Asynchronous Monitoring of Events for Distributed Cooperative Environments

Sharma Chakravarthy  Hui Liao†
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX 76019-0015
contact: sharma@cse.uta.edu

Abstract

Most of the work in active databases (event specification and detection) has concentrated on centralized environments. In other words, ECA rules are accepted and managed by active DBMSs within the same address space.

However, cooperative problem solving environments are inherently distributed and heterogeneous; hence there is a need for supporting ECA (Event-Condition-Action) rules in a distributed setting. This paper analyzes the design and implementation considerations of an asynchronous even t detector termed GEM (Global Event Manager) that can detect and communicate events along with their parameters in a distributed environment. This paper presents extension to Snoop event specification language to accommodate distributed event specification and develops an architecture and its implementation in the context of Sentinel active OODBMS. A distributed plan monitoring application and a flexible manufacturing application have been developed using the functionality and the system described in this paper.

1 Introduction

Much of the earlier research on active databases [1, 11, 8, 2, 6, 7] focus on the support of active capability in the context of centralized DBMS – both relational and object-oriented. In other words, ECA rules are accepted and managed by active DBMSs within the same address space (either in the application or the DBMS address space; very few DBMSs allow user defined functions to be executed in the DBMS address space. DB2 has an unfenced mode, for example.

All of the active DBMSs mentioned above do not address event specification outside of their address space. Rules cannot be specified on a composition of events that occur in several applications. That is, none of them address processing ECA rules in a distributed environment, which is a prerequisite for supporting cooperative information systems (CISs). Most of CIS applications are distributed in nature and hence require support for distributed events. For example, Telecommunication applications are inherently distributed and need coordination among component applications. A number of telephone companies are connected to the telephone network. Telephone subscribers run their telephones from one specific company, but may have accounts with other companies as well. Three major telephone companies, A T&T, GTE, and BELL, offer a special service to customers having an account with each of them and paying their bills promptly. This promptly action needs to monitor events from these three distributed companies. Active OODBMS in a centralized system cannot meet this requirement. A mechanism is needed to support processing ECA rules in a distributed environment.

This paper extends our earlier work on Sentinel by supporting asynchronous global events’ specification and detection in a distributed environment. Global event definitions are added to Snoop (an event specification language of Sentinel), and a server for asynchronous detection of global events and handling their parameters is implemented to detect events that span multiple applications. This paper develops an architecture and its implementation in the context of Sentinel active OODBMS.

1.1 Related Work

In addition to Sentinel, several research efforts have attempted to monitor the behavior of distributed systems. Most of the work only address distributed event detection without explicitly addressing: i) where to detect composite events (either locally or globally), ii) how to pass parameters, iii) architecture for asynchronous notification of events, and iv) rule processing. We believe that this paper differs from each of the related work...
in some or all of the above categories.

- Microsoft’s COM (component Object Model) [12] provides a basic event service that supports only primitive event detection. Any action that changes control, e.g., changes to data, changes to the views on data, renaming of objects, clicking the mouse can be treated as an event. There is no notion of composite events.

- CORBA [9] supports an event service to notify events to consumers via an event channel. Suppliers and consumers communicate with each other through a well-known event channel object. The push model and pull model are supported as event modification models. As in COM, no services are supported for composite events.

- In Schwiderski’s thesis [13], a general solution to monitoring the behavior of distributed systems is specified. This work is concerned with the syntax, semantics, detection, and the implementation of events in terms of physical time. Composite events is discussed but parameter passing and asynchronous notification are not addressed.

- In [10], addresses event detection over long periods of time in a distributed and partially connected systems. This thesis provides a conceptual framework and does not address the kind of implementation issues addressed in this paper.

The remainder of this paper is organized as follows. Section 2 discusses the design considerations that were considered important. Section 3 discusses the alternative architectures and their analysis. Section 4 discusses alternative strategies for processing global events. In section 5 we discuss the implementation details of the asynchronous global event manager. Finally, section 6 has conclusions.

