Transactions on Information and Communications Technologies vol 19 © 1998 WIT Press, www.witpress.com, ISSN 1743-3517

Events Specification in Active Database Management System

Carlos André Reis Pinheiro
Pós-Graduação em Ciência da Computação
Departamento de Ciência da Computação
Instituto de Matemática
Universidade Federal Fluminense
Praça do Valonguinho, s/nº - 4º andar
24210-130, Niterói/RJ - Brasil
carp@lci.dcc.uff.br

Abstract

Active Databases Management System have been objective of countless researches in the last years. Its active functionalities, as well as the concept of active objects or events are used in several application areas, not only in technology of databases. However, there is not a clearly defined notion distinguishing systems considered "actives" of the "non actives". However, certain essential characteristics are requested for the elaboration of these active functionalities, as ECA rules definition, events specification, production rules, and others. The present paper goals to give a generic view of one of those mentioned characteristics that I consider to be one of the most fundamental topics for the elaboration of active systems, that is the events specification. Starting from a solid and concise specification of the concerning events to a certain application, the elaboration of a efficient rules execution model for an active database management system is possible. This way, it will be given a general vision of the ECA rules, the event concept, its relationships, its types and classifications, and the pertinent operators to the events algebra. The establishment's necessity of the conditions that leads the events specification will be exposed, the definition of the executed actions when of the occurrence of events and the conditions validation, approached inside of the concept of the production rules.

1 Introduction

Quite explored area in researches in the technology database area has been the active database management system, relational or object-oriented.

A lot of applications considered "non traditional" are not supported or contemplated for the tradicional database management systems. These applications need to monitor the state's changes that happenes in its databases continually and has to discharge appropriate actions automatically. The moment of the event detection and the execution action related to the detection, are fundamental for these applications, what makes them petitioning of active systems for the management of its data bases.

An active database management system should be supply all the characteristics of a normal or "passive" DBMS, besides certain functionalities that makes it "active" Dittrich&Gatziu&Geppert[4]. It should support a process of rules definition and management (ECA rules - Event, Condition and Action's rules), in which will take reaction procedures in response to the specified events. It should also supply a meaning for events definitions, events conditions and events actions, it means that, a way in which the situations are described by the event-condition-action tuple. It should also has to supply a management methodology for the defined rules and the evolution of the established rule's base Geppert&Gatziu&Dittrich[11], Buchmann&Zimmermann& Blakeley&Wells[12], that is, to have the capacity to administer a set of rules definition, wherever they are stored, and manage the changes that may happen in the elapse of time. An active DBMS should have an execution model for the established actions. This execution model should be capable to detect the events occurrences and signal them, evaluate its conditions after the detection of them, executing the defined actions if the conditions are satisfied. Besides that, it should possess a well defined semantic's execution, it means that, the event specification, its detection and signaling should be very defined.

As additional characteristic for active database management systems, we can point out the capacity of the events representation in terms of event, condition and action rules, through the establishment of a data model.

This aspect in particular is the focus of the present paper, which I consider the main key for the success of an effective implementation of an active database management system. The base for implementation of an effective active database management system is a concise and very

defined model, semantic and syntactically of the events, the pertinent conditions of them and the actions that will be taken when the occurences and signaling of the event mentioned above[16-20].

The use of rules to support the active functionalities seems to be a consent in the database community. As mentioned previously, a rule consists in three components: an event, a condition and an action. Those types of rules, constituted by those three rules elements are denominated ECA (Event, Condition, Action). Once these rules are declared, the system should monitor the events occurrence continually, to evaluate the condition when the event will be signalled and then, execute the action if the appraised condition is true.

The events that will be monitored by the system can be classified as follows Dittrich&Gatziu&Geppert[4,5].

1.1 Database events

Typically insert, delete and update operations in relational databases.

1.2 Temporary events

Typically relative and absolute events related to a defined time scale.

1.3 External events

Typically events detected out of the database management system, in spite of the rules that will act on the same events processed by own database management system.

2 Events Definition

An event is an atomic occurrence Chakravarthy&Deepak[7,8],

We can make a distinction between logical events and physical events. In spite of being a difference, sometimes tenuous, that distinction is extremely important. The logical events are related to an operational concept, for example, an *end-of-insert* operation, and the physical events are related to the operational process in itself, for example, the point in the code after the last operation before coming back of an insert. Both kind of events fit in the concept of atomicity of an occurrence, as defined previously. However, the logical events correspond the event specification in conceptual level, differently of a physical event specification, the one that will be mapped. This way, the logical events, in conceptual level, will be mapped for physical events

using a specification and a mapping methodology or choosing an implementation that corresponds to a mapping, depending on how it will be done: by a compiler or manually. This methodology is consistent with the conceptual and physical levels of a database management system, and clearly differs from the implementation of the specification. A non univocal relationship exists in the mapping of the events, it means that a logical event is only mapped for a physical event, however, a physical event can have correspondence in more than a logical event. Physical events are those detected by the system.

