Applying Petri Nets in Active Database Systems

Xiaou Li, Member, IEEE, Joselito M. Medina, Member, IEEE, and Sergio V. Chapa

Abstract—Reactive behavior of active database systems is achieved through the definition of event-condition-action (ECA) rules. Generally, ECA rule representation and processing are separated in the majority of existing active database systems. In this paper, we propose a conditional colored Petri net (CCPN) approach to model and simulate ECA rules. CCPN can not only integrate rule representation and processing in only one model, but is also independent of the actual database system. Furthermore, we have developed a software platform named ECAPNSim, which can generate a CCPN model automatically from a text file of ECA rule description, and communicate with a traditional database system when an event is detected from the database or an action command is generated by the CCPN simulator.

Index Terms—Active database system, event-condition-action (ECA) rules, information system, Petri nets.

I. INTRODUCTION

PETRI nets have proved powerful on modeling, analysis, control, optimization, simulation, and implementation of various engineering systems such as production systems, robotic systems, computer communication networks, software engineering tools, enterprise engineering processes, e-commerce, etc. [1], [2]. The great advantage of Petri nets is that the dynamic behavior of the modeled system can be visually simulated through the flow of tokens within the graphical model. On the other hand, mathematical analysis methods of Petri nets theory permits one to analyze the modeled system formally within the same model. In this paper, we will show a novel application area of Petri nets—active database systems.

In recent years, one of the trends in database technology has focused on extending conventional database management systems (DBMS) to enhance their active functionality and to accommodate more advanced applications [3]. One of these enhancements was extending database systems with powerful active-rule-processing capabilities. In their most general form, active rules (also known as event-condition-action (ECA) rules) consist of three parts: an event causes the rule to be fired; a condition is checked when the rule is fired; and an action is executed when the rule is fired and its condition evaluates to be true [8]–[11].

However, traditional active mechanisms have been designed for centralized systems and are monolithic, thus, making it difficult to extend or adapt them to a new generation of distributed applications [4]. Buchmann pointed out: “the biggest challenge will be to move away from application-specific solutions and to provide application-independent platforms for which the underlying assumptions and the resulting semantics are clearly stated and easily understood by the developers of the application systems” [5]. Additionally, new large-scale applications such as enterprise application integration (EAI), e-commerce, or Intranet applications impose new requirements. In this context, different systems or subsystems provide services with events and data coming from diverse sources, and the execution of actions and evaluation of conditions may be performed on different systems. Furthermore, events, conditions, and actions may not be necessarily directly related to database operations [4]. These lead to the requirement of application-independent approaches, on which only active functionality and some parts/services of a DBMS are used to coordinate the services.

The objective of our research is to achieve such an approach. Our previous papers have reported some fragments of our results [13], [14]. In this paper, we will chain these fragments into an entire paper encompassing our research. First, a conditional colored Petri net (CCPN) is proposed to model ECA rules and simulate their execution. Based on CCPN model, ECA rule execution dynamics can be supervised. Furthermore, CCPN was implemented as an interface called ECA rule Petri net simulator (ECAPNSim) between ECA rule base and their underlying database system. ECAPNSim can not only work independently as an ECA rule simulator, but can also communicate with a conventional database system to realize the active functionality specified by ECA rules.

The rest of the paper is organized as follows. Section II gives an introduction on ECA rules and active database systems. CCPN model is proposed in Section III. Then, Section IV uses examples to show the modeling process of the CCPN approach and ECA rule simulation in the developed ECAPNSim platform. In order to illustrate CCPN and ECAPNSim, we apply it on an information system development. Section V makes a conclusion.

II. ACTIVE DATABASE SYSTEMS

Database systems are beginning to be applied to a range of domains associated with highly complex information processing, even more substantial quantities of data, or highly stringent performance requirements, in which the conventional component environment has proved to be unsatisfactory. This has resulted in a trend in database research toward more functionality required by an application being supported within the database systems itself, giving rise to database systems with more comprehensive facilities for modeling both structural and behavioral aspects of an application. Consequently, active database management system (ADBMS) was introduced in the early 1990s.

Manuscript received December 3, 2004; revised September 28, 2005. This work was supported by Mexican CONACyT under Project 39151-A and Project 38479-B. This paper was recommended by Guest Editor M. Zhou.

The authors are with Departamento de Computación, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV-IPN), Mexico City D.F. 07360, Mexico (e-mail: lix0@cs.cinvestav.mx; jmedina@uaeh.edu.mx; schapa@cs.cinvestav.mx).

Digital Object Identifier 10.1109/TSMCC.2007.897329
A. ECA Rules

An active database management system integrates event-based rule processing with traditional database functionality. Generally, its reactive behavior is achieved by incorporating ECA rules as part of the database. Occurrence of event triggers the evaluation of condition, and if the condition evaluates to be true, then the action is executed [3]. In ADBMS, ECA rules (sometimes called triggers) are defined through two models, i.e., knowledge model and execution model. The knowledge model indicates what can be said about active rules in that system. In contrast to the knowledge model, the execution model determines how a set of rules behaves at runtime [3].

