CoSent: A Cooperative Sentinel for Database Systems*

Wesley W. Chu, Xiaohong Yang, Wenlei Mao

Phone: (310) 825-2047

Fax: (310) 825-7578

Email: wwc@cs.ucla.edu

Computer Science Department

University of California, Los Angeles

October 29, 1998

Abstract

Active database systems have received increasing interest from both research and industrial communities. However, trigger conditions in the active rules are often difficult to specify, especially as the events and/or conditions complexities increase. To remedy this problem, we propose an active database system that supports rules with conceptual terms, approximate operators, and complex events. The conceptual terms and approximate operators are user and context sensitive. By introducing these high-level constructs, we not only simplify the rule specification process, but also increase the rule expressiveness. Knowledge-based relaxation techniques are used for rule specification and relaxation. High-level concepts and approximate operators used in rules are first relaxed into low-level active rules by using a tree-type knowledge structure called Type Abstraction Hierarchy which can be generated automatically from the database using clustering algorithms. The low-level rules are decomposed into a set of database triggers, which are then submitted into commercial active relational databases for simple trigger processing. Thus, our proposed high-level active database system supports

*This project is partly supported by DARPA contract F30602-94-C-0207

complex event detection without modification of the underlying database systems. This is in contrast to some of the research work on complex event detection that requires modification of the underlying database systems. A new, more flexible event condition evaluation scheme is proposed for processing complex conditioned events. Such an evaluation scheme also fits naturally into the distributed environment for detecting complex conditioned events that occur at different sites to cause a joint action. The proposed active database system with high-level rule processing and complex event detection has been implemented at UCLA. The system operates on top of commercial relational databases that demonstrates the feasibility of high-level rule processing and complex events detection.

1 Introduction

Active database systems (sentinels) [WC95] enhance traditional database functionalities with rule processing and triggering capabilities. Event-Condition-Action (ECA) rules are used to monitor and process events with conditions. When an event occurs, and if the conditions specified in the rule hold, then take the appropriate actions.

Traditional sentinels process rules with exact conditions. However, real world event conditions are often inexact, uncertain, and represented by high-level concepts. Further, these conditions are user and context sensitive. For example, a pilot would like to be notified if a *bad weather* forecast is reported in the region of his interest. Here "bad weather" is a high-level concept whose semantics depends on the user type (*e.g.*, "pilot", "sailor"), and application context (*e.g.*, "mission type"). To handle such inexact and uncertain conditions, we introduce knowledge-based relaxation (cooperative query answering) techniques [CMB93, CYC+96] to the sentinel system so that such system can support ECA rules with high-level concepts and cooperative operators.

Commercial database systems, such as Oracle and SyBase, provide only simple ECA rule processing capabilities. Each rule can only monitor one database event with certain conditions. Further, a single table can only be monitored by at most three triggers (one for each of the `INSERT`, `DELETE`, and `UPDATE` event) which hinder the usability and expressive power of the triggering systems. Triggering systems with complex event detection mechanisms have

been researched [AG89, Cha97, GD93]. However, such systems require modification of the underlying database systems, which is not suitable to the commercial database systems. The system described in [Cha97] uses a "Complex Event"–"Global Condition"–"Action" scheme to represent rules with complex events. Under such a scheme, complex events with different condition evaluation times are difficult to handle. Therefore, we extend the conventional Event-Condition-Action (ECA) scheme into "Conditioned Event"-Action (EcA) scheme that allows succinct expression of complex events with different condition evaluation timing requirements. Further, our system can enhance commercial database triggering systems with complex event detection and provide multiple triggers on a single table.

A large number of active rules exists in a large database triggering system. An event can occur multiple times during the lifetime of the system. Only a selective few may be of interest to a user. We provide a *valid interval* construct for active rules to allow the user to specify the event condition which reduces the amount of event detection and rule processing, and thus improve system performance.

Current commercial database systems provide gateways to access multiple database systems. It is possible to install triggers on different database systems through such gateways. However, to provide more flexible event detection and composition, as well as system performance, distributed event detection should be used. Therefore we have extended our triggering system to include distributed event detection which can detect events that occur at different sites to cause a joint action when the conditions of these events are satisfied.

In this paper, we first compare the currently available composite event detection techniques in section 2. Then in section 3, we discuss our extension to the traditional ECA scheme, which incorporates high-level concepts and cooperative operators with a flexible condition evaluation timing scheme. Then we present the centralized CoSent architecture in section 4. The distributed sentinels are discussed in section 5. Finally, we describe our experience and test examples of the system.

2 Related Works

The HiPAC project [Cea89] pioneered the active database systems research in the mid-1980s. Research works has been done in the areas of rule language design, rule execution semantics, rule debugging, and system architecture. Many active rule specification languages were designed. SQL3 includes simple trigger processing capabilities [IA94] The complex event processing research results in three types of systems, event-tree in Sentinel [Cha97], petri-net in SAMOS [GD93]. and finite state automata in Ode [AG89]. The semantics of complex events in event-tree based systems are studied in [CAK94]. Various research prototypes require either modification of underlying relational database systems or the use of object object-oriented systems as the underlying database.