Database operations produces state's changes that originates the events. Those execution operations or transactions are concomitant with the state's changes. In database applications, those are actually the events that interest us. Such state's changes are detected through the use of parameters associated to the operations. Consequently, we can understand that the events are properly associated with the operations mentioned, and its parameters are seen as parameters of the event associated to the operation.

3 Events Relationships

A logical event is conceptual defined as being atomic. On the other hand, a physical event can be seen as an occurrence in the defined time scale. This way, we can associate an occurrence time for each moment of the event. As we are going to map the logical events in physical events, we can consider that the occurrence time of a logical event is the same occurrence's time of a physical event associated to it.

Keeping in mind this notion of time scale for the event occurrence, we can verify that the events can precede or follow other events. The events may not have one another relationship mentioned to a time scale and its specific functionalities Chakravarthy&Deepak[8].

When the existence of an event causes in a definitive and irrefutable way the occurrence of another event in the future, we say that the first causally precedes on this last one. This relationship of causally precedes satisfies the transitive, irreflexive and asymmetric properties for any events. If an event E1 causally precedes event E2, and this event E2 causally precedes event E3, then the event E1 causally precedes event E3 (transitive). Also, we have an event that cannot causally precedes on itself, it means that, an event E1 cannot causally precedes the same event E1 (irreflexive). Last, if an event E1 causally precedes event E2, the event E2 cannot causally precedes event E1 (asymmetric).

When the occurrence time of an event is exactly the same of the occurrence time of another event, we say that these events possess

simultaneous occurrence. In monoprocessor environment, or in the absence of temporary and explicit events (external events not detected by the system), the events cannot have simultaneous occurrence. However, in multiprocessor environment, or in the presence of explicit events, it is possible to have more than one event in simultaneous occurrence. Naturally, this adds more complexity in the event detection processes as well as in the evaluation of its parameters.

When an event have a guarantee of its occurrence, we use to say that this is a definite event. For example, for the causally precedes' rule, we can verify that if an event E1 causally precedes an event E2, and the event E1 has occurred, we have that event E2 is a definite event, because its occurrence is right and guaranteed by the existence and occurrence of the event E1.

It is important to do a distinction among the interpretation of the conditions and the events. A condition is a boolean value of data values, for example, "Andre is older than 21". A condition doesn't produce any real effect, to not to be, to change the database state. A condition just can be valid in an time interval, while an event doesn't possess this limitation, because it is atomic by definition. The conditions define the possible database states and consequently they will be used in the event, condition and action rules.

4 Events Modifiers

The event modifiers are used to transform a physical event in a logical event. Actually, a established correspondence among the conceptual and physical events exists. We can highlight three different aspects in the processing that an event can suffer Chakravarthy&Deepak[8].

4.1 Event occurrence

When an event happens, its formal parameters are instanciated with the values of the current parameters.

4.2 Event detection

When an event is detected, the parameters are collected and stored by the events detectors.

Transactions on Information and Communications Technologies vol 19 © 1998 WIT Press, www.witpress.com, ISSN 1743-3517

The event is signalled through the shipping of an interruption message to evaluate the condition, indicating its occurrence and passing the current parameters.

Database operations, as insert and delete, are event expressions, and they have a just meant inside of a time interval. These events cannot be treated as instantaneous occurrences, unless the event corresponds to a pre-determined point in a specified time interval. Event modifiers supply a mechanism to create logical events, in conceptual level, that correspond to specific points in a certain time interval, and for mapping all the points of interest for the physical events in the system.

This way, two types of event modifiers are defined, *begin-of* and *end-of*, to transform an arbitrary time interval in the time escale established in two logical events. These two event modifiers create instantaneous corresponding events, and maps the points of occurrence of the events for a physical level. These event modifiers cannot map any operation or time interval for two very defined events. Actually, these events correspond to the events or methods *before_transaction* and *after_transaction* supplied by most of the programming languages motivated by event and native languages of the object-oriented databases.