2 Design Choices

The purpose of GEM (Global event manager) is to provide an ability to define rules across applications and to pass parameters of events (both primitive and composite) that are detected in other address spaces. Importantly, the above needs to be done without disturbing the local execution which entails that the event notification to an application (not the event detection) is truly asynchronous.

The required functionality of GEM can be summarized as follows:

1. Each application needs to reveal what events it is interested in from other applications.

2. An ability to create composite events from either primitive or composite events available from different applications.

3. Parameters of constituent events (whether they are detected locally in an application or globally by the GEM) need to be passed for use by conditions and actions of rules.

4. Finally, an application has to be notified when an event of interest occurs in another application and this has to be done asynchronously without application having to wait for an event from that application.

For item 1, we adopted a producer/consumer paradigm in which each application that is part of the distributed scenario can either be a producer, a consumer, or both. We also assumed that only consumers specify the events that are of interest to them and GEM is responsible for letting the producers know which events they need to export. This makes consistency checking easier and puts the burden on GEM (which is a server) rather than individual applications which are developed by different users. The only requirement is that each consumer be aware of the events that they intend to consume (in terms of the event name and which application generates that event).

For item 2, we support composition of global events from different applications using the full event specification language of Snoop [5, 3]. The event classification is extended to accommodate this. Four types of events can be identified in a distributed database system:

- Local Primitive Event

Local Primitive Events are events that are predefined in that application using primitive event expressions and can be detected by a mechanism embedded in the system [1]. Local Primitive Events are classified into database events and temporal events. Database events refer to database operations to manipulate data, such as insert, update, delete and retrieve. An absolute temporal event is the event specified with an absolute value of time. It is defined in (hh:mm:ss)/mm/dd/yy format. A relative event is an event corresponding to a time point and is specified with a reference point and the offset. It has the format like E1 + [time-string], while E1 refers to any event allowed in Snoop, and [time-string] refer to the time offset.

- Local Composite Event

Local composite events are composed of local primitive events and other local composite events by applying event operators [3, 4]. Since all the constituent events
of a local composite event is defined locally; a centralized detection mechanism is used for local composite event detection.

- **Global Primitive Event**

  *Global primitive events* are events that are defined and detected outside of the current application but are referenced/used by the current application in a distributed environment. Any event (either local primitive or local composite) defined in an application can be a potential global primitive event. Since a global primitive event is defined outside of the application, the information needed by a local application to use this global primitive event should include the global primitive event name and the application name. Other information, like parameters of the global primitive event, are supplied to the local application when this global primitive event is signaled. The parameter computation is handled by the system and is transparent to the user application. Two attributes are introduced in the global event specification: *Event* name and *Application* name. They indicate that the event with the name *Event* name is defined in application *Application* name.

- **Global Composite Event**

  *Global composite events* are related to events occurring from many sites (including the local site). They are constructed with *local primitive events*, *local composite events*, *global primitive events* and other *global composite events* by applying a subset of event operators defined in Snoopy. At least one of the constituent events is a global event (primitive or composite) in a *global composite event expression*.

The extensions to specify global events is shown below.

\[
\text{event name ::= name | Eventname:ObjectName | Eventname::AppId}
\]

\[
\text{AppId ::= SiteName@Appname}
\]

name, Eventname, Objectname, SiteName, and Appname are all Identifiers. For item Objectname the C++ environment used by Sentinel poses some unique problems. For example, it is not possible to send object instances across address spaces using rpc and our parameter list itself is an object.

Finally, for item 4), we had to use a combination of rpc and socket-based communication between the server (GEM) and individual applications. This is necessitated by the fact that the local application needs to be interrupted to indicate a global event of interest and after that a complex parameter list needs to be sent for rule processing.