3 Rules with Conceptual Terms and Approximate Operators

3.1 Cooperative Features

Traditional active rules require precise specification of trigger conditions and actions and monitor database attributes as events. To specify the active rules, rule designers need to have detailed knowledge (schema as well as data) about the underlying databases. However, such detailed knowledge is often difficult and time consuming to obtain. Furthermore, the rule designers and users' apprehension of a trigger condition may be inexact and are user and context sensitive. To remedy these shortcomings, we propose to generalize the ECA rules to support conceptual terms and approximate operators to improve the rule expressiveness. We use knowledge-based relaxation techniques to transform the high-level rules to low-level rules for processing in commercial database triggering systems.

3.1.1 Query Relaxation Techniques

Knowledge-based query relaxation was used in cooperative systems such as CoBase [CMB93, CYC+96]. Relaxation increases the search scope of the query condition and is able to provide

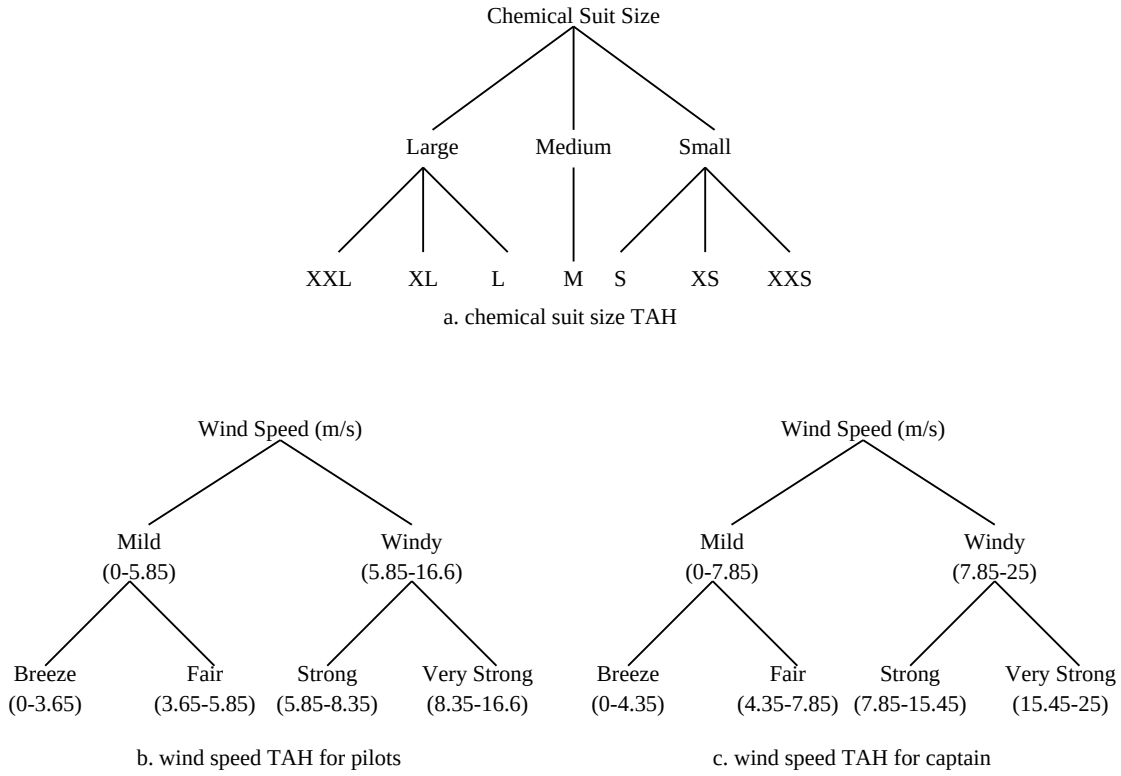

Figure 1: Examples of Type Abstraction Hierarchies

approximate matching when no exact match can be found. Applying relaxation techniques on trigger conditions allows the users to specify rules with approximate and cooperative terms, thus eases the trigger condition specifications, and increases the expressiveness of the rules.

We use a novel multi-level tree structure for knowledge representation called the Type Abstraction Hierarchy (TAH) [CYC+96]. High-level nodes in the TAH represent more general information than that of the lower nodes. *Conceptual terms* can be defined on the TAH nodes. As a result, queries with conceptual conditions can be specified and processed. For example, in the query, "find chemical suits with size large," the conceptual term *large* can be transformed into XXL, XL, or L as shown in Figure 1a. The query condition can be generalized (scope enlarged) by moving up and specialized (scope reduced) by moving down the TAH. The relaxation process is repeated until satisfactory answers are returned.

In addition to providing *implicit* modifications via TAHs, relaxation can be specified *explicitly* through the use of *cooperative operators* such as approximate, near-to, similar-to,

etc. The *approximate* operator relaxes the specified values within an approximate range. For example, "approximate 6:00am" is relaxed to (5:00am, 7:00am). The *near-to* operator can be used for specifying geographical nearness. The *similar-to* operator can be used to find objects similar to the given target object based on a set of attributes. Weights can also be assigned to the set of attributes in accordance to their relative importance. The returned answer sets are ranked based on a pre-specified measure that evaluates the nearness of the answers from the target object.

Clustering algorithms have been developed to generate TAHs automatically from data sources based on a set of attributes selected by the user [CC94, MC93, CCHY96]. Therefore, the TAHs are customized based on the user and context. The generated TAHs can be edited (*e.g.*, addition and deletion of TAH nodes, naming the TAH nodes with conceptual terms, *etc*) by the domain experts.

Relaxation control operators such as *relaxation-order*, *unacceptable-list*, *preference-list*, *relaxation-level*, and *not-relaxable* are provided to control the relaxation process. The user can specify the relaxation control in the query. A default relaxation control can also be obtained from user types.