As part of the event modifiers mapping, the parameters that will be computed also need to be specified. In way, the parameters of the event modifiers like *begin-of* includes the parameters of the entrance of the operation, while the parameters of the event modifiers like *end-of* includes the entrance parameters as much as the parameters of exit of the operation.

5 Events Classification

The events can be organized inside a hierarchy events classes. Each event class possesses an unique type of event and an instance of an event class is identified by its class type and for the moment of its occurrence. The term event is used to refer an event class or an instance of a certain event of an event class. The attributes of an event are instanciates in runtime, when the event really occurs. Its attributes can be inherited of a father's class and the same event can serve as father class to another event inherits its attributes. Everything works inside of a events classes hierarchy philosophy. However, two attributes in particular takes part in all the events, its type and the moment of its occurrence. It is exactly through these attributes that events are

identified and its occurrences detected. This way, the actions associated to the events can be discharged depending on the validity of the established conditions, following the ECA rules methodology. These procedures can be verified in the basic database management system operations. For example, we can establish that *end-of-insert*, *end-of-delete* and *end-of-update* are event classes. This way, for each insert, delete and update operations, an instance of the respective event class is created, which can have additional parameters of its father's class, for example as the field's name that is being inserted, deleted or updated.

The events can be classified in two basic types Dittrich & Gatziu & Geppert [4], Gatziu & Dittrich [6], Chakravarthy & Deepak [8], Fritschi & Gatziu & Dittrich [13], Hull & Jacobs [14], Ghanderarizadeh & Hull & Jacobs et al.[15]: primitive events and composite events.

Each event, no matter if it is primitive or composite, should possess a group of very defined parameters, that will be instanciated to the each event occurrence, that for its time, it is son of an event class of a certain type and it inherits all its properties. The inheritance concepts and hierarchy are the same of the object orientation for any aplication type or methodology.

5.1 Primitive events

Primitive events are the events that are pre-defined in the system. For each primitive event defined in the system, it should be equally defined an efficient detection mechanism of the same event. The primitive events can be subclassified in database events, temporal events and explicit events.

5.1.1 Database events

Database events are all those events related with the database operations, like insert, delete and update transactions. The database events corresponds to all the operations that manipulate data in its respective base. These events are associated to the *begin-of* and *end-of* of the insert, delete and update operations. These two events can be associated to any arbitrary function and its parameters corresponds to what is visible in the context that the logical events are mapped for physical events. This capacity to generalize the event association in arbitrary functions, gives us the possibility that any function can be reorganized by the system. This functionality is quite useful for the relational or object-oriented database management systems, which the number of data manipulation operations are not fixed.

5.1.2 Temporal events

A temporal event is defined by the specification of a point in the established time scale. In general, there are two ways of specifying a point in this time scale: using the absolute values or using an increment starting from a defined point in the time scale.

5.1.2.1 Absolute events An absolute temporary event is specified through an absolute value in the established time scale. It is exactly this value in the time scale that supplies the moment of the occurrence event. That event type is quite usual in several situations, mainly, in the ones that they need some processing batch, for example, backups. Suppose that a certain application needs to accomplish a daily backup of its data. Thus, defines then an absolute temporary event, with a specified hour and with a daily execution indication, or if it is necessary to generate a monthly transaction, a payroll, for example, an absolute temporary event should be specified, with a specified hour and with a monthly execution indication.

5.1.2.2 Relative event As an absolute temporary event, a relative temporary event possesses an univocal relationship with the established time scale, however, with a significant difference. A value incremental of time, a value referencial exists to specify the real event occurrence point in the time scale established.

A relative event, by definition, contains an absolute event. Consequently, a relative event is also a primitive event, as well as an absolute event is. We can consider that a relative event has its occurrence originating from an absolute event. We also verified, that an absolute event is, by definition, a definitive event, because its existence is guaranteed by the specification of its occurrence moment. Consequently, a relative event is also a definitive event, because it is composed by an added absolute event of an incremental value of time, guaranteeing its occurrence this way, therefore, its existence.

5.1.3 Explicit events

Explicit event are those that are not part of the system. These events are defined out of the system, through additional applications, or by the own user. Naturally, its parameters are, similarly, defined out of the system. This way, the explicit events are detected out of the database management system, however they are signalled by it, through the passage of its parameters. In spite of the explicit event not be detected by the system, it is accepted by the system and then processed. The mechanism that detects and signals an explicit event is detected and

signalled for the system can also be used to signal non explicit events, even if it has not to happen in fact. This can be useful when a certain application has to signal a non explicit event in an explicit way, as if it was "forcing" its occurrence. We can observe then, that the only difference between the detection of an event and the signaling of an explicit event is that in the first case the parameters are computed and processed by the own database management system, and in the second, they are simply supported by DBMS.

