A Distributed Rule Mechanism for Multidatabase Systems

Vasiliki Kantere[1], John Mylopoulos[1], and Iluju Kiringa[2]

[1] University of Toronto, 40 St George St., Toronto, Canada
{verena,jm}@cs.toronto.edu
[2] University of Ottawa, 800 King Edward, Ottawa, Canada
kiringa@site.uottawa.ca

Abstract. We describe a mechanism based on distributed Event-Condition-Action (ECA) rules that supports data coordination in a multidatabase setting. The proposed mechanism includes an ECA rule language and a rule execution engine that transforms and partitions rules when they are first posted, and then coordinates their execution. The paper also presents a prototype implementation in a simulated environment as well as preliminary experimental results on its performance. This work is part of an on-going project intended to develop data coordination techniques for peer-to-peer databases.

1 Introduction

In the last fifteen years, relational and object-oriented database management systems have been augmented with a mechanism for expressing active behavior to yield active database management systems (ADBMSs). Usually, ADBMSs use Event-Condition-Action (ECA) rules as the standard mechanism for capturing active behavior. Using a set of ECA rules, an ADBMS performs database definition and manipulation operations automatically, without any user intervention. However, ECA functionality has been considered mostly in centralized environments [4]. Some attempts were made to integrate ECA rules in distributed database systems in [18], [6], [7], [1], and [14]. These approaches usually elaborate on some of the aspects of the problem, but do not present a complete solution. A notable exception is the vision for ECA functionality in a distributed setting discussed in [4].

The recent surge of interest in the peer-to-peer (hereafter P2P) architectures of networking should motivate a change in our attitude towards distributed ECA rule mechanisms. Briefly, a P2P architecture is a dynamic node-to-node mode of communication over the Internet. Existing P2P applications support mostly file exchange, and there is no obvious way of performing advanced querying or enforcing consistency on data across peers. Overall, the existing P2P applications do not deal with data management issues. However, many application domains need the kind of advanced data management that would be offered by a P2P system to be able to manage efficiently data residing in peer databases [17], [2], [13]. One such domain is health care, where

R. Meersman et al. (Eds.): CoopIS/DOA/ODBASE 2003, LNCS 2888, pp. 56–73, 2003.

hospitals, family doctors, pharmacists, and pharmaceutical companies maintain databases about patients and would like to participate in a P2P system in order to exchange information about medical histories, medication, symptoms, treatments and the like for patients. Another example is genomic databases. There are many genomic data sources all over the world, from widely known ones, like GenBank and GDB, to individual and independent lab repositories [13]. It would be useful for scientists to be able to share such data and exploit information from different databases that is relevant to their research. Further domains that would benefit from P2P database technology are: e-commerce in general (e.g. buyers and sellers with the same business interests would like to share information and build *ad hoc* communities), and news industry (e.g. TV channels, radio stations, and press could set up partnership communities for sharing information).

In this paper we present a prototype distributed rule mechanism for a multidatabase system that we intend to use in a P2P setting. We consider multidatabase systems consisting of autonomous databases that reside on different nodes of a network and intend to exchange information without complying with any central administration. However, for simplicity we assume that the heterogeneity issue is solved through a mapping mechanism (e.g. mapping tables [13]). In particular, we propose an ECA rule language that includes a rich event language for the expression of composite events. We have also designed an execution model that minimizes the number of communication messages among the multidatabases involved in the execution of a rule. The proposed evaluation procedure is fully distributed, and involves partial evaluations of rule components in different databases, together with the composition of partial results into a global evaluation. We have also developed a prototype implementation of the proposed mechanism and have performed preliminary experiments in a simulated environment comparing it with a centralized rule execution model.

The next section gives specific scenarios that illustrate the usefulness of the active functionality in a multidatabase environment. Section 3 presents the main features of our ECA rule language. An execution model for the rule language of Section 3 is given in Section 4. Section 5 describes our prototype implementation, and reports on some experimental results. Section 6 discusses related work. Finally, Section 7 concludes the paper, and raises issues that need further work, particularly the intended application of our framework in the P2P setting.

2 A Motivating Example

Assume that there is a multidatabase system where the involved multidatabases can be databases of family doctors, hospitals and pharmacists. This example is adapted from [3]. Fig. 1 depicts a subnet of such a system with three multidatabases, on top of which resides a rule management system.

Assume that Dr. Davis is a family doctor and his database, DavisDB, is member of the multidatabase system. DavisDB is acquainted with the database of a pharmacist,

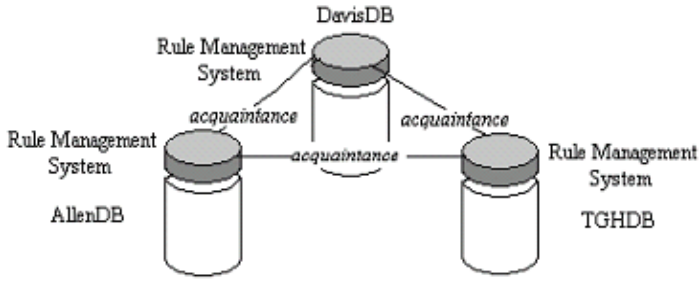

Fig. 1. Acquainted Multidatabases

AllenDB and with the database of the Toronto General Hospital, TGHDB. Also, AllenDB is acquainted with TGHDB. The three databases contain the following tables:

DavisDB:
Visit (OHIP#, Date, Time, Symptom, Diagnosis)
Prescription (OHIP#, DrugID, Date, Dose, Quantity)
DiseaseOccur (OHIP#, Date, DiseaseDescr)
Treatments(DiseaseDescr, TreatmRecord, DrugsRecord)
Allergies (OHIP#, Allergy)

AllenDB:
Presciption (Prescr#, CustName, CustPhone#, DrugID, Dose, Repeats, DiseaseDescr)
Allergies (DrugID, Allergy)