3 **Architecture**

Since global event detection involves multiple applications from different sites, the special features of distributed systems increase the complexity of event detection as compared to centralized systems. These special characteristics of distributed systems include concurrent processes running at multiple autonomous sites, lack of global time, message delays between sites, etc. Since the communication between applications plays a significant role in system performance, the architecture we choose for global event detection should reduce the communication overhead as much as possible.

We use the client/server architecture to reduce the number of messages communicated between applications if we choose the peer-to-peer approach. The client/server architecture is shown in Figure 1. This approach uses the client/server model to centralize global event detection on a server site. Global event management is a separate server process. It receives requests and messages from clients (local event detectors), builds its global event graph where global events are detected, and sends the event notification to clients when each event is detected. Each application runs LED to detect local events, and communicates with other applications through GEM.

Client/server architecture centralizes the global event detection on a server site, thereby decreasing the communication overhead between applications, and is likely to improve the system performance. On the other hand, since global event manager needs to detect global events in addition to transfer messages between applications, the implementation is more complicated. In Sentinel, client/server model is chosen to implement the global event manager (GEM). The architecture of client/server approach is shown in Figure 1. GEM is running as a daemon on the server site. Each client has a LED to detect local events. LED communicates with GEM using the remote procedure calls (RPC).
3.1 Asynchronous Communication Between Client and Server

To detect a global event, each client should send global event detection request and event specifications to the server, and receive the event to be notified from the server. This sending and receiving action should be carried out in an asynchronous manner. That is, after the client sends requests to the server, it will not be blocked waiting for reply. The client application should be able to continue its work and receive the notification from the server meanwhile.

We use a combination of RPC and socket between the clients and the server. In order to make the server communicate with different client processes running on the same machine, a socket communication interface is added to meet this request. Since each process has its own socket ID, the server can send response back to a specific client according to this socket ID no matter where the client process is running. When the client receives an event message from the server through its socket interface, it will make RPC calls to the server to receive event notifications.

The design is as follows: A client process makes a socket connection with server during its handshake with the server, and the server records the socket ID of this client at the same time. Whenever an event is detected by the server, according to its event subscribers (client application ID), the server will send a response to each such subscriber through socket according to the subscriber socket ID. After the subscriber (a client process) receives this response, it will make RPC calls to the server to get the event notification (event name and parameter list). In Sentinel, this approach is used to implement the global event manager. The architecture is shown in Figure 2.

4 Specification and Processing of Global Events

The global primitive event specification is as follows:

- **local_event_name**

- **remote_event_name**: host_name@application_name

**local_event_name** is the event name defined by user’s application. It can be used as a constituent event name to form composite events. **remote_event_name** is the event name that is defined in the application where this event is detected. host_name and application_name denote the name of the machine and application where this remote event is specified. It is assumed that an event **remote_event_name** has been defined and will be detected by the application **application_name** on the machine **host_name**. This global primitive event can either be specified as a stand-alone event definition, as shown above, or it can be used as a constituent of a composite event definition.

4.1 Global Composite Event

The global composite event specification is similar to local composite event specification except that at least one of the constituent event must be a global event. This constituent event can be presented as a local event name or a global primitive event specification. Below, we present some examples of global event definitions in Snoopy:

```java
class STOCK { public REACTIVE {
    public:
    STOCK();
    int get_total_stock();

    event end(e1) in buy_stock(int qty);
    event begin(e2) in sell_stock(int qty);
    event g1 = STOCK: sugar@appl1;
    event g2 = eIBM: manatee@app2;
    event g3 = (g2\>\> STOCK: e: eagle@app3) + g1;
    event g4 = ![g1, g2, g3];
    rule gr1[g1, cond1, test_action1, RECENT];
    rule gr2[g2, cond2, test_action2, RECENT];
    rule gr3[g3, cond3, test_action3, RECENT];
}
```