3.1.2 High-level Rule Condition Specification

In this section, we shall discuss the application of query relaxation techniques to high-level rule specification.

Rules with High-level Concepts

In a typical ECA rule, the trigger condition can be specified by presice values. For example, "the wave height is 3 meters and the wind speed is 16 meters per second." However, the trigger conditions are often "fuzzy" and difficult to specify. The user usually has only an approximate estimate of the situation. Further, specification usually varies and is user and context sensitive. Since the database content can be represented by TAHs, the user can customize the TAH by selecting the attributes used in generating the TAHs. The domain experts can then label the TAH nodes with conceptual terms (see Figure 1) and use them in the rule sepecification. For example, a user wants to be notified "if the weather at Bizerte is *very bad*." "Very bad" is a high-level concept that is determined by the user type. For

example, for an airplane pilot, "the weather is very bad" translates into "the wind speed is *very strong*, and the visibility is *very poor*", while for a ship captain, "the weather is very bad" translates into "the wind speed is *very strong* and the wave height is *very high*." Notice that "very strong", "very high" and "very poor" are conceptual terms and can be represented by the corresponding TAHs. The conceptual terms are user and context sensitive. For example, "very strong" wind speed for a pilot and captain has different interpretations. Based on the TAHs for the pilots, the above high-level rule condition is translated into "the wind speed is between 8.35 and 16.6 meters per second, and the visibility is less than 10 meters." For the captain, the above high-level rule condition is translated into "the wind speed is in the range 15.45 to 25 meters per second, and the wave height is greater than 5 meters." as shown in Figure 1b and 1c.

Rules with Cooperate Operators

We introduce cooperative operators such as *approximate*, *near-to*, and *similar-to* in the rules to increase expressiveness. For example, if a ship is scheduled to pass through near to Bizerte approximately on 9/1/1998, the captain wants to be informed if the weather condition is *bad*. Here "near to" and "approximate" are both cooperative operators. Introducing these operators into the triggering system greatly simplifies the rule specification. The user need not know the domain knowledge to specify the ranges of these operators. The range values of "near to" and "approximate" can be obtained from the corresponding TAHs which can be customized based on user and context.

Relaxation Controls in High-level Rules

To specify a concept or cooperative operators in high-level rules, TAHs are used as knowledge representation to interpret the terms. Default TAHs for a user type and context can be used if no specific TAH is specified. The user can also supply specific TAHs for representing the conceptual and approximate terms. Further, the relaxation process of a conceptual term can be controlled and specified by the user through relaxation control operators during the relaxation process (*e.g.*, `relaxation-level`).

3.1.3 Rule Action Specification

Cooperative features of the relaxation techniques can also be used in rule action specification. For example, consider the following rule, "if not enough large-sized chemical suits at the warehouse in city X, then find 10,000 units of large-sized chemical suits from depots near to city X." In the action part of the above rule, "10,000" and "large-sized" can be *implicitly* relaxed if there are less than 10,000 large-sized chemical suits available. By introducing cooperative features into active rule action specification, the rule designers can rely on CoBase [CMB93] to relax the query condition if the exact condition is not satisfied and relax the quantity to "approximately 10,000", and "large-size" to "medium" or "extra large".

3.1.4 An Example

Consider an Air Force database containing information of the aircraft departure rates and aircraft maintenance problems. Daily departure numbers of different types of aircrafts are inserted into the `ac_departure` table. Other attributes in the `ac_departure` table include the date the tuple is inserted, the type of aircraft. Similarly, `ac_problem` summarizes different maintenance problems occurred to each type of aircrafts daily. If the departure of a specific type of aircraft, *e.g.*, C-5, within the last seven days was *significantly low* and the occurrence of the fuel filter problem on the same type of aircraft is *extremely high* during the same period, the commander should be notified of this situation. The following is the corresponding high-level rule representation.

R: If departure rate of C-5 within the past 7 days is *significantly low*

and

if fuel filter problem rate of C-5 is *extremely high* within the past 7 days

then report the departure rate, problem type and date of occurrences to the commander.

Note that in the above example, the rule has a *conditioned event* with a *valid-interval specification* and an *action*. A conditioned event has an *event* and a *condition*. A condition can have all the cooperative constructs introduced in the following section. The transition

tables/tuples, such as `inserted` or `deleted`, can be used in the condition specification to refer to the corresponding tuples. The interpretation of these transition tables/tuples varies according to the granularity of the rule specification. An action is a user defined procedure which may contain cooperative operators. The event detection will only detect the events occur within the valid interval.

3.2 Constructs in Cooperative Rule

3.2.1 Event Types

An event can be either a *simple event* or a *complex event*. A simple event is a *database update event* or a *time event*. A database update event is either an database insert, delete, or update. A time event can be absolute time, *e.g.*, 3/1/1998 12:00pm, or relative time, *e.g.*, 2 days after an aircraft fuel filter problem occurs, or periodic time event, *e.g.*, 3:00pm everyday. A complex event is a regular expression of conditioned events. Two types of operators can be used in the complex event expression: unary and binary operators. The binary operators contain `AND`, `OR`, `IMMEDIATE-FOLLOWED-BY`, and `FOLLOWED-BY`. The unary operators include `*`, `+`, and `?`, representing zero or more, one or more, and zero or one occurrences of immediately sequence of the same conditioned event, respectively.