TGHDB:
DiseaseOccur (Date, OHIP#, DiseaseDescr)
Treatment(TreatID, TGH#, Date, DiseaseDescr, TreatDescr, PhysID)
Admission (AdmID, OHIP#, AdmDate, ProblemDesc, PhysID, DisDate)
MedRec (OHIP#, Date, Time, Record)
Medication (OHIP#, Date, Time, DrugID, Dose)

Suppose that Dr. Davis wants to keep track of treatment methods and medications for a number of specific diseases. Thus, he wants to get information about treatments in the hospital and the medication that Mr. Allen is selling after prescription for this disease. The appropriate rule for this is:

Rule 3:
when TGHDB.('insert',
(Treatment,(_, _, _, <DiseaseDescr_value>, <TreatDescr_value>, _)))
OR AllenDB.('insert',
(Prescription,(_,_,_, <DrugID_value>, _, _, <DiseaseDescr_value>))))
if <DiseaseDescr_value> in DiseaseDescrSet
(*where* DavisDB.('retrieve',15, {DiseaseDescrSet})
then DavisDB.('update',(Treatments, (<DiseaseDescr_value>, _, _)))

The underscores denote values of the corresponding attributes that are of no interest or computable at runtime. The *where* clause denotes that we get the set of values of the *DiseaseDescr* attribute of the *Treatments* relation by performing the query 15 below on DavisDB:

Query 15: *Select* DiseaseDescr
From Treatments

Suppose that a patient visits Dr. Davis and the latter suggests that she should be admitted to the hospital for examination. When she is admitted to the hospital, information on the diagnosis of Dr. Davis should be transferred to the TGHDB. To do this, the following rule may be appropriate:

Rule 4:
when DavisDB.('insert', (Visit, (<OHIP#_value1>,
<Date_value>,<Time_value>, _,<Diagnosis_value>))) «
TGHDB.('insert', (Admission, (_, <OHIP#_value2 >, _, _, _, _)))
if <OHIP#_value1> = <OHIP#_value2>
then TGHDB.('insert', (MedRec, (<OHIP#_value2>,
<Date_value>, <Time_value>, <Diagnosis_value>)))

The symbol '«' means sequence and is introduced explicitly in the next section. This rule denotes that when there is an insertion in the *Visit* relation of DavisDB for a patient, and afterwards there is an insertion in the *Admission* relation of TGHDB for another (possibly different) patient, if the two patients are the same person according to their OHIP#, then insert to the *MedRec* relation of TGHDB a tuple with information about the diagnosis of Dr. Davis for that patient.

3 Distributed ECA Rule Language

The ECA rules have the following general form:

R: *when <event>,*
[if <condition >,]
then <action>

where the brackets denote an optional part. Many ECA rule languages have been proposed in the previous decade for centralized environments. Most of them involved event, condition and action languages that provided only basic operators. However, there are some rich event languages in the literature [10]. The main goal in proposing a rule language for a multidatabase environment is to provide clear semantics specifically adapted to the distributed nature of such a system, so that they are able to express events, conditions and actions on more than one databases and that will allow distributed evaluation of rules. Also, such a rule language should be rich enough in order to be suitable for a variety of multidatabase systems, and furthermore, for P2P database systems. As far as we know there are no rule languages for distributed database systems suitable for distributed evaluation. Moreover, the existing ones for centralized environments offer event languages with semantics that cannot be used for partial and asynchronous evaluations of composite events.

In the rest of this section we define the languages that we use in order to declare each one of the three ECA parts, namely event, condition, and action.

Operator	Type	Function	Syntax
∧	Binary	Logical AND	<event1>∧<event2>
∨	Binary	Logical OR	<event1>∨<event2>
!	Unary	Logical NOT	!<event>
«	Binary	Loose Sequence	<event1> « <event2>
*	Unary	Zero or more occurrences	*<event>
+	Unary	One or more occurrences	+<event>
#	Binary	Exact number of occurrences	<number of occurrences>#<event>
&	Binary	Maximum number of occurrences	<number of occurrences>&<event>
$	Binary	Minimum number of occurrences	<number of occurrences>$<event>
>	Binary	Strict sequence	<loose sequence expression> > <not expression>

Fig. 2. Event Operators

3.1 Event Language

The event language that we propose provides a set of operators with clear semantics for a multidatabase environment. These semantics allow the construction of a wide variety of composite events. We believe that the high level of expressiveness of the event language makes it suitable for many types of multidatabase systems. A simple or primitive event SE in a database DB, which we denote as DB.SE, can be either a database or a time event. A time event can be an absolute, a relative or a periodic time event. A database event is one of the following four types of primitive database operations: *retrieve*, *update*, *insert*, and *delete*. A composite event is an expression that is formed by applying the operators of an event algebra on simple or composite events. An event instance is an occurrence in time of the respective event. Generally, an event instance starts at a time point t_s and ends at a time point t_e. Our convention is that an event instance is instant and occurs at the time point t_e.

The event algebra that we propose for a multidatabase system is shown in the table of Fig. 2. The general form of an operator is $OpType_{ti}$, where *OpType* is the type of operator and *ti* is the time interval within which an instance of the composite event specified by the operator should be completed. More specifically, for every instance of a composite event, the earliest simple event instance and the latest one should have a time distance equal or shorter than the time interval denoted by the value of *ti*. The *ti* parameter of the operators is the main difference between our event algebra and the ones used for the declaration of composite events in centralized database environments. The time interval of an operator provides reference time points (either relative or absolute) that allow the performance of partial asynchronous distributed evaluations in several sites, which produce partial results (i.e. event instances) that could be used for the detection of an event instance of a global rule.

Let CE(t) denote a composite event instance occurring at the time point t. Also, let E denote an event expression (either simple or composite). Then the following defines the event operators of our language (See Fig. 2)[1].

1. Conjunction: $\wedge$

$CE(t) = (E1 \wedge_{ti} E2)(t) := E1(t_1)$ *AND* $E2(t_2)$, where $t_2 = t$ if $t_1 \le t_2$, or $t_1 = t$ if $t_2 < t_1$, and $t_1, t_2 \in ti$. Thus, the conjunction of E1 and E2 given ti means that both E1 and E2 occur during ti.