Event **e1** is a local primitive event which occurs after the method **buy_stock** is executed. Event **e2** is a local primitive event that is triggered before the method **sell_stock** is executed. Event **g1** is a global primitive event that is triggered when the event **STOCK:e2** is detected by application **appl1** on **sugar site**. Event **g2** is a global primitive event that raises when the event eIBM is signaled by application **app2** on site **manatee**. Event **g3** is a global composite event specified by operator (\>\>) and (+). Its constituent events are composed with global composite event **g1**, global primitive event name **g2**, and global primitive event specification **STOCK:e1:eagle@appl3**. Event **g4** denotes a global composite event with “!” operator. Three rules **gr1**, **gr2**, **gr3** are defined for events **g1**, **g3**, **g4**. Since the above events are declared at the class level, they are detected for each instance of class

Figure 2: A combination of RPC and socket Design Model of GEM
STOCK

4.2 Alternatives For Global Even t Detection

In Sentinel, global primitive events are detected by the corresponding remote sites, and the event notifications are sent to GEM if the event is used by other applications. Unlike a global primitive event, the global composite event detection is more complicated since the constituent events can be either local or global. A composite event can be detected either at the local site or by GEM. The appropriate choice of global event detection site plays a significant role in system performance since it involves network communication. Based on the event detection site, two alternatives are discussed, and the following examples (shown in Figure 3(A)) is used to compare these two approaches.

In the example, a global event \( G \text{AND} \) is composed by a global composite event \( G \text{OR} \) and a local event \( ee \). Event \( G \text{OR} \) is defined on a remote site and is composed of four local events: \((L_1, L_2, L_3, L_4)\) with \( \text{OR} \) operation.

1. Global composite event is detected by GEM

In this case, the whole global composite event tree is sent to the GEM server. Every leaf node of the event tree is to be detected by its corresponding site and the notification will be sent from this site to the GEM after this event is detected. For the above example, the number of communications between node sites and the server is 4, since the event tree has four leaf nodes (corresponding to primitive events). When a global composite event (the root node of the event tree) is signaled by the GEM, an event notification will be sent from the server to the corresponding local site where this event tree is sent from. So, the total number of communication between clients and the server is: \(4 + 1 = 5\).

To consider the general case, we introduce a global composite event tree which is shown in Figure 4.

\( G \) is a constituent event of global event \( E \) and is defined by \( n \) local events \( G_1 \ldots G_n \) on a remote site. \( L_1 \ldots L_m \) are other constituent events of event \( E \). Since the whole event tree is detected by the GEM, each leaf node need to send notification to the server when the corresponding event is detected. The number of messages between clients and the server is: \( m + n + 1 \).

2. Global composite event is detected at the local site

In this case, all the constituent local events of a composite event are detected on its corresponding remote sites. Only when the global composite event is detected, the event notification is sent from the remote site to the GEM. In the above example, the event subtree for node \( G \text{OR} \) (as shown in Figure 5.1(C)) is detected on the remote site. The event tree sent to the GEM will be the one shown in Figure 5.1(B). Since only the node \( ee \) and the node \( G \text{OR} \) need to send the notification to the GEM, the number of communications will be \(2 + 1 = 3\).

To consider the above general case, the event tree of the constituent event \( G \) is detected on the remote site, the number of messages between clients and the server is: \( m + 1 + 1 = m + 2\).

In Sentinel, alternative 2 is used for global composite event detection. It reduces the network communication overhead and thus improves system performance.

5 Implementation Details of GEM

Global event detection is the only task performed at the server site. Server accepts messages from clients, invokes the corresponding service, and sends the message back to clients.

A local event detector at the client site is integrated into the local application. It is implemented as a static library and provides the function calls for event detection. Since LED needs to communicate with the GEM in the server, it should be running as a local daemon. Moreover, as an LED needs to communicate with local application to exchange messages, both LED and the local process should share the same address space. Forking a subprocess for LED does not meet this requirement. A lightweight process (or a thread) is a better alternative to accommodate this goal. Multiple threads execute as concurrent execution streams sharing the same address space performing tasks associated with the desired services.
5.1 Extensions to the Local Even tDetector

Each client has a local event detector (LED) which is composed of an even t detection graph. In addition to detecting local events, LED sends the local event notification to the server whenever a required (by the server) event is raised, and receives the event notification from the server when a global event is detected by the server. To accommodate the above requirement, LED is modified to include extensions to event class hierarchy.