5.2 Composite events

Composite events are those events that are formed by the application of a logical operators group on primitive events, or even on composite events. This way, those events are seen as event expressions. For the formation of those expressions, a logical formalism should be followed by the event algebra.

The event algebra, associated with this logical formalism, is the mechanism in which a knowledge base can be developed Stefik[2]. This knowledge base will be used for the implementation of the "active functionalities"; through the definition of production rules using the ECA rules, in the development of the active database management system, relational or object-oriented.

5.3 First order logic for composite events

For the composition of primitive events in composite events, or even of composite events in another composite events, we should use the concept of events expressions. The actions on these events expressions should follow a logical formalism. The formalism used here will be the first order logic Russell&Norvig[1].

Inside of the first order logic, we have sentences that can be formed by atomic sentences or, composite sentences through the use of logical connectives. These sentences can be actually faced as events expressions. This way, the events expressions should follow the first order logic rules and possess its properties. Consequently, the event expressions can be assembled starting from the operators (connectives in the first order logic) listed as follow:

5.3.1 Disjunction($\vee$)

The disjunction of two events happenes at least when the occurrence of one of them is detected. For example, if we have two events, E1 and E2, being composed through an event expression like E1 $\vee$ E2, this

expression will be valid, that means, detected, when the occurrence of any of the events E1 or E2 happens. This operator is usual when an application needs to discharge a procedure based on the occurrence of any event among a list of specified events.

5.3.2 Conjunction($\wedge$)

The conjunction of two events happenes, if and only if, the occurrence of the two events is detected. For example, if we have two events, E1 and E2, being composed through an event expression like E1 $\wedge$ E2, this expression will be valid, that means, detected, only when the occurrence of the two participant events of the expression, E1 and E2. This operator is usual when an application needs to discharge a procedure based on the occurrence of all the events of a list of specified events.

5.3.3 Implication($\Rightarrow$)

The implication or composition of two events is valid, that means, detected, when the occurrence of the last event is signalled after the occurrence of the first event had been detected. For example, if we have two events, E1 and E2 being composed through an event expression like E1 $\Rightarrow$ E2, this expression will be valid, that means, detected, when the occurrence of E2 is signalled after the occurrence of E1. This implies that the moment of the occurrence of the event E1 in the defined time scale is certanly smaller than the occurrence of the event E2 in the same time scale. This operator is usual when an application needs to discharge a procedure based on the order of occurrence of the events in a list of specified events.

6 Production Rules

The active databases are characterized by supplying an efficient mechanism for the implementation of production rules Hanson&Widom[9], Kim&Chakravarthy[10], that actually is, the action of turning a database management system active. These rules correspond to the possible database operations, occurrences of database state, as well as the transition among these states, and other procedures and tasks related to a database management system.

In a general, the production rules for the active database systems possess the following form: given a pattern, they have a corresponding action Hanson&Widom[9].

These rules are known as rules based on patterns. Here, pattern can also be indicated as condition or predicate. This reinforces the concept of the use of first order logic for the events algebra, since this it is used

of the predicates logic for the composed sentences formation. The production rules are discharged when the database state changes. Consequently, these rules are discharged implicitly by events, as an insert, a delete or an update of data. For the active database systems, these events should be defined explicitly. A production rule then could have the following format Hanson&Widom[9]:

on event
if condition
then action

For this formation, these rules are also indicated as rules based on events. In this case, the rule is discharged when the occurrence of the associated event is detected by the system. Once the rule is discharged, that is to say, its associated event was detected and signalled, its condition will be evaluated. If the condition is satisfied, the action regarding this rule will be executed.

The ECA rules can be implemented in the practice to attribute active functionalities in the database management systems, exactly like this.

7 ECA Rules in DBMSs Non Actives

In view of the absolute majority of the database management systems, that are at least in the market, they are traditionally "passive", it would be important to have the possibility of "active" functionalities implementation to the same databases. These facilities could come as ECA rules' designer Chakravarthy&Deepak[8], helping not only in the specification of the referring events to a certain application, as well as in the documentational organization of the same databases.

The implementation of ECA rules will be shown through the use of a visual interface. Starting from the definition of a rule, a trigger creation DDL will be generated in a common relational database. The implementation of these triggers, creates, in the practice, an ECA rules execution model for a certain application. The elaboration of this execution model allows us to implement active functionalities in traditional databases.