2. Disjunction: $\vee$

$CE(t) = (E1 \vee_{ti} E2)(t) := E1(t_1)$ *OR* $E2(t_2)$, where $t_2 = t$ if $t_1 \le t_2$ or there is no t_1, or $t_1 = t$ if $t_2 < t_1$ or there is no t_2, and $t_1, t_2 \in$ ti. Thus, the disjunction of E1 and E2 given ti means that either E1 or E2, or both E1 and E2 occur during ti.

3. Negation: !

$CE(t) = (!_{ti} E)(t) :=$ NOT $(\exists t' \in ti : E(t'))$. Thus, the negation of E given *ti* means that E does not occur during ti. Also, t is the end point of ti.

4. Loose sequence: «

$CE(t) = (E1 \ll_{ti} E2)(t) := E1(t_1)$ *AND* $E2(t)$, where $t_1 \le t$ and $t_1, t \in$ ti. Thus, loose sequence of E1 and E2 given ti means that E1 occurs before E2 during ti.

5. Strict sequence: >

$CE(t) = ((E1 \ll_{ti} E2) >_{ti} (!_{ti} E3))(t) := E1(t_1)$ *AND* $E2(t)$ *AND* $(NOT\ E3(t_3))$, where $t_1 \le t_3 \le t$ and $t_1, t_3, t \in$ ti. Thus, the strict sequence of E1 and E2 given ti and E3 means that E1 occurs before E2 without E3 occurring between them during ti.

6. Zero or more occurrences: *

$CE(t) = (*_{ti} E)(t) := (E(t_1)$ *AND* $E(t_2)$ *AND* … *AND* $E(t_{m-1})$ *AND* $E(t_m))$ *OR true*, where $t_1 \le t_2 \le \ldots \le t_{m-1} \le t_m$ and $t_m = t$ and $t1, t \in$ ti and $1 \le m$ and $m \in$ N. Thus, 'star' of E given ti means that E either does not occur or occurs one or more times during ti. Note that the goal of the * operator, according to the definition above, is not to search for the validity of the expression $*_{ti}E$, but to keep track of the instances of the event expression E, if there are any. The role of the 'star' operator is associative. It is not used in the declaration of original event expressions, but in the construction of subevents (see Section 4).

7. One or more occurrences: +

$CE(t) = (+_{ti} E)(t) := E(t_1)$ *AND* $E(t_2)$ *AND* … *AND* $E(t_{m-1})$ *AND* $E(t_m)$, where $t_1 \le t_2 \le \ldots \le t_{m-1} \le t_m$ and $t_m = t$ and $t_1, t \in$ ti and $1 \le m$ and $m \in$ N. Thus, 'plus' of E given ti means that E occurs one or more times during ti.

8. Exact number of occurrences: #

$CE(t) = ({}_m\#_{ti} E)(t) := E(t_1)$ *AND* $E(t_2)$ *AND* …*AND* $E(t_{m-1})$ *AND* $E(t_m)$, where $t_1 \le t_2 \le \ldots \le t_{m-1} \le t_m$ and $t_m = t$ and $t_1, t \in$ ti and $m \in$ N. Thus, given m and ti E occurs exactly m times during ti.

9. Maximum number of occurrences: &

$CE(t) = ({}_{m'}\&_{ti} E)(t) := E(t_1)$ *AND* $E(t_2)$ *AND*…*AND* $E(t_{m-1})$ *AND* $E(t_m)$, where

[1] In all the definitions throughout the paper the symbol N represents the set of natural numbers

$t_1 \leq t_2 \leq ... \leq t_{m-1} \leq t_m$ and $t_m = t$ and t_1, t ∈ ti and $m' \geq m$ and $m', m \in$ N. Thus, given m' and ti E occurs at most m' times during ti.

10.Minimum number of occurrences: $

$CE(t) = (_{m'}\$_{st}\ E)(t) := E(t_1)\ AND\ E(t_2)\ AND\ \ldots\ AND\ E(t_{m-1})\ AND\ E(t_m)$, where $t_1 \leq t_2 \leq ... \leq t_{m-1} \leq t_m$ and $t_m = t$ and t_1, t ∈ ti and $m' \leq m$ and $m', m \in$ N. Thus, given m' and ti E occurs at least m' times during ti.

3.2 Condition Language

The composite condition of the ECA rule is a Boolean expression using the operators of the Boolean algebra, i.e., AND, OR, and NOT. These operators take as operands either composite or simple conditions that can involve more than one database. A simple condition is one of the following:

1. a mathematical expression of the form: $f(X_1, X_2, \ldots, X_n)\ mop\ Y$, where *mop* (which stands for mathematical operator) is either $=, \neq, >$ or $<$; $X_1, X_2, \ldots, X_n, Y \in \Re$; $n \in N$; and f is any kind of mathematical function.
2. a logical expression of the form $X == Y$ *or* $X \neq Y$, where *X, Y* are strings.
3. an expression of the form $X \in \{X_1, X_2, \ldots, X_n\}$, where X, X_1, X_2,…, X_n are strings, or X, X_1, X_2,…, $X_n \in \Re$.

The symbol $\Re$ in the definitions above represents the set of real numbers. For all three cases: X_i, for i = 1,..,n, X or Y are either variables or constants. If they are variables, they take values either directly from parameters of the simple event instances of the event part of the rule or indirectly from results of queries that are performed in order to evaluate the condition. The definition of such queries is parametrical and can use parameters from the event part of the rule.

3.3. Action Language

The composite action of the ECA rule is a conjunction of simple or composite actions. A simple action is of the form: DB_i.(Tr). Here, the expression DB_i, for $\forall i \in$ N and $1 \leq i \leq n$, is one of the n-1 acquainted databases to the one on which the rule that contains this specific action is to be installed. The variable Tr comprises the necessary information in order to perform the predefined transaction that is actually the simple database operation (insert, update, delete, retrieve) that we want to perform on DB_i.