To accommodate global events, a REMOTE class is added to the class hierarchy LED. REMOTE class is a subclass of the EVENT class, and represents global event objects. Each global event that is detected outside of the application and notified from the server is an object instance of REMOTE class. There are two attributes of this REMOTE class: App_ID and instance number. According to the place where a global event is detected, the value of App_ID attribute is assigned in a different way. If a global event is detected outside of the application process, App_ID is the ID of an application where this global event is defined and detected. An application ID is of the form SiteName@AppName which denotes application AppName running on the machine SiteName. If a global event is detected inside the application, the value of App_ID is a reserved word REMOTE, which is to differentiate the global event instance from local event instance. The App_ID attribute combined with the event_name (an attribute of NOTIFIABLE class) make a unique ID for each global event. instance number is the occurrence number of this global event instance.

5.1.1 Extended Local Event tDetector (ELED)

Extended Local Event Detector (ELED) is an extension to LED to detect global composite events at the local site. Similar to LED, ELED is an instance of EVNTLIST class that records information of all the global event instances. Each node of the EVNTLIST linked list is related to a unique application and contains all the global event instances that are detected outside of this application. An event link ed list whih is an ELIST class instance is related to each node and contains all of the global event instances information that belongs to this specific application. Each node of such ELIST instance corresponds to a REMOTE node and becomes a leaf node of the event graph. A composite event can be constructed from REMOTE nodes, Primitive Event nodes, and other composite event nodes. Whenever a global event is detected outside of the application, a GEM Interface will receive the event notification along with application ID and event parameter list from the server and further notifies ELED. ELED then determines the specific EVNTLIST node according to the application ID and propagates the event notification to its corresponding REMOTE event instance.

5.1.2 GEM Interface

To extend LED for supporting global event detection, a nework interface is needed to exchange messages between the client application and the GEM. A GEM Interface on the client site is implemented to communicate with the server. It generates an event tree list which contains global composite events that are to be detected by the server, sends this event tree list to the server, and receives the global event notifications from the server. In order to support different applications running on the same machine, a socket connection is built between client and server, and the unique socket address is recorded by the server for later message reply. In addition to the socket connection, a client application makes remote procedure calls to the server to request global event detection and receive event notifications from the server.

5.1.3 Event Tree Propagation By Client

To decrease the communication overhead between a client and the server, we should send the composite event tree only if the global event nodes inside this tree are other than local event nodes. Since the composite event detection tree is generated according to Snoop operator semantics, one parent node can have at most three children. As a result, a sub-tree with less than or equal to two children is sent to the server only when at least one of the child event is to be detected by the server, a subtree with less than or equal to three children is sent to the server only when at least two of the children are to be detected at the server. According to the event tree sent from a client, the server builds the global event graph. The sub event trees sent to the server is created by spp (Snoop pre-processor) when it parses and analyzes the global event definition in an application. A global event specification file that contains the global event tree information is created by spp. Global event trees are created from the global event specification file run time and are sent to the server during the first handshake between client and the server.

5.2 Implementation Of the Server

GEM is implemented using client/server module as illustrated in Figure 1. GEM is installed on server and provides services to the clients by detecting global events. When a client sends requests to the server, the server sends the requirements to the local service, processes it, and sends the necessary information back to the client. In this way, clients communicate with each other through server in a transparent way.
5.2.1 Architecture

On every local site, local events are detected by the Local Event Detector, and a client application communicates with the server through a GEM interface. In order to receive detections of global events, each client should register and send the necessary information (global event specification) to the server at the beginning of the process. In addition to sending event occurrences to a local event manager (LED) for composite event detection and rule execution of ECA rules, local events need to be sent to GEM for global event detection according to the global event specification. The registration and local event notification to the server are managed by a GEM interface which is an extension to LED.