An event may have multiple occurrences. To specify which set of occurrences for an event is of interest, an *occurrence modifier* can be specified. For example, if `E` is an event, `E{1, 3-4, 6-}` specifies that only the first, third, fourth, and all the occurrences of `E` above the sixth are of interest.

3.2.2 Event-Condition Evaluation Scheme

Traditionally, an active rule is specified as an event-condition-action. The event can be either a simple event or a complex event *without* condition specifications. The condition is evaluated only after the event occurs. If the condition is satisfied, the action is taken [CAK94]. However, such a condition evaluation timing scheme does not allow the user to specify the event condition to be evaluated *at the time* of the occurrence of a sub-event. For example, a user wants to be informed if a certain stock increases by ten percent in value, followed by

a "buy" recommendation for the stock. The condition that the stock increases ten percent needs to be evaluated at the time of the report of the stock rather than after the entire event (stock increases by ten percent in value, followed by a "buy" recommendation) has already occurred. An alternative way of implementing this rule is to install a conditioned event into the database and let the database evaluate the condition at the moment the event occurs. However, due to the limitation of current commercial database systems on the number of triggers on a single table, this approach is not implementable on top of commercial databases. The traditional event condition evaluation scheme can be achieved by using a global condition in our rule specification. To distinguish it from the traditional ECA scheme, we label our "conditioned event"-"action" scheme as EcA.

3.2.3 Event Definition and Parameterized Rules

Within a domain, many event specifications are either identical or different by only a few parameters. To increase the reusability of the event components, we provide *event definition* constructs and *parameterized rules*.

If a sub-event is used in different event definitions or in multiple rules, a rule designer can first define a named-event and then refer to the name of the event wherever the named-event is needed as shown in the following example,

```
DEFINE EVENT Esub1 AS ...
DEFINE EVENT Esub2 AS ...
DEFINE EVENT E1 AS Esub1 FOLLOWED-BY Esub2 ...
DEFINE EVENT E2 As Esub1 AND Esub2 ...
```

sub-events `Esub1` and `Esub2` are used in the definitions of both events `E1` and `E2`.

Parameterized rules can be used to specify a set of rules with the same structure. Each parameter in the rule can be substituted with different values to generate a set of different rules. For example, we can specify a set of rules to monitor the weather condition for different locations using parameter `$loc$`:

```
DEFINE EVENT bad_weather[$loc$]
AS INSERT ON weather_report
```

```
IF (INSERTED.location = $loc$ AND
    INSERTED.weather IS "BAD")

ON bad_weather['Los Angeles'] DO ...
ON bad_weather['Boston'] DO ...
```

here "Los Angeles" and "Boston" are two value instances of parameter `$loc$`. The parameter can be used across the boundary between the conditioned event definition and action definition.

3.2.4 Valid Intervals Definition in Rules

To facilitate the rule designer to specify the event of interest so that no irrelevant information will be sent to the user, our system provides a valid intervals definition functionality. The *valid intervals* are a set of temporal intervals, where each interval has a begin point and an end point. The effective valid interval is the disjunction of the individual valid intervals. The begin point or end point can be specified by any event, for example, 3/1/98 or "troop enters a certain region event." If the begin point or end point event can occur multiple times and is not occurrence-modified to a point event, the first occurrence of such event is to be taken as the begin or end point.

For any interval, if the begin point is missing, the event valid interval starts from the system starting time; and if the end point is missing, the valid interval continues until the system is shut down.

For example, the following two event definitions

```
DEFINE EVENT troop_move_in[$trp$, $loc$] AS
INSERT ON troop_info FOR EACH ROW
IF (INSERTED.troop = $trp$ AND INSERTED.location = '$loc$')

DEFINE EVENT troop_move_out[$trp$, $loc$] AS
DELETE ON troop_info FOR EACH ROW
IF (DELETED.troop = $trp$ AND DELETED.location = '$loc$')
```

define two interval end-point events, `troop_move_in` and `troop_move_out`. The following definition defines that the `weather_bad_for_troop` event will be detected only if a "bad weather" report for region `loc` is received and also if the troop `trp` is in the region specified by `loc`.

```
DEFINE EVENT weather_bad_for_troop[$trp$, $loc$] AS
INSERT ON weather_info FOR EACH ROW
VALID FROM troop_move_in[$trp$, $loc$] TO troop_move_out[$trp$, $loc$]
IF (INSERTED.weather IS "BAD" AND
    INSERTED.location = $loc$)
```