3.4 A Further Rule Example

We present a sample rule (from the health care domain of Section 1) emphasizing on the condition part. Suppose that we want to monitor a specific patient in the hospital that has 2 or more epileptic crises in a day. If she is taking a specific drug, <DrugID_value1>, and the dose is more than <constant>, if she is not allergic to drug

<DrugID_value2>, then change the medication to the latter. We assume that the family doctor of the patient is Dr. Davis. The corresponding rule is:

Rule 5:
when $_2$\$$_{(0, 0, 0, 1,-, 0, 0)}$ TGHDB.('insert',('MedRec',
(<OHIP#_value>, _, _, 'epileptic crisis')))
if (<DrugID_value> == <DrugID_value1>) AND (<Dose_value> > <constant>)
AND ({<Allergy_value1> ∈ AllergySet1} ≠
{<Allergy_value2> ∈ AllergySet2})
(*where* TGHDB.('retrieve', 15, {<OHIP#_value>,
<DrugID_value>, <Dose_value>})
and DavisDB.('retrieve',45,{<OHIP#_value>, AllergySet1})
and AllenDB.('retrieve',17,{<DrugID_value2>, AllergySet2}))
then TGHDB.('insert', ('Medication', (<OHIP#_value>,
<currentDate>, <currentTime>, <DrugID_value2>, _)))

Query 15: *Select* DrugID, Dose
From Medication
Where OHIP# = <OHIP#_value>, Date = <current date>,
Time = <most recent time>
Query 45: *Select* Allergy
From Allergies
Where OHIP# = <OHIP#_value>
Query 17: *Select* Allergy
From Allergies
Where DrugID = <DrugID_value2>

In this example, the value (0,0,0,1,_,0,0) denotes the period of one day (the actual set of parameters is (second, minute, hour, day, weekday, month, year). For details on the definition and use of the time interval of an operator see [11]). The 'where' clause in the condition means that we have to perform some queries in order to get the values of the attributes that we want to compare. Thus, we perform query 15 in TGHDB in order to retrieve the drug name and the dose that the patient is receiving currently; we perform the query 45 in DavisDB and the query 17 in AllenDB to retrieve the allergies of the patient and the allergies that the drug <DrugID_value2> can provoke, respectively. The results of queries 45 and 17 are stored in the variables AllergySet1 and AllergySet2 respectively.

4 Processing ECA Rules

When a new rule is created in a database, the latter is responsible for the coordination of the global evaluation of the rule. However, the evaluation of the rule involves other databases with respect to the event, condition and action. This section describes the

general evaluation procedure as well as the two basic algorithms that construct the input of the partial evaluations that take place in databases involved in the rule.

```
NewRule ()
 {   while (creation of a new rule)
     {  -- decompose the rule into subrules and send
              them to the respective databases
        -- run the evaluation procedure for this rule
        -- run the garbage collection for this rule }}
 EvaluationProcedure (Rule newRule)
 {   while (true)
     {  -- wait until a subrule instance of this rule is received
        -- boolean valid = evaluate the original rule with all the accumulated subrule
             instances
             if (valid)
             {if (additional info for the condition is
                          needed)
               { -- request additional info for the
                      evaluation of the condition
                 -- boolean newruleinstance = evaluate   the condition with all the
                      accumulated info
                 if (newruleinstance)
                 { --  execute the action part of the global rule
                   -- throw away the subrule instances and the additional condition informa-
                          tion used to form this rule instance }}
             else
             {  -- execute the action part
              -- throw away the subrule instances
                 used to form this rule instance }}}}
 GarbageCollection (Rule newRule)
 { periodically remove the obsolete subrule instances}
```

Fig. 3. Evaluation Procedure

4.1 Algorithm for the Global Evaluation

Fig. 3 presents a pseudo code of the general algorithm for the global distributed evaluation procedure. The evaluation procedure is as follows: The new rule, to which we refer as global/original rule, is decomposed into subrules that can be evaluated separately in each one of the databases that are involved in the original rule. We produce one such 'subrule' for each database involved in the event part. For those databases that appear both in the event and the condition parts, we add the appropriate condition expression to their corresponding subrule. This condition expression is a sub-expression of the condition part of the global rule. Each subrule is sent to the database it was made for and is evaluated there. The action part of these subrules is to send the subrule instance to the database responsible for the global evaluation of the rule. The global evaluation of the original event and the original condition expression

starts when the local evaluation of one of the created subrules produces an instance that could contribute to the production of an instance of the original-global rule. When such a subrule instance is received in the database where the global rule was created, the procedure tries to find a valid match using the subrule instances that are stored and not used by other rule instances. If a valid match is found, possible additional condition information is requested and gathered. Finally, if the condition instance is true, the action part is executed. The global evaluation procedure aims to minimize the number of messages exchanged among the multidatabases.

The algorithm in Fig. 3 guarantees that the maximum number of messages (that are exchanged among the databases involved in the processing of a rule until the event and the condition expressions are valid and the action part is executed) is the lowest. This number is Max # messages = n+2k+r, where n, k and r is the number of databases involved in the event, condition and action part, respectively (for details see [11]).

4.2 Transformation Algorithm

Before the event expression is decomposed into parts, it is transformed in order to have a form more convenient for decomposition. The goal of the following algorithm is to push the 'not' operators down to the leaves of the tree of the event expression. The evaluation of the 'not' operators is hard and by pushing them down in the event tree, we may be able to eliminate some of them. Moreover, it is possible to give to the rule a form that is more convenient for the evaluation procedure. However, the decomposition of an event expression is possible even without running the transformation algorithm first. The steps of the algorithm are executed multiple times until there are no transformations left to be done. Note that, in case that the operand of the 'not' operator is a composite event, the time interval of the 'not' operator has to be the same as the time interval of the operator of its operand.

transformed = true;
while (transformed)
{ transformed = false;