On the server site, a socket interface is implemented to receive the request and send the reply message to clients. When the server receives the request from clients, it invokes corresponding services, and begins global event detection. Whenever a global event is detected by the server using the event’s subscribers, the server will notify the corresponding clients along with event names and parameter lists through the socket interface.

5.2.2 Data Structures Of Global Event Manager

![Diagram of Global Event Manager](image)

Figure 5: Data Structure of Global Event Manager

The data structure of GEM is illustrated in Figure 5. At each client site, an Extended Local Event Detector (ELED) combined with LED are used to detect local and global events. A GEM Interface that is implemented by socket mechanism is established to communicate with the GEM server which includes sending event detection requests and receiving global event notifications. On the server site, Global Event Manager receives requests from client applications and records the client socket ID and application ID. It also receives event trees from clients and builds a global event graph. Whenever a global event is detected, it will propagate event notification to its parent nodes according to its subscribers, compute its parameter linked list, and send this notification along with its parameter list back to specific clients as appropriate. When a parent receives an event notification from its child (constituent) event, it will record this event instance along with its parameter list for further event detection.

5.2.3 Class Hierarchy In GEM

As shown in Figure 6, the class hierarchy of GEM is similar to that of LED. A PRIMITIVE class in LED specifies primitive event objects that is defined by method signatures inside this application. Since global primitive events denote external events that are detected outside of the local application, the PRIMITIVE class is not useful anymore. Instead, a GLOBAL class is defined to represent the global primitive event objects. Three attributes are defined inside the GLOBAL class: send_sname, sendename, event_no. send_sname denotes the application ID that this event instance needs to be notified by the server after it is raised. sendename is the name of this event that application send_sname uses. It has the same value of ename attribute of a REMOTE class instance which is related to this global primitive event in application send_sname. event_no denotes the instance number of the occurrence of this event. To capture the global event features, two attributes are added to the NOTIFIABLE class: site and sendback. site attribute specifies the application ID where this event is defined and detected. sendback is a flag to indicate whether this event notification needs to be sent to any applications by the server after it is detected. Because of the time delays associate with communication and network failures, “P” and “P*” operators are not currently supported by GEM.

![Diagram of GEM Class Hierarchy](image)

Figure 6: GEM Class Hierarchy
5.2.4 Communication Between A Client And The Server

Since global event detection is distributed between clients and their server, the communication between client and server is important. There are four types of messages passed between client and server:

- Global event detection request from a client to the server.
  This is done during the handshake between a client and the server. The request includes global events that need to be detected by the server. The global event graph in the server is constructed based upon the event trees sent by a client.

- A client name list from client to server.
  This event name list contains all the events that need to be detected in this site and sent to the server after this event has occurred.

As mentioned earlier, instead of propagating every event from a local site, only those events that are used by other sites need to be notified to the server. A global event name table in each local site is used to record such event names. Whenever a new client makes a connection to the server, the global event name table on related local sites is updated dynamically according to the name list sent from the server.

- Event notification from client to server.
  When a local event is signaled, in addition to notifying the local server, it will check the global event name table and sends the notification message to the server if it needs to.

- Event notification from server to client.
  Whenever a global event is detected in the server, it will check its site subscribers, and sends the notification message back to these sites.

6 Conclusions

This paper presents an approach to monitor events in a distributed environment to support cooperative applications. Toward this end, Snoopy was extended and the snoopy pre-processor was modified in addition to implementing the GEM itself.

The main contribution of this paper are: identification of the functional requirements of distributed event detection, choice of an architecture and implementation of GEM. In the client/server architecture, GEM receives requests from client applications and provides services (RPC calls) to detect global events and sends event notification back to the client whenever a global event is detected. This model supports several applications running on the same machine as well as those running on the different machine. Finally, several applications have been implemented using GEM.

References