1) Knowledge Model: The knowledge model essentially supports the description of active functionality; it determines when a rule is executed, what condition acts as a filter, and which action is internal or external. The features dealt with in this model often have a direct representation within the syntax of rule language. Thus, each ADBMS has its own knowledge model. But, almost all ADBMSs take ON event IF condition THEN action style, although some systems have other styles like ON-IF-DO, ON-WHEN-DO, etc. In Section II-C, we will give an example of ECA rules. Let us first consider the three parts of ECA rules.

a) Event: An event is something that happens in the database system or its environment at a point in time. For example, an event may be the beginning or end of a data modification corresponding to insert, update, delete commands, such as “update of amount on BONUS,” “insert on SALES,” etc. An event can be primitive or composite. Primitive events correspond to elementary occurrences and can be mapped directly to a point in time determined by something occurring in the database system or its environment. In this paper, we consider the following three types of primitive events:
- **time event** in which the event is raised at some point in time (e.g., the 20th of June 2005 at 14:00);
- **operation event** in which the event is raised by an operation on some piece of structure (e.g., insert a table, update an attribute, delete a table);
- **external event** in which the event is raised by a happening outside the database (e.g., the temperature reading goes above 30°C).

Composite events are defined as combinations of primitive or composite events using a set of event constructors. In this paper, we consider the following eight types of compositions:
- **Disjunction** OR(e1, e2): This composite event occurs when either event e1 or event e2 has occurred;
- **Conjunction** AND(e1, e2): This composite event occurs when both event e1 and event e2 have occurred in any order;
- **Sequence** SEQ(e1, e2): This composite event occurs when event e1 occurs before event e2;
- **Simultaneous** SIM(e1, e2): This composite event occurs when events e1 and e2 occur at the same time;
- **History** TIMES(n, e1, Int): This composite event is signaled when event e1 occurs n times during the time interval Int;
- **Negation** NOT(e1, Int): This composite event detects the nonoccurrence of the event e1 in time interval Int;
- **Closure** CLOSURE(e1, Int): This composite event is raised only when the first time event e1 is signaled, regardless of later occurrences of e1 in the time interval Int;
- **Any ANY(m, e1, e2, . . . , en):** This composite event is raised when m of n different events e1, e2, . . . , en have occurred, where m ≤ n.

Composite events can be mapped to a point in time based on the information about their component events. For example, event part of a rule may be on detection of both event e1 and e2 (Conjunction(e1, e2)), or on detection of a sequence of event e1 and e2 (Sequence(e1, e2)). These kinds of event detections are much more complicated than detecting a simple primitive event, since one only needs to identify the two events e1 and e2 but also need to evaluate their occurrence order. For some composite events (such as closure, history, etc.), time information of the component events and frequency of an event occurrence during a time interval also needs to be evaluated [3], [13].

b) Condition and action: When an event is detected, we say that the rule is triggered. When the rule is triggered, it may be activated or not depending on the evaluation result of the condition. The condition part of the ECA rules is like a filter. In ECA rules, the condition is generally optional. When no condition is given for an ECA rule, an event-action rule results. In this case, the condition is evaluated true forever. A condition can be a predicate on database state or query, with one or more application procedure calls (or method invocations), e.g., “BONUS_amount > 100.”

An action can be any case, which will update database state or make some special operations such as data modification and retrieval in relational DBMSs, transaction operations (for example commit and abort), method invocation in OODBMSs, procedure calls in relational DBMSs, rule operations, etc. For example, “update EMP set rank = update.rank + 1 where emp_id = update.emp_id.” The fact that the action may execute data operation may cause the occurrence of other events. This leads to the cascaded triggering of rules.

2) Execution Model: The execution model of ECA rules specifies how a set of rules is treated at runtime, and it is closely related to aspects of the underlying DBMS (e.g., data model, transaction management), including the relationships between triggering and triggered transactions in terms of commit and abort dependencies as well as the semantics of rule execution with respect to concurrency control and recovery. Fig. 1 shows a simple architectural view of the execution of ECA rules. A more detailed explanation on execution model is given in [3] and [6].

B. Existing Active Database Systems

To date, the majority of existing active databases have their own rule (or trigger) languages, e.g., Starburst [8], Postgres [10], Ariel [9], SQL-3, HiPAC, EXACT, NAOS, Chimera [11], Ode, SAMOS [7], Sentinel, REACH, etc. This means that an ECA rule defined in one system (e.g., Postgres) cannot be recognized and executed by another (e.g., SAMOS). As we
mentioned in the introduction that this situation makes it difficult or impossible to handle large complex distributed systems, a unified language that can be recognized and handled by all database systems is urgently required. Motivated by this objective, we proposed the CCPN model and developed ECAPNSim [13], which can be recognized by a lot of database systems. Furthermore, we have experimented ECAPNSim successfully on Oracle, Postgres, Progress, Foxpro, Primebase, and Access database systems. The results showed that CCPN could work as a unified ECA rule specification language for a lot of SQL-based relational database systems. In our CCPN approach, ECA rules are described in the general ON-IF-THEN style.

