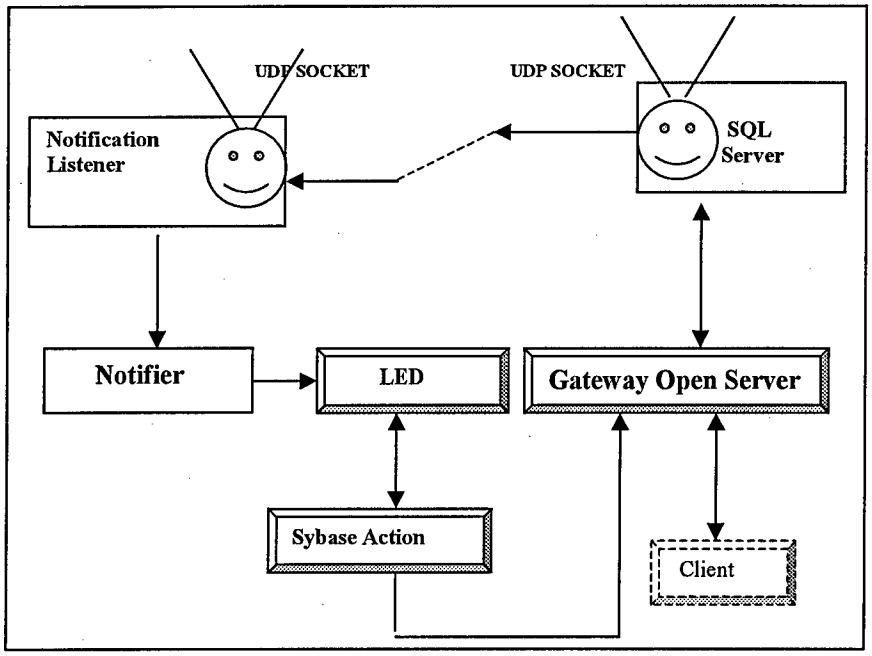


Figure 15. Workflow of Event Notifier

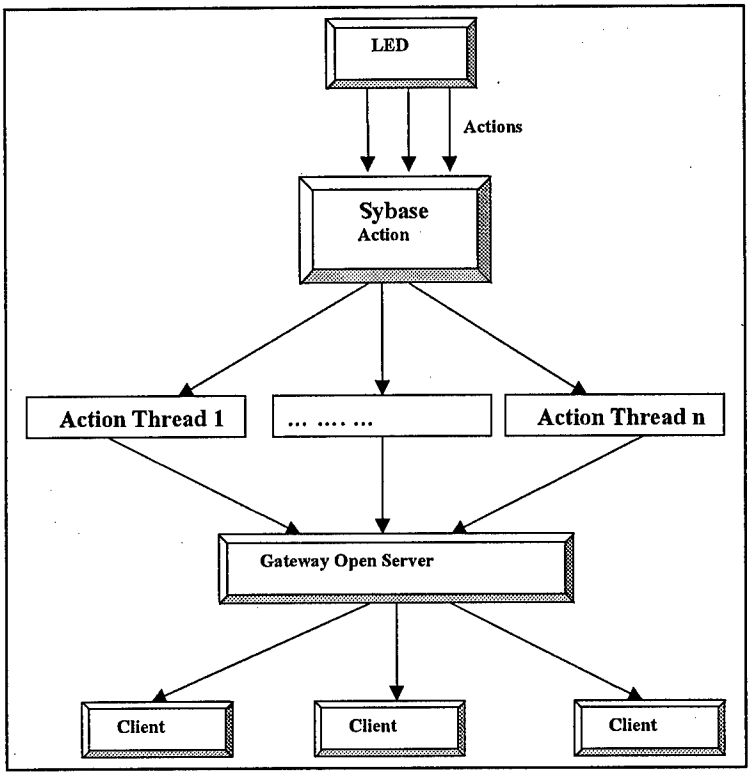


Figure 16.Action Handler

5.6 Parameter Context

The parameter contexts in Sentinel are introduced as *recent, continuous, and cumulative* and *chronicle* [CHA94]. The ECA Agent supports all parameter contexts. To get the context of a table, a user should use the following syntax: **TableName.inserted** or **TableName.deleted**

System table sysContext is created to store the occurrence of the tables defined on certain events. The structure of table sysContext is shown in Figure 17

Column_name	Type	Length	Nulls
tableName	varchar	50	not null
context	varchar	12	not null
vNo	int	4	not null

Figure 17. Structure of Table sysContext

Tuples are inserted into sysContext when the action is invoked. Each tuple is associated with a table. Since for one table there may be many tuples (corresponding to the same event

occurrence), in a composite event, there are many tuples for the same table for the same parameter context. A list is generated after event detection. The list is derived from LED, changed into the list that records table occurrences then generate SQL code to insert tuples into sysContext table. The old tuples whose **tableName** and **context** is the same as that of new tuples should be deleted before inserting new tuples.