3.2.5 Parameter Binding and Passing in Rules

Parameter binding and passing topics for active databases have been studied [WC95]. In such a scheme, an event is often specified by a set of parameters. When the event occurs, the values of the parameters are passed to the condition evaluator. However, since commercial relational databases such as Oracle and SyBase limit the number of triggers on a single table, we cannot easily adopt the parameter binding and passing scheme. To achieve the same effect as parameter binding and passing scheme, we can utilize global condition specifications to bind the parameters from different events, as shown in the following specifications for the "aircraft problems" example

```
DEFINE EVENT departure_rate[$aircraft$, $days$, $description$] AS
  INSERT ON ac_departure FOR EACH ROW
  IF (SELECT SUM(d.departure_num) FROM ac_departure d
      WHERE  INSERTED.ac_type = $aircraft$ AND
             d.ac_type = INSERTED.ac_type AND
             INSERTED.date - d.date >= 0 AND
             INSERTED.date - d.date <  $days$) = $description$
      USE-TAH ac_departure_summary_tah($days$, $aircraft$)

DEFINE EVENT aircraft_problem[aircraft, days, problem, description] AS
```

```
  INSERT ON ac_problem FOR EACH ROW
  IF (SELECT SUM(p.departure_num) FROM ac_problem p
      WHERE  INSERTED.ac_type = $aircraft$ AND
             p.ac_type = INSERTED.ac_type AND
             INSERTED.problem = $problm$ AND
             INSERTED.date - p.date >= 0 AND
             INSERTED.date - p.date <  $days$) = $description$
      USE-TAH ac_problem_summary_tah($days$, $aircraft$, $problem$)

ON departure_rate['C-5', 7, "significantly low"]
  AND
  ON aircraft_problem['C-5', 7, 'filter problem', "extremely high"]
  IF departure_rate.INSERTED.date = aircraft_problem.INSERTED.date
  DO report_to_nearby_commanders['Air Force', 'DS', 'Norfolk',
          "Within the past 7 days from :(departure_rate.INSERTED.date),
           the departure rate of C-5 is significantly low and
           the occurrence of filter problem on C-5 is extremely high."],
```

the `departure_rate.INSERTED.date = aircraft_problem.INSERTED.date` binds the occurrence date for the two events.

4 Centralized CoSent Architecture

4.1 CoSent Architecture

The centralized CoSent consists of a *Trigger Processing Agent*, a *Action Processing Agent* and a *Notification Agent*. The trigger processing agent accepts and manages cooperative rules, transforms the cooperative rules into low-level rules without cooperative terms, installs triggers and actions into the notification and action processing agent respectively. The trigger processing agent also manages and detects complex events. The action processing agent

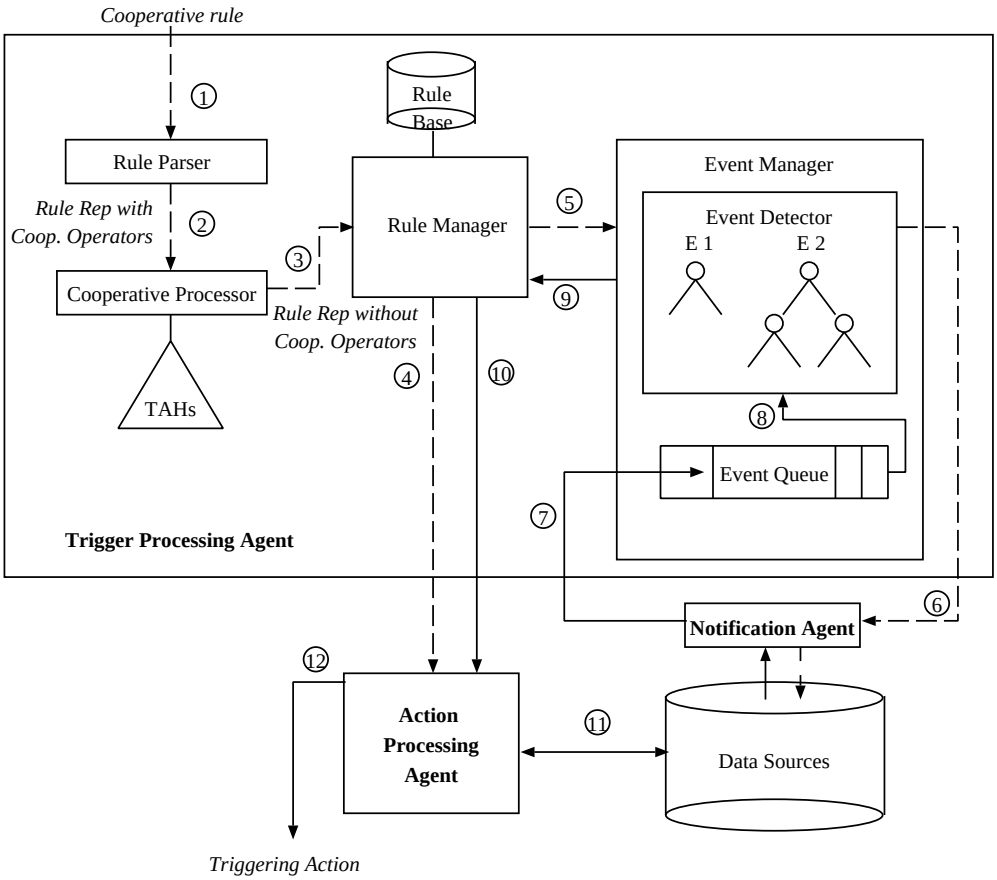

Figure 2: The Cooperative Sentinel Architecture. The dashed line depicts installation flow, and the solid line depicts execution flow.

stores the action implementations, accepts trigger actions from the trigger processing agent, and fires appropriate actions when the trigger processing agent notifies it of the triggering rules, together with event triggering information. The action processing agent may also request additional information from other agents such as the cooperative query agent and triggering processing agent. New rules can be easily added into the action processing agent online without disturbing the trigger processing agent. The notification agent monitors the underlying database changes as required by the trigger processing agent. The notification agent informs the trigger processing agent when such changes occur.