C. Example

Let us see an example of ECA rules in an active database system.

Example 1: The database is about the personal management of a company, which is defined by the following three tables and a rule base:

EMP(emp_id, name, rank, salary)
BONUS(emp_id, amount)
SALES(emp_id, month, number)

The rule base contains four rules, which are described as follows.

Rule 1: When an employee’s bonus is increased by more than 100, then the employee’s rank is increased by 1.

Rule 2: When an employee’s rank is updated and then increased by 1, then the employee’s bonus is increased by 10 times the new rank.

Rule 3: When an employee posts sales greater than 50 and his rank is lower than 15, his bonus is decreased by 100.

Rule 4: When an employee’s rank reaches 15, then his salary increases by 10%. In order to see the rule relation clearly, we analyze the event, condition, and action parts of these rules and rewrite them in ON-IF-THEN style as follows.

Rule 1: ON update of amount on BONUS
IF BONUS_amount >100
THEN update EMP set rank = update.rank+1
where emp_id = update.emp_id

Rule 2: ON update of rank on EMP
IF EMP_rank >5
THEN update EMP set amount = old.amount+rank*10 where emp_id = update.emp_id.

Rule 3: ON insert on SALES
IF EMP_rank >5
IF SALES_number >50 & EMP.rank <15

Rule 4: ON update of rank on EMP
IF EMP_rank = 15
THEN update EMP set salary = old.salary*1.1
where emp_id = insert.emp_id.

It is not difficult to observe that there are four operation events in the rule base. They are:
e1: update_BOUNS_amount;
e2: update_EMP_rank;
e3: update_EMP_salary;
e4: insert_SALES.

All rules have a condition part, and all the actions in these rules are operation events.

III. CCPN

To date, not much research can be found on the use of Petri nets in active database systems. To the best of our knowledge, SAMOS is the unique existing active database system that uses Petri nets as event detector [15]. In SAMOS, a colored-Petri-net-like model was defined for composite-event specification. In order to achieve a correct model of a composite event, many additional places and transitions have to be used to represent the temporal information. Although [16] is about network management systems rather than active database systems, it is another interesting relational research. In [16], colored Petri nets (CPNs) are used to specify the dependence between events. But their models are much larger than those in [15], since they have to put extra structure to express temporal relations between primitive events. Our CCPN model will overcome the disadvantage of using a redundant structure to specify a temporal relation between primitive events since this information can be considered as a condition on transitions.

A. Informal Introduction

First, let us see how to represent ECA rules with CPN.

In order to express clearly and exactly the three components of an ECA rule, we need to analyze ECA rule execution again. When an ECA rule is executed, event detection likes a logic process to get a result 1 or 0 (corresponding to whether the event is detected or not). Therefore, it is more convenient to model it as a place, and the event content may be modeled as a “color.” If the detection result is 1, then a colored token is deposited into this place. Additionally, an action of a rule may be an event of another rule (note that the conjunction of the set of actions and events is not empty), modeling actions as places also obey human cognition.

The condition part of an ECA rule will be evaluated after event detection; thus, it is like a guard for transition. Here, we call this transition a conditional transition to emphasize the importance of conditions. If there are tokens in each input place, then the transition is enabled, and if its condition is evaluated...
true, then the transition is fireable. The above idea may be explained intuitively as shown in Fig. 2(a) and (b). In Fig. 2(a), rule is mapped into a transition, event and action are mapped into input and output places of the transition. Finally, condition is attached to the transition as a guard. In Fig. 2(b), event is an AND composite event, so both event 1 and event 2 are mapped into input places of the rule transition.

As shown in Fig. 2(a), a simple ECA rule with only one primitive event may be modeled as a CPN. But, for complicated composite events, basic elements of CPN are not sufficient. For example, composite events sequence (SEQ(e1,e2)), times (TIMES(e1, Int)) cannot be modeled by an ordinary CPN directly. In [16], a very complex structure is used to represent a sequence composite event. However, this will result in an extremely huge CPN model, which will be inconvenient for large rule-base development, and will be difficult to be implemented and managed.