All occurrences of events and their associated table names for each context are kept in the system table sysContext. There are four steps to properly handling a parameter context irrespective of which one it is:

- When a primitive event occurs, the occurrence of associated table is put into table **dbname.username.TableName.inserted** or **dbname.username.TableName. deleted.**
- Retrieve the parameter context list from LED and generate the SQL code for inserting tuples.
- Insert tuples into **sysContext** through the Gateway Open Server.
- Join **sysContext** and **dbname.username.TableName.inserted** (or **dbname.username.TableName.deleted** to get the context.

6 Conclusion

We have demonstrated in this paper that by introducing an ECA Agent outside the SQL Server, full-fledged active functionality can be supported as a value-added capability. Nearly the full range of active functionality can be supported without resorting to an integrated active database architecture. The ECA Agent makes the Sybase SQL Server more powerful and provides transparent active capability to database clients.

Although this paper describes an architecture, design, and implementation of an ECA Agent for Sybase, we believe that the approach developed in paper is general and can be used for any relational DBMS.The functionality includes:

- Transparent interface for users to create primitive and composite events.
- Persistence of user created events using the native DBMS capability.
- Trigger management (create and drop) for primitive and composite events.
- Detection of primitive and composite events.
- Invoke actions within the SQL Server.
- Support for multiple parameter contexts.
- Collecting and passing parameters to conditions and actions.

Considering the recent proliferation of commercial relational systems that support primitive trigger capability, the prospect of seamlessly integrating an active component is very attractive indeed. For many users, this may be the only practical way to use production rules today. Additionally, because ECA Agent is external to Sybase, its power is not limited to what the database can provide, but by what the architecture supports.

Our future research concerns the following issues:

- The current implementation supports immediate coupling mode. We plan on extending this with detached and deferred coupling
- We plan on supporting heterogeneous distributed active capability by using this approach t enhance native capability and use a global event detector (GED) for events and rules across application/systems.

- Optimization of components of the ECA Agent. Since the communication between ECA Agent and SQL Server is based on the socket, security and system efficiency will be affected. We plan on decreasing communication times to increase system efficiency and make the system more secure.
- Support active database semantics in the other RDBMS such as Informix, Oracle, and DB2 by using the idea of an ECA Agent.

7 Acknowledgments

This work was supported in part by the Office of Naval Research and the SPAWAR System Center–San Diego, by the Rome Laboratory, DARPA, and the NSF grant IRI-9528390.

8 References

[ACT96] ACT-NET Consortium (1996). The Active Database Management System Manifesto: A Rulebase of ADBMS Features. ACM SIGMOD Record, 25(3):40--49.

[BER91] Berndtsson, M. ACOOD: an Approach to an Active Object Oriented DBMS. Master's thesis, University of Skovde, September 1991.

[BER92] Berndtsson, M. and Lings, B. (1992). On Developing Reactive Object_Oriented Databases. *IEEE Quarterly Bulletin on Data Science, Special Issue on Active Databases*, 15(1-4):31--34.

[BER94] Berndtsson, M. (1994). Reactive Object-Oriented Databases and CIM. In *Proceedings of the 5th International Conference on Database and Expert Systems Applications*, volume 856 of *Lecture Notes in Computer Science*, pages 769--778. Springer

[CER96] Ceri, S., Fraternali, P., Paraboschi, S., and Branca, L. (1996). Active Rule Management in Chimera. In ActiveDatabase Systems - Triggers and Rules For Advanced Database Processing, chapter 6, pages 151--176. Morgan Kaufman.

[CHA89] Chakravarthy, S. (1989). Rule Management and Evaluation: An Active DBMS Perspective. ACM SIGMOD Record, 18(3):20--28.

[CHA90] Chakravarthy, S. and Nesson, S. (1990). Making an Object-Oriented DBMS Active: Design, Implementation and Evaluation of a Prototype. In Bancilhon, F., Thanos, C., and Tsichritzis, D., editors, Advances in DatabaseTechnology - EDBT'90. International Conference on Extending Database Technology, volume 416 of Lecture Notes in Computer Science, pages 393--406. Springer.