1. check if there are pairs of consecutive 'not' operators. If there are, eliminate them and set transformed = true.
2. check for the pattern: $!_{ti1}(E1 \wedge_{ti1} E2)$. If found, replace it with the pattern: $(!_{ti1} E1) \vee_{ti2} (!_{ti1} E2)$ and set transformed = true.
3. check for the pattern: $!_{ti1}(E1 \vee_{ti1} E2)$. If found, replace it with the pattern: $(!_{ti1} E1) \wedge_{ti2} (!_{ti1} E2)$ and set transformed = true.
4. check for the pattern: $!_{ti1}(E1 \ll_{ti1} E2)$. If found, replace it with the pattern: $(!_{ti1} E1) \vee_{ti2} (!_{ti1} E2) \vee_{ti2} (E2 \ll_{ti1} E1)$ and set transformed = true.
5. check for the pattern: $!_{ti1}((E1 \ll_{ti1} E2) >_{ti1} (!_{ti1} E3))$. If found, replace it with the pattern: $(!_{ti1} E1) \vee_{ti2} (!_{ti1} E2) \vee_{ti2} (E2 \ll_{ti1} E1) \vee_{ti2} (E1 \ll_{ti1'} E3 \ll_{ti1''} E2)$, where ti1' + ti1'' = ti1 and set transformed = true.
6. check for the pattern: $!_{ti1}(+_{ti1} E)$. If found, replace it with: $!_{ti1} E$ and set transformed = true.

7. check for the pattern ! $_{ti1}$ ($_{m}$ \$ $_{ti1}$ E). If found, replace it with: $_{m-1}$ & $_{ti1}$ E, and set transformed = true.
8. check for the pattern ! $_{ti1}$ ($_{m}$ & $_{ti1}$ E). If found, replace it with: $_{m+1}$ \$ $_{ti1}$ E, and set transformed = true.
9. check for the pattern ! $_{ti1}$ ($_{m}$ # $_{ti1}$ E). If found, replace it with: ($_{m-1}$ & $_{ti1}$ E) $\vee$ $_{ti2}$ ($_{m+1}$ \$ $_{ti1}$ E), and set transformed = true.
10. Change the time interval of 'not' operators that have as operand a composite event to be equal to the time interval of the operator of their operand.

}

In steps 1, 2, 3, 4, 5, 9 of the algorithm above the time interval ti2 of the transformed expressions is of zero length: $|ti2| = 0$. Also, note that the disjunction operator in step 9 serves as an exclusive OR. Step 10 is associative and adapts the time interval of the 'not' operators to the interval in which the non-occurrence of their operand should be evaluated. After the transformation the 'not' operators have as operands only simple events. The initial event expression is semantically equivalent to the transformed expression. For details about the semantic equivalence of the expressions before and after the transformations see [11].

4.3 Decomposition Algorithm

The transformed event expression is broken into subrules, one for each database involved in the event. Each one is sent and installed in the appropriate database. The following algorithm describes the generation of event expressions of these subrules:

For (each database involved in the event expression)

{ -- In the event expression replace the simple events that do not belong to this database with time variables that represent their time of occurrence.

-- Simplify the event expression according to the following rules:

1. Eliminate the nodes of all the unary operators that are left with no operand (i.e. their operand is a time variable).
2. Eliminate the nodes of all the binary operators except 'disjunction' that are left with one operand, that is not a composite event of the 'not' operator, and all the binary operators that are left with no operands. For the disjunction cases, substitute the time variable with 'null'.
3. For all operators except 'disjunction' that have operands changed because of elimination, change their time interval, ti, to the sum of their initial ti and the ti intervals of the 'underlying' eliminated operators, (for exceptions see [11]).
4. If there is a 'strict sequence' operator eliminate the part that has no information relevant to this database. Eliminations are performed according to steps 4a, 4b, 4c, (for details see [11]).
5. Change the 'loose sequence' operators that have a binary operator eliminated in their right operand to a 'conjunction' operator.
6. Merge two successive 'plus' operators to one with time interval equal to the sum of their time intervals.
7. Change the 'not' operators to 'star' ones with the same time interval.

}

In the last step of the algorithm above we substitute the 'not' operators with 'star' in order to just gather the necessary information which will be used to evaluate the 'not' operator in the original event expression during the global evaluation. If we try to evaluate the 'not' operators in the partial evaluations of the local subevents, we may loose instances that could form an instance of the event of the global rule. (For details on the semantic equivalence of the expressions before and after the decomposition and on the change of 'not' operators to 'star' see [11]).

A subevent concerning one database matches at least the same combinations of event instances as those matched by the original event expression. Transforming and decomposing the condition of the rule is done in the obvious way of first order logic.

We consider a sample rule from the health care domain of Section 2. Suppose that until the end of the second day of a patient's admission to the hospital we want a trigger when she has not had either a prescription for a specific drug by Dr. Davis in the past 12 days or less than 2 epileptic crises. Then, if the patient has not been under some other medication in the past 12 days, we would like to give this specific drug to the patient. The ECA rule is:

Rule 6:

when TGHDB.('insert', (Admission, (_,<OHIP#_value >, _, _, _, _))) $\ll_{(0,0,0,2,-,0,0)}$ $(!_{(0,0,0,12,-,0,0)}((_2 \,\&_{(0,0,0,2,-,0,0)}$ TGHDB.('insert', ('MedRec', (OHIP#_value, _, _, 'epileptic crisis')))) $\vee_{(0,0,0,12,-,0,0)}$ DavisDB.('insert', (Prescription, (<OHIP#_value>, <DrugID_value>,_,_,_)))

if (DrugSet1 = ∅) AND (DrugSet2 = ∅)
(*where* TGHDB.('retrieve', 31, {OHIP#_value, DrugSet1}) *and* DavisDB.('retrieve', 25,{OHIP#_value, DrugSet2})

then TGHDB.('insert', ('Medication', (OHIP#_value, <currentDate>, <currentTime>, <DrugID_value>, _)))

Query 31: *Select* DrugID,
From Medication
Where OHIP#=OHIP#_value

Query 25: *Select* DrugID,
From Prescription
Where OHIP# = OHIP#_value, Date > <12 days ago>

Here, the information retrieved by query 31 and query 25 is stored in DrugSet1 and DrugSet2, respectively. The generated subrules for TGHDB and DavisDB are:

TGHDB Subrule:

when TGHDB.('insert', (Admission, (<OHIP#_value >, _, _, _, _))) $\wedge_{(0,0,0,2,-,0,0)}$ $(_2 \,\$_{(0,0,0,2,-,0,0)}$ TGHDB.('insert',('MedRec', (OHIP#_value, _, _, 'epileptic crisis'))))

if (DrugSet1 = ∅) where TGHDB.('retrieve', 31, {OHIP#_value, DrugSet1})

then propagate the subrule instance

DavisDB Subrule:

when $*_{(0,0,0,12,-,0,0)}$ DavisDB.('insert', (Prescription, (<OHIP#_value>, <DrugID_value>, _,_,_)))

if (DrugSet2 = ∅) where DavisDB.('retrieve', 25, {OHIP#_value, DrugSet2})

then propagate the subrule instance

5 Implementation

The presented algorithms have been implemented to compare our distributed rule mechanism with a naïve implementation in terms of communication messages among the databases. In the naïve case the rule evaluation is centralized: all the simple event instances needed for the evaluation of a specific rule are sent individually, as soon as they occur, to the database that is responsible for the global evaluation of that rule. Also, when an event is instantiated, condition information is requested by the responsible database from the appropriate ones, and the condition instances are sent from the latter to the former. Hence, the naïve case has no partial evaluations of subrules: the database where the global rule resides collects all the necessary information.

5.1 Experimental Setup

We experiment on 4 rules that involve databases DB1 and DB2. The rules are declared in DB1:

Rule r1:		**Rule r3**:	
when	DB1.E1 $\wedge_{ti}$ DB2.E2	*when*	DB1.E1 $\wedge_{ti1}$ (DB2.E2 $\wedge_{ti2}$ DB2.E3)
if	*null*	*if*	*null*
then	DB1.A1 AND DB2.A2	then	DB1.A1 AND DB2.A2
Rule r2:		**Rule r4**:	
when	DB1.E1 $\wedge_{ti}$ DB2.E2	*when*	DB1.E1 $\wedge_{ti1}$ (DB2.E2 $\wedge_{ti2}$ DB2.E3)
if	DB2.C1	*if*	DB2.C1
then	DB1.A1 AND DB2.A2	*then*	DB1.A1 AND DB2.A2

In the rules above, DB1.E1 is a simple event in DB1 and DB2.E2, DB2.E3 are simple events in DB2. Also, DB2.C1 is a simple condition evaluated in DB2, Finally, DB1.A1 and DB2.A2 are actions to be executed in DB1 and DB2, respectively.
As we can observe, r1 and r3 have an empty condition part. With them we aim to test how the number of event instances influences the number of communication messages between DB1 and DB2. In rule 1 the event is composed by a simple event from DB1 and a simple event from DB2. Thus, for every event instance of DB2.E2 a new message with this information is sent by DB2 to DB1. In rule 3 there is a simple event from DB1 and a composite (composed of two simple events) event of DB2. Thus, in the case of our rule mechanism, again one message is sent by DB2 to DB1 for every instance of the composite event of DB2. However, in the naïve case one message is sent for every event instance of DB2.E2 or DB2.E3 and the composite event instances of DB2 are formed and validated in DB1. The rules r2 and r4 are similar to r1 and r3, respectively, with a condition part referring to DB2. For these rules, the naïve case has additional messages for the request and the accumulation of condition information.

In the experiments we test the two mechanisms for a total of 10, 100, 200, 300 and 400 maximum *possible* subrule instances. For example, in the first test, for rules r1 and r2 we have 10 event instances 5 instances of DB1.E1 and 5 of DB2.E2. Thus we have exactly 10 subrule instances in total for rule r1 and r2. In fact, the actual number of subrule instances for r1 and r2 is the maximum one. (However, r2 would have a