The trigger processing agent consists of a *Rule Parser*, *Cooperative Processor*, *Rule Manager* and an *Event Manager*. The rule parser takes a cooperative active rule and generates an internal representation of the rule, which contains the cooperative terms. The cooperative

processor translates the cooperative rules into a set of EcA rules with exact conditions and action specifications. The rule manager is responsible for the storage, scheduling, termination management, and installation of the rules. All cooperative active rules are stored in the rule base. The event parts are installed in the event manager and the action parts are installed into the action processing agent. The event manager consists of an event detector and an event queue. The event queue buffers the incoming simple notification event, and informs the event detector of the occurrence of simple events. Given the conditioned event from the rule manager, the event detector constructs an event tree which captures the semantics of the conditioned event. All event trees are maintained in the event detector. When simple notification events happen, the event detector processes them according to the event trees, evaluates the conditions, and informs the action processing agent of the occurrence of the events.

4.2 CoSent Information Flow

CoSent information flow consists of two phases: the installation phase and execution phase. We use the aircraft problem example described in Section 3 (Rule R) to illustrate the installation and execution flow of the system. In the installation phase, the rule manager analyzes and decomposes all the cooperative EcA rules and installs the necessary information in the event manager and the action processing agent. When a sentinel event occurs, CoSent goes through the execution phase to determine whether any rules are triggered. In the following descriptions, steps (1) through (6) represent the installation phase; steps (7) through (12) represent the execution phase.

(1) Input a high-level active rule such as rule R.

(2) The rule parser parses the high-level rule and generates an internal rule representation (RuleRep) for communication among modules. RuleRep of R has ConditionedEventRep (EcRep) and ActionRep (ARep). The ConditionedEventRep is the conditioned event part of the rule, *e.g.*, in rule R, the ConditionedEventRep (EcRep) represents the conditioned-event (with local condition evaluation)

Ec: If departure rate of C-5 within the past 7 days is *significantly low* (Ec1)

and

if fuel filter problem rate of C-5 is *extremely high* within the past 7 days (Ec2).

The ActionRep represents user defined action, *e.g.*, in rule R, the ActionRep (ARep) represents the action

A: Report the departure rate and problem occurrence, and

data of occurrences to the commanders.

(3) The cooperative processor translates cooperative terms in the conditioned event of the rule into a set of range specifications, *e.g.*, in rule R, the high-level conditioned event (Ec) is translated into a low-level conditioned event (Ec') with range specifications

Ec': If departure rate of C-5 within the past 7 days is less than one departure per day (Ec1')

and

if fuel filter problem rate of C-5 is greater than five instances

per day within the past 7 days (Ec2'),

which does not have any cooperative terms.

(4) The rule manager installs the action part of the rule, *e.g.*, A of R, into the action processing agent.

(5) The rule manager installs the low-level conditioned event, *e.g.*, Ec', of the rule into the event manager. The event manager builds an event tree for the incoming low-level conditioned event. The subtree rooted at each node represents a complex conditioned event. Based on the input from its children nodes, a node determines the occurrence of its associated complex conditioned event. A condition evaluation mechanism is also included at each node to evaluate the associated condition. For example, when the event manager receives Ec', it constructs an AND tree with leaf nodes representing Ec1' and Ec2'. The root node of the AND tree represents Ec', where E is an 'AND' event

of Ec1' and Ec2' and C' is a joint condition that guarantees the dates of occurrences of both Ec1' and Ec2' are the same. Ec1' represents a simple conditioned event with event E1

E1: insert into `ac_departure` table

and condition C1'

C1': departure rate of C-5 within the past 7 days is less than one departure per day.

Ec2' represents a simple conditioned event with event E2

E2: insert into `ac_problem` table

and condition C2'

C2': fuel filter problem rate of C-5 is greater than five instances per day within the past 7 days.

(6) The event manager installs the simple sentinel triggers, e.g. triggers E1 and E2, into the notification agent.

(7) When a sentinel event occurs, *e.g.*, E1 or E2, the notification agent saves the transition information of the event and then sends an event notification to the event queue of the event manager

(8) The queue notifies the event detector of the occurrence of a sentinel event. The event detector processes the event notification using the event trees.

(9) If a root of any event tree is reached, which implies the occurrence of the conditioned event represented by the event tree, an event notification message with appropriate parameter binding will be sent to the rule manager. For example, suppose E1 had happened and C1' was satisfied (which means Ec1' had occurred), now E2 happens, after the condition C2' is evaluated to true, the conditioned event Ec2' occurs. As a result, the event E happens at the root node of the tree. If the condition C' is also satisfied, then the conditioned event Ec' occurs.

(10) The rule manager schedules the execution order of the set of rules that are triggered by this event and send the parameter binding to the action processing agent. In our example, only a single rule R is triggered, the parameter bindings, aircraft type and problem type are sent to the action processing agent.

(11) The action processing agent acquires additional information from the other agents such as CoBase agent and trigger processing agent if necessary. In this example, no additional information is required.

(12) The action processing agent invokes the user defined procedure, *e.g.*, the procedure A with correct parameter binding.

5 Distributed CoSent Architecture

When data is distributed over multiple sites, distributed trigger processing is needed. We extend CoSent to detect events occurring at different sites to cause a joint action. We shall present the following two distributed event detection approaches in the trigger system. (1) Centralized event management with distributed event detection, (2) Distributed event management with distributed event detection. The main difference between these two approaches is that the distributed simple event detection approach uses global complex event processing, while the other approach processes complex events in a distributed manner.

5.1 Centralized Event Management with Distributed Event Detection

In the centralized CoSent architecture, the notification agent monitors database events from a single data source. In order to monitor database event from multiple data sources, we can either rely on database gateways with triggering capabilities (Figure 3(1)), or using a notification agent for each data source for data source event detection (Figure 3(2)).