[CHA94] Chakravarthy, S., Anwar, E., Maugis, L., and Mishra, D. (1994). Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules. Information and Software Technology, 36(9):559--568.

[CHA95] Chakravarthy, S., Krishnaprasad, V., Tamizuddin, Z., and Badani, R. (1995a). ECA Rule Integration into an OODBMS: Architecture and Implementation. In Proceedings of the 11th International Conference on Data Science, pages 341--348.

[CHK94] Chakravarty, S., Krishnaprasad, V., Anwar, E., and Kim, S. K. (1994). Composite Events for Active Databases: Semantics Contexts and Detection. In Proceedings of the 20th International Conference on Very Large Data Bases, pages 606--617.

[CHM94] Chakravarthy, S. and Mishra, D. (1994). Snoop: An Expressive Event Specification Language for Active Databases. Data and Knowledge Science, 14(1):1--26.

[DAY88] Dayal, U., Blaustein, B., A. Buchmann, S. C., and et al. (1988a). The HiPAC project: Combining active databases and timing constraints. ACM SIGMOD Record, 17(1):51--70.

[DAY94] U.Dayal, E. N. Hanson, and J. Wisdom. Active Databasebase Systems. In Modern Database Systems: The Object Model, Interoperability, and BeyondAddison-Wesley, Reading, Massachusetts, 1994

[GEH92] Gehani, N., Jagadish, H. V., and Smueli, O. (1992b). Event specification in an active object-oriented database. In Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, pages 81--90.

[HAN89] Hanson, E. N. (1989). An Initial Report on the Design of Ariel: A DBMS With an Integrated Production Rule System. ACM SIGMOD Record, 18(3):12--19.

[HAN92] Hanson, E. N. (1992). Rule Condition Testing and Action Execution in Ariel. In Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, pages 49--58.

[HAN93] Hanson, E. N. and Widom, J. (1993). An Overview of Production Rules in Database Systems. The Knowledge Science Review, 8(2):121--143.

[LEE96] Lee, H. Support for Temporal Events in Sentinel: Design, Implementation, and Preprocessing. Master's thesis, University of Florida, Gainesville, 1996.

[LIA97] Liao, H. Global Events in Sentinel: Design and Implementation of a Global Event Detector. Master's thesis, University of Florida, Gainesville, 1997.

[SHY91] Shyy, Yuh-Ming and Su, Stanley Y. W., "K: A High-Level Knowledge Base Programming Language for Advanced Database Applications," SIGMOD '91, ACM, Denver, CO., May 29-31, 1991, pp. 338-347.

[STO87] Stonebraker, M., Hanson, E., and Hong, C. H. (1987). The Design of the Postgres Rule System. In Proceedings of the 3rd International IEEE Conference on Data Science, pages 365--374.

[STO88] Stonebraker, M., Hanson, M., and Potamianos, S. (1988). The POSTGRES rule manager. IEEE Transactions on Software Science, 14(7):897--907.

[STO92] Stonebraker, M. (1992). The Integration of Rule Systems and Database Systems. IEEE Transactions on Knowledge and Data Science, 4(5):415--423.

[SU93] Su, S. Y. W. and Chen, H-H. M. (1993). Temporal Rule Specification and Management in Object-Oriented Knowledge Bases. In Proceedings of the 1st International Workshop on Rules in Database Systems, Workshops in Computing, pages 73--91. Springer Verlag.

[SU91] Su, Stanley Y. W. and Park, Jong H., "An Integrated System for Knowledge Sharing among Heterogeneous Knowledge Derivation Systems," International Journal of Applied Intelligence, Vol. 1, 1991, pp. 223-245.

[SYB96] Sybase. Sybase SQL Reference Manual: Volume 1. Sybase, Inc., 1996.

[VAN96] Vance, D. Supporting Active Database Semantics in Sybase. Master's thesis, University of Florida, Gainesville, 1996.

[WID94] Widom, J. and Chakravarthy, S., editors (1994).Proceedings of the 4th International Workshop on Research Issues in Data Science - Active Database Systems. IEEE-CS. ISBN 0-8186-5360-4.

[WID96] Widom, J. (1996). The Starburst Active Database Rule System. IEEE Transactions on Knowledge and Data Science, 8(4): 583--595.

*U.S. GOVERNMENT PRINTING OFFICE: 1999-610-130-81089

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.