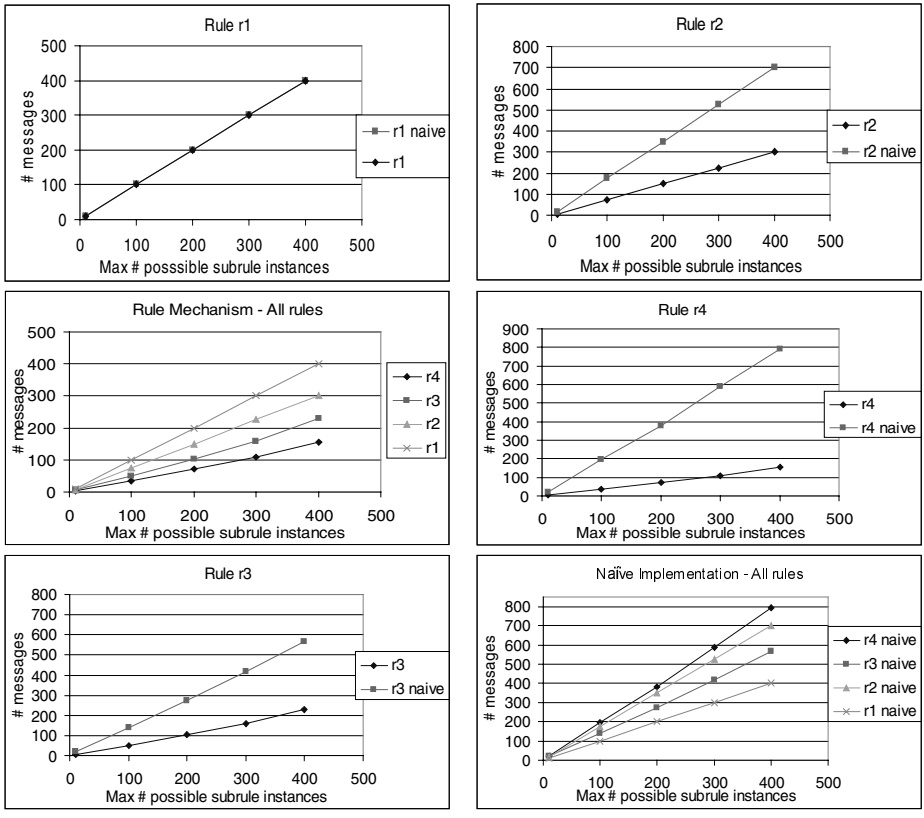

Fig. 4. Experimental results

smaller number if we took into account the form of the condition in order not to communicate subrule instances with an invalid condition). As for rules r3 and r4, the test comprises 15 event instances: 5 of each one of DB1.E1, DB2.E2 and DB2.E3. Nevertheless, the maximum number of possible subrule instances is 10 and depends on the time interval ti2. Consequently, for rules r1 and r2 the maximum number of possible subrule instances coincides with the total number of primitive event instances of DB1 and DB2, whereas, for r3 and r4, given a maximum number of subrule instances d, the total number of event instances involved is 3/2d (d/2 DB1 and d DB2 primitive event instances). The reason why we count the exchanged messages versus the maximum number of *possible* subrule instances and not versus the number of primitive event instances, is because the number of rule instances depends totally on the first, but only partially on the latter. For example, for a total of 100 primitive event instances we could have at most 50 rule instances of rule r1 but only 33 of rule r3, (note that we assume that there are 100/2 instances of events DB1.E1 and DB2.E2 in the first case and 100/3 instances of events DB1.E1, DB2.E2 and DB2.E3, in the second).

For r1 and r2 the time parameter ti is set to a value big enough so that none of the possible composite event instances is rejected by the evaluation procedure because of

a difference in the occurrence time of the component simple event instances bigger than ti. The same holds for the parameter ti1 r3 and r4. Yet, in these rules we play with the parameter ti2 of the composite subevent of DB2 in order to observe how the partial evaluations benefit our rule mechanism (in terms of number of messages) versus the naïve implementation. Therefore, we perform the experiments for 3 values of ti2: 0, 5, 15 seconds. When ti2 equals 0, only simultaneous instances of DB2.E2 and DB2.E3 form subevents of DB2 used by the evaluation procedure for r3 and r4. Also, when ti2 equals 15 almost all the instances of DB2.E2 and DB2.E3 participate in subevents of DB2 that are used to form instances of r3 and r4. Additionally, for r2 and r4 we play with the percentage of valid condition instances of DB2.C1 and perform tests for 100%, 60%, 40%, and 0% of valid condition instances. The number of the latter influences the number of global rule instances.

5.2 Experimental Results

The graphs in Fig. 4 show the average number of messages exchanged between DB1 and DB2 versus the number of maximum possible subrule instances for the experiments on rules r1, r2, r3, r4 performed using the implementation of the proposed rule mechanism and the naïve one. For each one of the four rules, for a specific maximum number of subrule instances, we average the number of messages we get from testing all the possible combinations of the values of the parameters ti2 and the percentage of valid condition instances.

The first four graphs of Fig. 4 compare for each one of the rules the results of the proposed mechanism and the naïve implementation. As we can observe, the two rule mechanisms have exactly the same number of exchanged messages for r1, but have great differences for the rest of the rules. Moreover the difference in the number of exchanged messages is big for r3, bigger for r2 and even bigger for r4. A closer observation shows that this occurs because from experiment r1 to r2 or r3 the number of exchanged messages increases for the naïve case whereas it decreases for the proposed mechanism! The reason is that the proposed rule mechanism takes advantage of the limitations imposed by the form of the rule in order to diminish the number of messages during the evaluation. Moreover, in many cases, the rule mechanism communicates groups of DB2 subrule instances instead of sending them one by one. In case of r4, where there are restrictions both in the event and the condition part, the difference in the number of exchanged messages becomes even bigger.

The last two graphs compare the results of all the experiments for the proposed rule mechanism and the naïve implementation. We can certify by them the previous conclusion: for the naïve case the number of exchanged messages augments as the number of primitive event instances increases and when there is a condition part in the rule (r2 and r4), whereas for the proposed mechanism it decreases when there is a condition part or/ and there is a composite event of DB2.

As we can observe, the experimental results on all four rules are linear, which means that the number of messages is proportional to the maximum number of subrule instances. The linearity is due to the fixed percentages of valid condition instances and the fixed values of ti2 (while the frequency of occurrence of the event instances was

kept reasonable). The linearity of the results would disappear in case of a real workload where the number of valid condition instances and the number of composite subevent instances would not depend on the number of primitive event instances.

6 Related Work

Generally, as stated in the introduction, the ECA functionality has been considered mostly in centralized environments as far as databases are concerned. However, there are attempts to integrate ECA rules in distributed environments such as [18], [6], [7], [1], [4],[14], [5], [9], [15] and [8].

In [4], a framework for characterizing active functionality in terms of dimensions of a distributed ECA rule language and its execution model is introduced. The problem of incorporating ECA rules in a distributed environment is properly defined and broken into parts in order to discover how distribution affects the ECA rules and their execution model. The authors try to reconsider the techniques used in centralized databases and decide if and how they can be adapted in order to fit the distributed dimension of the problem.

In [14] an approach for managing consistency of interdependent data in a multidatabase environment is presented. They introduce the 'data dependency descriptors' (D^3s), objects distributed among the databases that describe the consistency constraints among data and the restoration transactions. Contrary to the D^3s mechanism, our approach is more general by providing d-ECA rules as a global means of expressing data coordination, including various degrees of consistency enforcement. In the near future, we plan to complete the implementation of an application showing how to enforce consistency constraints similar to those enforced by the D^3s mechanism.

An interesting work is presented in [6], where local and global events are detected asynchronously in a distributed environment and are spread together with their parameters to sites interested in them. This work is based on the ADBMS Sentinel and its event specification language Snoop. Both are extended to accommodate composite events from various sources and an appropriate execution model. This work is similar to ours in its mechanism for distributed event detection. However, it differs from our work in two ways. First, recall that the database on which a newly created ECA rule resides coordinates the global evaluation process of the new rule. The event and rule managers on this database act as global event and rule managers. Therefore we do not need a dedicated Global Event Manager in the sense of [6] which could cause a bottleneck in the network. Second, our execution model distributes entire rules to acquainted databases, not only subevents. In fact, the evaluation of the condition together with the respective event is meaningful in order to decide if the event instance is worth of propagation.