Since a database gateway provides a single view for multiple data sources, distributed events coming from multiple sources can be viewed as events coming from a single view. To CoSent, this database gateway acts like a single data source and no modification is needed

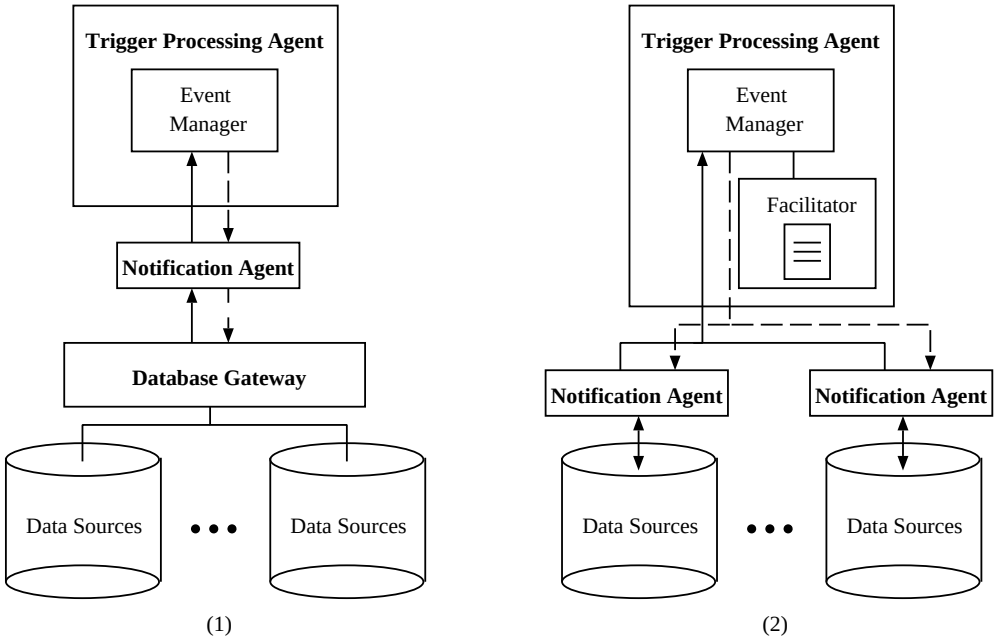

Figure 3: Centralized event management using (1) database gateway, or (2) distributed notification agents for event detection.

for the triggering processing agent. Notification agent and action processing agent need to be added for processing distributed events.

If no database gateway with triggering capabilities is available for any of the data sources, we have to develop a notification agent for each data source. The notification agent for a data source translates a simple data source event specification into the underlying data source trigger, or emulate simple trigger capability if the data source does not support simple triggers. A facilitator is added to provide the information on the trigger capabilities of the underlying data sources. When a simple event needs to be installed into a data source, the global event manager consults with the facilitator to locate and then installs the simple event into the notification agent.

Once simple events are installed into the underlying data sources through either database gateways or notification agents, the event processing via event management is the same as that in centralized CoSent.

5.2 Distributed Event Management with Distributed Event Detection

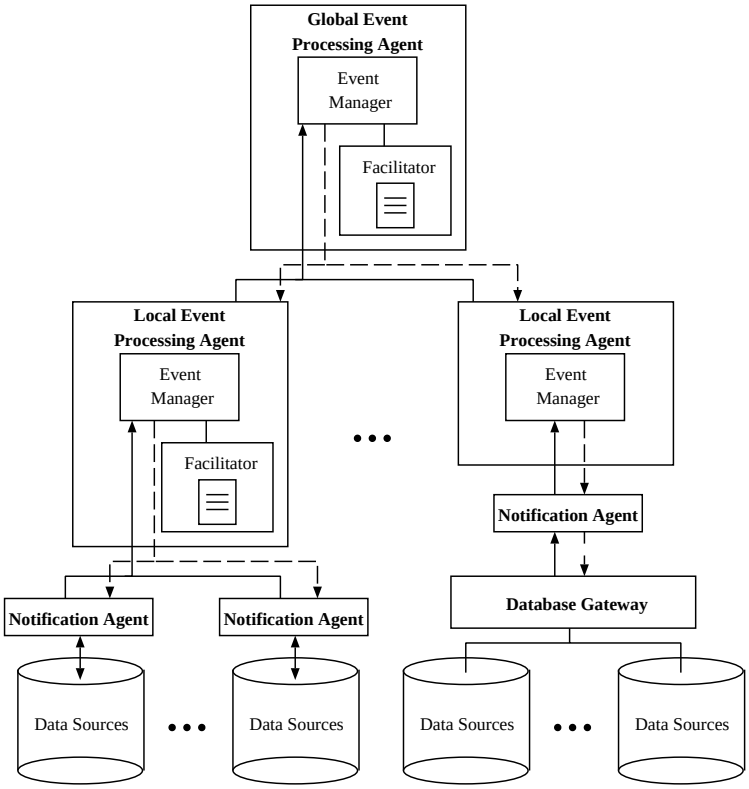

Figure 4: Distributed event management with distributed event detection. The event processing agent on the left uses distributed notification agent approach, while the agent on the right uses a database gateway.

Complex event usually consists of several sub-events, each of which comes from a set of closely related data sources. To distribute the workload from the centural trigger processing site, to improve the triggering system performamce, and to minimize the communication cost in trigger processing, the event management can be distributed and placed close to the data sources. Therefore, we propose a hierarchical distributed event processing architecture as shown in Figure 4. An event processing agent monitors sub-events from other event processing agents or data sources via the notification agent. An event processing agent consists of a event manager and an optional facilitator. The facilitator contains the information on the event processing capabailities of different event processing agents and their data sources.