For the above reasons, we defined new elements on CPN especially to characterize ECA rules features. Fig. 3 shows a list of all CCPN elements that we have defined. In CCPN, places are classified into primitive, virtual, copy, and composite places. Primitive and composite places map primitive and composite events; a virtual place is used for storing conjunction and disjunction composite events; copy places are used when one event triggers more than one rule—in such a case, we make copies of the event so that all rules triggered by this event can be enabled when the event is detected. Transitions are classified into rule, composite, time, and copy transitions. Rule transitions map ECA rule; composite transitions are used to generate composite events from primitive events; time transitions are used to deposit a token to a time event (one of the three types of primitive events); and copy transitions are used to generate copies of events (may be primitive or composite events). Arcs are classified into normal arcs and inhibitor arcs. Inhibitor arcs are used for negation composite-event generation. As in CPN, tokens in CCPN take information too. But, CCPN does not use arc expressions, which are very important in CPN. We use condition on transition to restrict transition firing.

B. Definition of CCPN

Our CCPN model is similar to a CPN [12], but it is not a CPN.

Before starting to define CCPN, we review some notations generally used in Petri net literatures.

- A multiset \( m \), over a nonempty set \( S \), is a function \( m \in [S \rightarrow N] \), which we represent as a formal sum \( \sum_{s \in S} m(s)s \).

By \( S_{MS} \) we denote the set of all multisets over \( S \). The nonnegative integers \( \{m(s) \mid s \in S\} \) are the coefficients of the multiset. \( s \in m \) if \( m(s) \neq 0 \).

- The element of type \( T \). The set of all elements in \( T \) is denoted by the type name \( T \) itself.

- \( B \) is used to denote the boolean type (containing the elements \{false, true\} and having the standard operations).

- The type of a variable \( v \) is denoted by \( Type(v) \). the domain of \( Type(v) \) maybe a set of expressions.
The type of an expression $expr$ is denoted by $Type(expr)$.

A binding of a set of variables $V$, associating with each variable $v \in V$ an element $b(v) \in Type(v)$.

The value obtained by evaluating an expression $expr$ in a binding $b$ is denoted by $expr(b)$. $Var(expr)$ is required to be a subset of the variables of $b$, and the evaluation is performed by substituting for each variable $v \in Var(expr)$ the value $b(v) \in Type(v)$ determined by the binding.

The notation $Type(v)$ is extended to $Type(A) = \{Type(v) \mid v \in A\}$ where $A$ is a set of variables.

**Definition 1 (CCPN):** CCPN is an 11-tuple $CCPN = \{\Sigma, P, T, A, N, C, Con, Action, D, \tau, I\}$ where

1) $\Sigma$ is a finite set of nonempty types, called color sets.
2) $P$ is a finite set of places. For better graphical representation, $P$ is divided into four subsets, i.e.,
   \[ P = P_{pri} \cup P_{com} \cup P_{vir} \cup P_{cop} \]
   where $P_{pri}$, $P_{com}$, $P_{vir}$, and $P_{cop}$ are sets of primitive, composite, virtual, and copy places.
3) $T$ is a finite set of transitions. $T$ is divided into three subsets, i.e.,
   \[ T = T_{rule} \cup T_{copy} \cup T_{comp} \cup T_{time} \]
   where $T_{rule}$, $T_{copy}$, $T_{comp}$, and $T_{time}$ are sets of rule, copy, composite, and time transitions.
4) $A$ is a finite set of arcs such that $P \cap T = P \cap A = T \cap A = \Phi$.
   \[ A = A_{inh} \cup A_{nor} \]
   where $A_{inh}$ and $A_{nor}$ represent the sets of inhibitor and normal arcs, respectively.
5) $N$ is a node function. It is defined from $A$ to $P \times T \cup T \times P$.
6) $C$ is a color function. It is defined from $P$ to $\Sigma$.
7) $Con$ is a condition function. It is defined from either $T_{rule}$ or $T_{comp}$ into expressions such that
   - $\forall t \in T_{rule} : [Type(Con(t)) = B]$ where the $Con$ function evaluates the rule condition.
   - $\forall t \in T_{comp} : [Type(Con(t)) = B]$ where the $Con$ function evaluates the temporal condition.
8) $Action$ is an action function. It is defined from $T_{rule}$ into expressions such that
   \[ \forall t \in T_{rule} : p \in t : [Type(Action(t)) = C(p)_{MS}] \]
9) $D$ is a time interval function. It is defined from $T_{comp}$ to a time interval $[d_1, d_2]$, where $t \in T_{comp}$, and $d_1$ and $d_2$ are the initial and final time point of the interval, respectively.
10) $\tau$ is a timestamp function. It is defined from $M(p)$ to $\{0\} \cup \mathbb{R}^+$, which assign each token in place $p$ a timestamp corresponding to natural clock with the form $year : month : day − hour : minute : second$. For example, a token has timestamp 2005 : 01 : 15 − 11 : 16 : 46.
11) $I$ is an initialization function. It is defined from $P$ into closed expressions such that
   \[ \forall p \in P : [Type(I(p)) = (C(p)_{MS}, \tau(C(p)_{MS}))] \]

Graphically, the above elements are represented as Fig. 3.