A framework with a motivation similar to ours is reported in [5]. Here, however, a distributed service-based mechanism is used to provide an ECA functionality that relies on ontologies known to all participant peers. Instead, our mechanism – as shown in [12] – also uses ingredients known only to acquainted peers such as tables mapping heterogeneous data originating in the acquainted peers [13]. In [9], the idea of adapt-

ing ECA functionality to an application profile is similar to our idea of on-the-fly instantiation of ECA rules. However, we show in [12] that such an instantiation is not sufficient, and needs to be complemented by the automatic generation of rules.

The authors in [18] present a method that uses ECA rules in order to maintain global integrity in a federated database system. Unlike our approach, [18] uses a global database of meta information. Our approach deals with such meta information in a distributed fashion. Moreover, in [18] only primitive event instances without any processing are propagated. The system in [18] is similar to the naïve implementation we used for comparison in the experiments of Section 5.

Generally speaking, most of the projects that try to incorporate ECA functionality in distributed environments in order to manage data [5], [15], [8] have not dealt with optimized distributed evaluation of coordination ECA rules. Also, even though people have talked about distributed conditions tied to specific databases, [16] asynchronous evaluation of condition parts is not considered.

7 Conclusions

We have presented a novel distributed ECA rule mechanism for coordinating data in a multidatabase environment. The mechanism consists of a rule language and an execution model that transforms rules to more easily manageable forms, distributes them to relevant databases, monitors their execution and composes their evaluations. The mechanism is designed in a manner that minimizes the number of messages that need to be exchanged over the network. We have also conducted a preliminary experimental evaluation to compare the implementation with a naïve centralized execution model. Our objective is to use this mechanism to support coordination among databases in a P2P network [2], [3]. Unlike conventional multidatabase systems, the set of participating databases in such a setting is open and keeps changing. So are the rules that define the degree of coordination among the databases. Moreover, we assume that there is no global schema, control, coordination or optimization among peer databases. Also, access to peer databases is strictly local. These assumptions violate many of the premises of traditional (multi)database systems, where a global schema is taken for granted, optimization techniques are founded on global control, and database solutions are arrived at design time and remain static during run time.

In the near future we will present a complete SQL extension based on the rule language we propose. Moreover, we will explore the optimization of the number of communication messages based on which database performs the global evaluation of a rule. The presented execution model used the decoupled coupling mode for both pairs of event-condition and condition-action in order to achieve asynchronous event and condition evaluation, and action execution. However, we intend to follow the idea in [18] that tackles the issue of distributed ECA functionality in the context of advanced transactions by systematically studying coupling modes in a distributed context.

References

1. Arizio, B. Bomitali, M.L. Demarie, A. Limongiello, P.L. Mussa. Managing inter-database dependencies with rules + quasi-transactions. In *3rd Intern. Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pages 34–41, Vienna, 1993.
2. M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. In *SIGMOD Rec.*, Sep. 2003.
3. P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini. Data Management for Peer-to-Peer Computing: A Vision", in proceedings of the *Second Workshop on Databases and the Web*, 2002.
4. G. von Bültzingslöwen et al. ECA Functionality in a Distributed Environment, in *Active Rules in Database Systems*.Springer Verlag, New York, pages 147–175, 1999.
5. M. Cilia, C. Bornhövd, A. P. Buchmann. Moving Active Functionality from Centralized to Open Distributed Heterogeneous Environments, in proceedings of *Cooperative Information Systems, 9th Conference*, pages 195–210, Trento, Italy, September 5–7 2001.
6. S. Chakravarthy, H. Liao. Asynchronous monitoring of events for distributed cooperative environments. In *Intern. Symp. on Coop. Database Sys. for Advanced Applications,* 2001.
7. S. Chawathe, H. Garcia-Molina, and J. Widom. Flexible constraint management for autonomous distributed databases. In *Bulletin of the IEEE Technical Committee on Data Engineering*, 17(2):23–27, 1994.
8. Collet. The NODS Project: Networked Open Database Services, in proceedings of the *Intern. Symposium on Objects and Databases*, pages 153-169, Sophia Antipolis, 2000.
9. Gatziu S., A. Koschel, G. von Bültzingsloewen, H. Fritschi. Unbundling Active Functionality, *SIGMOD Record 27(1): 35–40*, 1998.
10. Zimmer D., Unland R., On the semantics of complex events in active database management systems. In *Proceedings of the 15th International Conference on Data Engineering*, 1999.
11. V. Kantere, *A Rule Mechanism for P2P Data Management,* Tech. Rep. CSRG-469, University of Toronto, 2003.
12. V.Kantere, I.Kiringa, J. Mylopoulos, A. Kementsietsidis, M. Arenas. Coordinating Peer Databases Using ECA Rules, *in proceedings of the International Workshop on Databases, Information Systems and P2P Computing, Berlin*, September 2003.
13. A. Kementsietsidis, M. Arenas, R. J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues, in *Proceedings of the ACM SIGMOD Intern.Conference on Management of Data, San Diego, June 2003.*
14. G. Karabatis, M. Rusinkiewicz, A.Sheth. Aeolos: A System for the Management of Interdependent Data. In A. Elmagarmid, M. Rusinkiewicz, A.Sheth (Eds.), *Management of Heterogeneous and Autonomous Database Systems*, Chapter 8, Morgan Kaufmann 1999.
15. A. Koschel, P.C. Lockermann. Distributed events in active database systems: Letting the genie out of the bottle, in *Data & Knowledge Engineering V. 25*, pages 11–28, 1998.
16. L. Lakshmanan, F. Sadri, and S. Subramanian. SchemaSQL: an Extension to SQL for Multidatabase Interoperability. *ACM Transactions on Database Systems*, 26(4), 2001.
17. W.S. Ng , B.C. Ooi, K.L. Tan, and A.Y. Zhou. PeerDB: a P2P-based System for Distributed Data Sharing. Tech. Report, University of Singapore, 2002.
18. C. Turker and S. Conrad. Towards maintaining integrity in federated databases. In *3rd Basque Intern. Workshop on Information Technology* Biarritz, 1997.