To process an active rule with complex events, the global event manager in the trigger processing agent consults the facilitator, and decomposes the complex event into sub-events, so that each sub-event can be handled by local event processing agent. The sub-event is then installed into the corresponding local event processing agent.

Such distributed event management allows parallel event processing, thus improves the system response time. This is especially helpful when there are a large set of active rules with a high triggering frequency.

6 Implementation and Experience

We have implemented a prototype cooperative sentinel system at UCLA. It operates on Sun Solaris as well as on the Windows NT system. The data sources include Oracle 7.3, Oracle 8 and SyBase database systems. The trigger processing agent is implemented in C++. Orbix CORBA is used for agent communication. A Java based user interface that includes map display based on MapObject is also available for rule specification and rule activation monitoring. We have measured the performance of the trigger system on a test database (around 250 tables, with average 5,000 tuples per table). The CoSent system with 150 high-level complex rules is tested. The average delay between database update and action notification is less than 1 second for an average of 3-level rule complexities. In our system, conceptual terms and approximate operators such as near-to and similar-to can be used to specify cooperative active rules. These cooperative terms are user and context sensitive. Relaxation control operators such as use-tah and relaxation-level are also provided to further refine the relaxation process.

We have resolved the following list of problems during the implementation.

1. Since neither Oracle nor SyBase provides message passing mechanism to communicate between the application process and triggers, we had to use an ad-hoc method to implement the notification agents. The problem is more pronounced when porting our system from the Solaris to the NT system. A methodology for notification agent construction is necessary for different data sources.

2. Since our goal is for our system to operate on top of commercial database systems, we did not modify the internal triggering mechanism. As a result, the transition information is not available when a simple database event is notified to the event manager. In order to access transition information in event condition evaluation, the database trigger has to preserve the transition information in a transition table.

3. In our initial development, we did not have valid interval control. As a result, the amount of information the system has to maintain increases rather quickly. Therefore, the performance of the triggering system decreases as time goes on, even though many rules are no longer relevant. The introduction of the valid interval concept remedies this problem.

4. The action processing was an integral part of the trigger processing agent in the initial development. Whenever the user inserts a new action procedure, the entire system has to be brought down and recompiled. This is clearly not acceptable for mission critical applications. Our new design separates the action processing agent from the triggering processing agent, thus allows seamless addition of newly specified action procedures.

7 Conclusion

We have presented the incorporation of high-level concepts and cooperative operators into traditional active rule specifications. Knowledge-based relaxation techniques are used to transform the rules with high-level concepts to low-level rules to be used on top of conventional commercial database trigger systems. As a result, rule designer is able to focus more on the semantics of the active rules rather than on the user and context specific range specifications. We propose the EcA condition evaluation scheme which facilitates more flexible and more expressive active rule specification. Valid interval and occurrence modifier constructs are provided to increase the expressive power of the rule system, as well as to improve the system performance. Our system can operate on existing commercial database systems without modification of the underlying systems. Two appproaches of distributed sentinels, centralized event management and distributed event management, are also pre-

sented. We have constructed a prototype CoSent at UCLA. CoSent is operating on top of the trigger systems of commercial relational datebase systems (*e.g.*, Oracle and SyBase). We have demonstrated the feasibility of applying the relaxation technology into the active rule systems, and performed complex event detection.

References

[AG89] R. Agrawal and N. Gehani. Ode (object database and environment): The language and the data model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 36–45, Portland, Oregon, May 1989.

[CAK94] S. Chakravarthy, E. Anwar, and S-K. Kim. Composite events for active databases:semantics contexts and detection. In *The 20th International Conference on Very Large Databases*, pages 606–617, Santiago, Chile, 1994.

[CC94] Wesley W. Chu and Kuorong Chiang. Abstraction of high level concepts from numerical values in databases. In *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, 1994.

[CCHY96] W. W. Chu, K. Chiang, C. Hsu, and H. Yau. An error-based conceptual clustering method for providing approximate query answers. *CACM*, 1996.

[Cea89] S. Chakravarthy and et al. HiPAC: A research project in active, time-constrained database management (final report). Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, MA, Auguest 1989.

[Cha97] S. Chakravarthy. SENTINEL: An Object-Oriented DBMS With Event-Based Rules. In *SIGMOD '97*, Arizona, USA, 1997.

[CMB93] Wesley W. Chu, M. A. Merzbacher, and L. Berkovich. The design and implementation of CoBase. In *Proceedings of ACM SIGMOD 93*, pages 517–522, Washington D. C., USA, May 1993.

[CYC+96] Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow, and Chris Larson. CoBase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 6(11), 1996.

[GD93] Stella Gatziu and Klaus Dittrich. Event in an active object-oriented database system. In *Proceedings of the First International Workshop on Rules in Database Systems*, Edinburg, September 1993.

[IA94] ISO-ANSI. Iso-ansi working draft: Database language sql3. Technical Report X3H2/94/080 and SOU/003, ISO-ANSI, 1994.

[MC93] Matthew Merzbacher and Wesley W. Chu. Pattern-based clustering for database attribute values. In *Proceedings of AAAI Workshop on Knowledge Discovery*, Washington, DC, 1993.

[WC95] Jennifer Widom and Stefano Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, 1995.