### C. Transition Firing

In CCPN, we define transition firing by two steps:

1) verify if a transition is enabled;
2) verify if it is fireable according to the condition evaluation result.

Before defining rule transitions, we present the token element concept in CCPN, which is the basic unit of CCPN state transition. All transition firings are based on token elements.

**Definition 2 (Token Element):** In CCPN, a token element is a 4-tuple $(p, c, data, timestamp)$ where $p \in P$ is the place, $c \in C(p)$ is the color of the token, $data$ is the color information corresponding to the color structure of $c$, and $timestamp$ specifies the natural time when the token is deposited into place $p$. The set of all token elements is denoted by $TE$. A marking is a multiset over $TE$. The initial marking $M_0$ is the marking, which is obtained by evaluating the initialization expressions

\[ \forall (p, c, data, 0) \in TE : M_0(p, c, data, 0) = (I(p))(c, \tau) \]

For example, a token element at place $p_1$ may be

$(p_1, \text{update of amount on BONUS, set amount = old.amount + rank * 10 where emp_id = update.emp_id, 2005 : 01 : 15 − 11 : 16 : 46})$.

Generally, a place $p$ has tokens of the same color $c$ although they may have different $data$ and $timestamp$. However, if $p$ is a virtual place, it is possible that various colors (i.e., different $c$) are in place $p$. In another words, if $p \in P_{vir}$, then tokens in $p$ may possess various colors that take from its antecedent places. If $p \in P_{pri} \cup P_{com} \cup P_{cop}$, then all tokens in $p$ have the same color. For some composite events, we need to know that tokens in place $p$ are of the same or different colors. Therefore, we introduce a notation $N_{Color}(p)$, which represents the number of colors in place $p$. If $p \in P_{vir}$, then it is possible that $N_{Color}(p) \geq 1$. If $p \in P_{pri} \cup P_{com} \cup P_{cop}$, then $N_{Color}(p) \leq 1$.

In CCPN, we define transition enabling similarly by evaluating if there are tokens in each of the input places of the transition, with the exception that for a transition with inhibitor arcs, it is defined by evaluating if there is no token in the input place of the transition.

**Definition 3 (Transition Enabling):** A transition $t \in T$ is enabled at marking $M$ if

1) $\forall p \in t : |M(p)| = 0$, type($t$) = Negation.
2) $\forall p \in t : |M(p)| \geq 1$, else.

A transition is enabled means that its input places are ready for firing. Then, we need to evaluate if the transition itself is ready for firing, which corresponds to the condition evaluation. If both the input places and the transition are ready for firing, we say that the transition is “fireable.” In CCPN, there are two types of conditions: 1) rule-type conditions, which are predicates on the database state and 2) temporal conditions, which are used to detect composite events. For a rule transition, we need to evaluate if the condition is completed by the current database state; for a composite transition, we need to evaluate if the time interval covers the token’s timestamp. Additionally, for a copy transition, if it is enabled it can immediately fire.

**Definition 4 (Transition Firing):** A transition $t \in T$ fires if

1) $\forall t \in T_{rule}, t$ is enabled and $Type(Con(t)) = true$. 
Definition 5 (Enabled Function): When transition \( t \in T \) is enabled, enabled function \( C_{\text{enabled}} \) is defined from \( P \times T \) into expressions such that

\[
\forall t \in T, \ p \in t : \ [\text{Type}(C_{\text{enabled}}(p, t))] = C(p)_{MS}.
\]

When transition \( t \) is enabled, the enabled function \( C_{\text{enabled}} \) specifies what token elements transition \( t \) is enabled about. Then, if the transition is composite type, we need to define the composite color after firing.

Definition 6: When a transition \( t \in T_{\text{comp}} \) is enabled, composition function \( C_{\text{composition}} \) is defined from \( T \times P \) into expressions such that

\[
\text{Type}(C_{\text{composition}}(t, p_{\text{out}})) = \text{Type}(t)(C(p_{\text{in}})_{MS})
\]

where \( p_{\text{in}} \in t \), and \( p_{\text{out}} \in t \).

Similar to CPN firing mechanism, after transition firing, tokens in input places of the firing transition will be removed, and the generated token elements will be deposited to its output places according to the definition of the places and the transition. Similarly, we divide the process into three cases.

Case 1: When transition \( t \) is a rule-type transition, the token element in the input places of transition \( t \) will be removed, and a new token element corresponding to the action of the rule will be deposited to its output place.

Case 2: When transition \( t \) is a copy-type transition, we simply remove the token element in the input place and deposit a copy of the same token element to all output places.

Case 3: When transition \( t \) is a composite-type transition, we consider two situations.

1) The composition type of \( t \) is \( \text{Negation} \), the input place of \( t \) will keep the same state, and a new token with negation information of the input place will be deposited into its output place.