The use of a graphic interface for the ECA rules specification has a serie of advantages. A graphic interface increases significantly the visualization of the data that to be manipulated, facilitating the elaboration model. Through distinct procedures for creation of the DDLs to each kind of database management system, we can create an abstraction layer in the elaboration model, independent of the database

used, increasing the portability and the flexibility of the database management system. A very important fact is that the use of this interface will propitiate a visualization of the data in level of ECA rules, and not in triggers level associated to tables separately. This fact facilitates a lot the maintenance of the defined model.

For example, a management system of retail stores possesses a sales control. Each sale made is associated to the salesperson that accomplished it, and calculating the percentage reserved for the salesperson. An active functionality would be the automatically update of the salesperson's salary after each sale made.

For it, we can define a production rule for the implementation of this functionality. Through an ECA rule, the event should be defined, the condition associate to it and the action to be accomplished when of the occurrence of the same event. The event definition is made through the interface proposal, being its occurrence detected through a database trigger. The condition or conditions are established in agreement with the problem's necessities. In our example, we defined only one condition, in this case, a sale not null. The action to be executed it is placed in the body of the trigger, that for us, it is an algorithm to update the salesperson's salary.

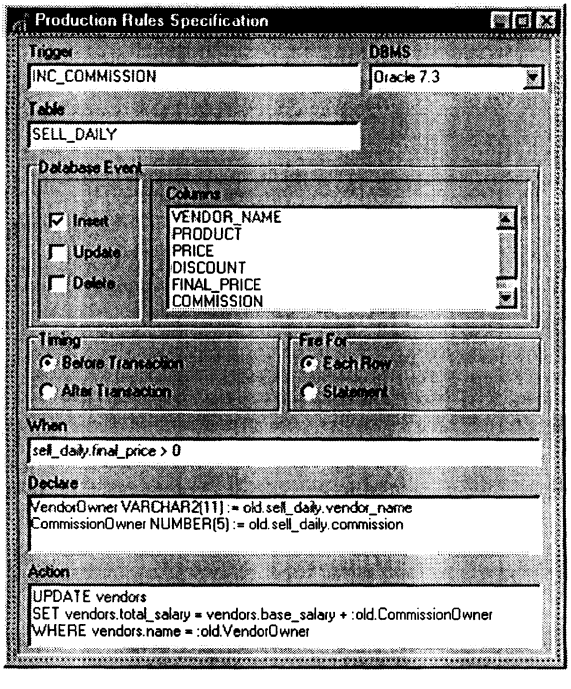

Figure 1. Graphic interface for events specification.

The production rule definition above would generate a DDL for the trigger's creation in a Oracle database, and it would have the following format:

```
CREATE TRIGGER inc_commission
BEFORE INSERT ON sell_daily
It GOES EACH ROW
WHEN (vendors.final_price > 0)
DECLARE
    VendorOwner VARCHAR2(11) := old.sell_daily.vendor_name
    CommissionOwner NUMBER(5) := old. sell_daily.commission
BEGIN
    UPDATE vendors SET vendors.total_salary =
    vendors.base_salary + :old.CommissionOwner
WHERE vendors.name = :old.VendorOwner;
END;
```

This way, for each sale made by a certain salesperson, its percentage will be added automatically to its total salary.

Naturally, the DDLs generated for each event, condition and action rule, should be submitted to the database management system used in the application in subject.

Through the use of the proposed interface, it can be generated for the same ECA rule definition, differents DDLs of triggers creation, for different database management systems. This way, it could be settled down a rules execution model independent of the database management system used.

8 Conclusion

The commercial databases, at least most of them, possesses the capacity to work with a great mass of data, and every day they supply to a larger spectrum of functionalities to turn them more "friendly" in the use and more powerful in the execution. Its interfaces are improved and more complexity of procedures and functions are added. However, a "uncomfortable" still persists in these database management systems. The system, as a whole, manipulates this mass of data in a "passive way". Many researches in database technology area are turned to soothe this "problem". These researches turns around the active functionalities implementation for the database management systems. These active functionalities are defined and implemented through the ECA rules. These rules facilitates the creation of a knowledge base and they enable

the implementation of production rules, that actually, is the concept of active database in operation.