2) Otherwise, a token element with enabled color will be removed from the input places of \( t \), and a token element with composed color information will be deposited into the output place of \( t \).

A formal definition is as follows.

Definition 7 (Token Transition 1): When transition \( t \) is enabled at marking \( M_1 \) and is evaluated to be firable, it fires, marking \( M_1 \) changes to marking \( M_2 \), defined by

1) if \( t \in T_{\text{rule}} \), \( \forall p \in P \)

\[
M_2(p) = M_1(p) - C_{\text{enabled}}(p, t) + \text{Action}(t, p);
\]

2) if \( t \in T_{\text{copy}} \), \( p_1 \in t \), \( p_2 \in t \)

\[
M_2(p_1) = M_1(p_1) - C_{\text{enabled}}(p_1, t)
M_2(p_2) = M_1(p_2) + C_{\text{composition}}(p_1, t);
\]

As defined in Definition 2, each token element has a timestamp. After transition firing, token timestamps are modified according to how the new token is created. Similarly, the three types of transitions need to be considered separately.

1) When a rule transition \( t \in T_{\text{rule}} \) fires, a new token is created by the \( \text{Action} \) function after the transition firing, and the timestamp assigned to the new token takes the system current time, i.e., \( \text{token}(p, c, \text{data}, \text{currentTime}()) \), where \( p \in t^* \).

2) When a copy transition \( t \in T_{\text{copy}} \) fires, tokens in \( p_1 \in t^* \) are replicated to every place \( p_j \in t^* \) with the same timestamps.

3) When a composite transition \( t \in T_{\text{comp}} \) fires, token timestamps are modified in two ways.

- If a time interval \( D \) is specified on \( t \) (corresponding to composite events \( \text{Negation}, \text{Closure}, \) and \( \text{History} \)), the timestamp of the new token created to denote the composite event is the end point of the time interval \( D = [d_1, d_2] \), i.e., the new token is \( \text{token}(p, c, \text{data}, d_2) \), where \( p \in t^* \).
- Otherwise, the new token takes the timestamp of the last constituent event (or token) occurred, i.e., \( \text{token}(p, c, \text{data}, \text{min}(\tau(t^*))) \), where \( p \in t^* \).

Sometimes, an enabled transition cannot fire for not satisfying its conditions. In this case, we just remove the enabled tokens from its input places, while the other places keep the same state as before. In this case, no new tokens are created.

Definition 8 (Token Transition 2): When a transition \( t \in T_{\text{rule}} \cup T_{\text{comp}} \) is enabled at a marking \( M_1 \), but does not fire because \( \text{Type}(\text{Con}(t)) = \text{false} \), marking change still exists, and a new marking \( M_2 \) is defined as

\[
\forall p \in P : M_2(p) = M_1(p) - C_{\text{enabled}}(p, t).
\]

IV. MODELING AND SIMULATION WITH CCPN

A. Modeling ECA Rules With CCPN

In an ECA rule base, rules are interrelated by their common events when an event participates in two or more rules. In Example 1, the event “update of rank on EMP” is both an event of Rule 2 and Rule 4, and an action of Rule 1 and Rule 3. Fig. 4 shows the CCPN model of the rule base in Example 1 in
Section II. The event “update of rank on EMP” is modeled as place E1, and E1 is copied as two CopyOf_E1s to trigger Rule 2 and Rule 4. Table I gives a detailed modeling information on events and actions. Table II gives rule modeling.

Generally, the modeling process can be described in the following steps.

1) Model primitive events and actions as primitive input and output places.
2) Construct composite transitions to produce composite events.
3) Introduce a copy transition when one event contributes to more than one rule.
4) Model ECA rules as a rule transition with conditions attached to the transition.
5) Unite all common places.

CCPN models have been developed for the above composite events too. However, composite events detection is much more complicated than primitive events. We have modeled eight basic composite events mentioned in Section II (see Figs. 5–8. Thus, a composite event can be represented by a token in a composite (or virtual) place $e_c$, and also composite event detection are converted into composite transition firing. In this way, a CCPN can simulate ECA rule execution as well as detect composite events in ECA rules. We have developed an algorithm to convert ECA rules in text into a CCPN model automatically (see [13] for details).

Of course, rules in Example 1 are very simple; there are no composite events. Here, we introduce a more complicated example in which composite events are included.

Example 2: Here, an example is illustrated with three active rules, where composite events are the event part of ECA rules. The database consists of the following two tables:

| EMPLOYEE(emp_id, name, position, salary, start_date) |
| SALES(emp_id, amount, sales_date) |

Now, we have some operation rules on the database, which are generally in the text as follows.
Rule 1: When an employee’s record is added into the database or his salary is updated, and if his new salary is more than the manager’s salary, then the employee’s salary will be decreased, and the new salary will be the 20% of the manager’s salary.

Rule 2: When an employee is new in the company and gets high sales on his first work day, then his salary will be increased by 10%.

Rule 3: When the records were not added into the sales table on a working day, then notify the manager about the sales.

After analyzing these rules, we rewrite the above rules into ON-IF-THEN style. For context convenience, we find all the primitive events (or actions) in these rules. First, there are totally four events: insert employee, update employee’s salary, insert sales, and notify the manager (an external event). We will use e0, e1, e2, and e9 to represent them, i.e.,

- e0: insert EMPLOYEE;
- e1: update EMPLOYEE’s salary;
- e2: insert SALES;
- e9: notify the manager.

Then, Rule 1 may be triggered by two events e0 and e1. Furthermore, as a composite event, e0 and e1 are OR related by Rule 1, i.e., the composite event of rule 1 is OR(e0, e1). Rule 2 may be triggered by two events e0 and e1. However, both e0 and e1 are needed to trigger Rule 2; the composition is a conjunction of the two events, i.e., AND(e0, e1). Rule 3 has a negation event NOT(e2). Now, the three rules are redescribed as follows.

Rule 1: ON OR(e0,e1) IF EMPLOYEE.salary > (select salary from EMPLOYEE where position = ‘manager’) THEN EMPLOYEE.salary = manager.salary * 0.20.

Rule 2: ON AND(e0,e2) IF true THEN EMPLOYEE.salary = EMPLOYEE.salary * 1.10.

Rule 3: ON NOT(e2) IF true THEN notify the manager.

There are three composite events in the above rule base. They are OR(e0,e1), AND(e0,e1), and NOT(e2). ECAPNSim will produce the three rules into a CCPN automatically as shown in Fig. 9. In Fig. 9, places named with letter “E” represents primitive events, and places named with “EC” denotes composite events. Transitions T2 and T3 are composite transitions corresponding to the event part of Rule 2 and Rule 3, and transitions T5, T6, and T7 are rule transitions corresponding to Rule 1, Rule 2, and Rule 3. Places E9 and E10 represent action parts of the rules. Table III and IV specify all the components of Fig. 9 in detail.

Through this CCPN model, it is not difficult to observe that the number of places is not as many as the sum of the number of events and actions. This is because many events and actions are the same database operations.
TABLE III
PLACE SPECIFICATION OF FIG. 9

<table>
<thead>
<tr>
<th>place [type]</th>
<th>event(or action)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0 [primitive]</td>
<td>e0</td>
</tr>
<tr>
<td>E1 [primitive]</td>
<td>e1</td>
</tr>
<tr>
<td>E2 [primitive]</td>
<td>e2</td>
</tr>
<tr>
<td>EC3 [virtual]</td>
<td>OR(e0, e1)</td>
</tr>
<tr>
<td>EC4 [virtual]</td>
<td>AND(e0, e2)</td>
</tr>
<tr>
<td>EC5 [composite]</td>
<td>NOT(e2)</td>
</tr>
<tr>
<td>E9 [primitive]</td>
<td>e9</td>
</tr>
</tbody>
</table>

TABLE IV
TRANSITION SPECIFICATION OF FIG. 9

<table>
<thead>
<tr>
<th>transition [type]</th>
<th>condition</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0 [copy]</td>
<td></td>
<td>E0</td>
<td>EC3</td>
</tr>
<tr>
<td>T1 [copy]</td>
<td></td>
<td>E1</td>
<td>EC3</td>
</tr>
<tr>
<td>T2 [composite]</td>
<td>not specified</td>
<td>C_E0</td>
<td>EC4</td>
</tr>
<tr>
<td>T3 [composite]</td>
<td>not specified</td>
<td>C_E2</td>
<td>EC5</td>
</tr>
<tr>
<td>T4 [copy]</td>
<td></td>
<td>E2</td>
<td>2 C_E2</td>
</tr>
<tr>
<td>T5 [Rule 1]</td>
<td>cond_Rule 1*</td>
<td>EC3</td>
<td>E1</td>
</tr>
<tr>
<td>T6 [Rule 2]</td>
<td>no condition</td>
<td>EC4</td>
<td>E1</td>
</tr>
<tr>
<td>T7 [Rule 3]</td>
<td>no condition</td>
<td>EC5</td>
<td>E9</td>
</tr>
</tbody>
</table>

B. ECAPNSim: ECA Rule Simulator

Based on the CCPN model, a software platform named ECAPNSim has been developed. ECAPNSim can generate a CCPN model automatically from a text file of ECA rules description, and return a command to the database according to the action information after transition firing. ECAPNSim is implemented in Java under MAC OS X Server. It is used as a layer between database users and a traditional relational database system. ECAPNSim provides reactive behavior to traditional databases, i.e., a traditional database can work like an active one through ECAPNSim, without special ECA rule syntax and semantics development. For instance, a Postgres version 7.1 can work like an active Postgres by connecting with CCPN through JDBC driver instead of utilizing the trigger semantics.