The definition of a good execution model and a concise establishment of the ECA rules, as the events definition, the conditions linked to it, and the actions related to the event in agreement with the validation of the respective conditions, is the base for a solid implementation of active functionalities in databases management systems. In this sense, it was objectified to elucidate the understanding of the work methodology based on the events specifications for the production rules implementation.

Acknowledgments

I would like to thank Ana Cristina Bichara Garcia and Luciana Ferraz Thomé, for the incentive and suggestions for the execution of this present paper.

References

[1] Russell S., Norvig P. *Artificial Intelligence : A Modern Approach,* Prentice Hall, 1995.

[2] Stefik M. *Introduction to Knowledge Systems,* Morgan Kaufman Publishers, 1995.

[3] Rich E., Knight K. *Artificial Intelligence,* McGraw Hill, 1991.

[4] Dittrich K. R., Gatziu S., Geppert A. *The Active Database Management System Manifesto,* 2nd Workshop on Rules in Databases (RIDS), Athens, Greece, September 1995, Lecture Notices in Computer Science, Springer 1995.

[5] Dittrich K. R., Gatziu S. *Team Issues in Active Database Systems,* Intl. Workshop on an Infrastructure goes Temporary Databases, Arlington, Texas, 1993.

[6] Gatziu S., Dittrich K. R. *Events in an Active Object-Oriented Database System,* 1st Intl. Workshop on Rules in Database System, Edinburg, 1993.

[7] Chakravarthy S. *Comparative Evaluation of Active Relational Databases,* Technical Report UF-CIS-TR-93-002, Database System R&D Center, Computer and Information Sciences Departament, University of Florida, Gainesville, 1993.

[8] Chakravarthy S., Deepak M. *Snoop: An Expressive event Specification Language goes Active Databases,* Technical Report UF-CIS-TR-93-007, Database System R&D Center, Computer

and Information Sciences Departament, University of Florida, Gainesville, 1993.

[9] Hanson E. N., Widom J. *An Overview of Production Rules in Database Systems,* Technical Report UF-CIS-TR-92-031, Database System R&D Center, Computer and Information Sciences Departament, University of Florida, Gainesville, 1992.

[10] Kim S. K., Chakravarthy S. *Confluent Rule Execution Model goes Active Databases,* Technical Report UF-CIS-TR-95-032, Database System R&D Center, Computer and Information Sciences Departament, University of Florida, Gainesville, 1995.

[11] Geppert A., Gatziu S., Dittrich K. R. *Rulebase Evolution in Active Object-Oriented Database Systems: Adapting the Past to future Needs,* Technical Report 95.13, Institut für Informatik, Universität Zürich, Zürich, Switzerland, 1995.

[12] Buchmann A. P., Zimmermann J, Blakeley J. A., Wells D. L. *Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions,* Technical Report, Department of Computer Science, University Darmstadt, Darmsstadt, Germany Cumputer Science Laboratory, Texas Instruments, Inc., Dallas.

[13] Fritschi H., Gatziu S., Dittrich K. R. *FRAMBOISE - an Approach to construct Active Database Mechanisms,* Technical Report 97.04, Institut für Informatik, Universität Zürich, Zürich, Switzerland, 1997.

[14] Hull R., Jacobs D. *Language Constructs goes Programming Active Databases,* Computer Science Department, University of Southern Califórnia, Los Angeles.

[15] Ghandeharizadeh S., Hull R., Jacobs D., Castillo J., Escobar-Molano M., Lu S., Luo J., Tsang C., Zhou G. *On Implementing Language goes Specifying Active Database Execution Models,* Computer Science Department, University of Southern Califórnia, Los Angeles.

[16] Chakravarthy S. *Architectures and Monitoring Techniques goes Active Databases: An Evaluation,* Technical Report UF-CIS-TR-92-041, Computer and Information Sciences Departament, University of Florida, Gainesville, 1992.

[17] Hanson E. N., Johnson T. *Selection Predicate goes Active Databases Using Interval Skip Lists,* Technical Report TR94-017, Computer and Information Sciences Departament, University of Florida, Gainesville, 1994.

[18] Chakravarthy S., Tamizuddin Z., Zhou J. *The Visualization and Explanation Tool goes Debugging ECA Rules in Active Database,*

Technical Report UF-CIS-TR-95-028, Computer and Information Sciences Departament, University of Florida, Gainesville, 1995.

[19] ORACLE *Oracle7 Server Concepts,* Release 7.3 Part No. A32534-1, 1995.

[20] ORACLE *Oracle7 Server Application Developer's Guide,* Release 7.3, Part No. A32534-1, 1995.