Fig. 10 shows the buildtime architecture of ECAPNSim. ECAPNSim was developed with several buildtime tools, which support the activities performed during rule specification such as ECA-CCPN converter, incidence matrix generator, termination analysis, interrelation complexity analysis, etc. Fig. 11 shows the runtime architecture of ECAPNSim. At runtime, ECAPNSim can be performed in simulation mode or real mode. In simulation mode, events happen randomly, and the CCPN model is activated by detecting these random events. In real mode, first, ECAPNSim detects events from database state modifications and then executes the CCPN simulator; second, if some rules are triggered by the detected events, then ECAPNSim sends the actions specified in the fired rules to the database; third, the database system makes an operation according to the action coming from ECAPNSim. Here, we give an example to show the ECAPNSim.

Example 3: An enterprise consists of departments that work in projects, and every project has employees that belong to a department. We define the following four tables to build the enterprise database:

- Table EMP is used to store information about the employees, where
  - TheEmp key used to distinguish an employee;
  - Name employee’s name;
  - Salary employee’s salary;
  - Manager head of the project where the employee works in;
  - HisDep department where the employee is assigned;
  - Bonus extra bonus for employees.

- Table DEP is used to store information about a department of the enterprise, where
  - TheDep key of the table to distinguish a department
  - Name department name
  - Budget amount of money assigned to a department
  - Prod production of a department.

- Table PROJECT is used to store information about a project that is being developed by the enterprise, where
  - TheProj key used to distinguish a project
  - Name project name
  - Budget amount of money assigned to a project
Manager head of the project.

Table WORK_IN is used to relate employees that work in a certain project, where

TheProj project key
TheEmp employee’s key.

This database schema needs an active behavior in order to keep a consistency in the database information, which can be achieved by using the following ECA rules.

Rule 000: Whenever a department is deleted from the database, we need to remove all the employees that worked in that department.

Rule 001: Whenever a project is finished or cancelled, it should be removed from the database. And if it occurs, then all the records related to the project need to be deleted from table WORK_IN.

Rule 002: Whenever a record is deleted from table WORK_IN and the corresponding employee exists in table EMP, then he needs to be deleted from table EMP.

Rule 003: When an employee is deleted from table EMP and he exists in table WORK_IN, then the corresponding record should be deleted from table WORK_IN.

Rule 004: When either an employee is inserted in table EMP or the employee’s salary is updated, and if his salary is more than $15,000.00, then the new employee should be deleted from table EMP.

Rule 005: When an employee is inserted in table EMP and the production for his department increases to more than 90 points, then the new employee is awarded with a bonus of $100.00.

Rule 006: When a new employee is inserted in table EMP, but he is not added to a project on his first working day, then he is added to the project with the least number of employees.

Now, we analyze the events in the above rules. Primitive events mentioned in the above rules are as follows:

e0 : delete_DEP;
e1 : delete_PROJECT;
e2 : delete_WORK_IN;
e3 : delete_EMP;
e4 : insert_EMP;
e5 : update_EMP_Salary;
e7 : update_DEP_Prod;
e9 : insert_WORK_IN;
e14: update_EMP_Bonus.

And the composite events are:

OR(e4:e5);
AND(e4:e7);
NOT(e9);
SEQ(e4:e10).

Then, we create an ECA rule base from the above rules where each rule is rewritten in the ON-IF-THEN style.

Rule 000: ON e0
IF true
THEN delete from EMP where EMP.HisDep = DEP.TheDep.

Rule 001: ON e1
IF true
THEN delete from WORK_IN where PROJECT.TheProj = WORK_IN.TheProj.

Rule 002: ON e2
IF exists (select from EMP where TheEmp = WORK_IN.TheEmp)
THEN delete from EMP where WORK_IN.TheEMP = EMP.TheEmp.

Rule 003: ON e3
IF exists (select from WORK_IN where TheEmp = EMP.TheEmp)
THEN delete from WORK_IN where EMP.TheEmp = WORK_IN.TheEmp.

Rule 004: ON e6
IF EMP.Salary > 15000
Rule 005: ON e8
IF DEP.Prod > 90 & EMP.HisDep = DEP.TheDep
THEN update EMP set value Bonus = 100 where EMP.TheEmp = new.EMP.TheEmp.

Rule 006: ON e11
IF true
THEN insert into WORK_IN values (newProject, EMP.TheEmp).

We save the above ECA rules as a text file named Enterprise1.eca. Then, the ECA-CCPN converter reads it and converts it into a CCPN as shown in Fig. 12. The place and transition specifications of the CCPN are interpreted as Tables V and VI. Once we have the CCPN model of the rule base, all the build-time tools can be used. Furthermore, the database management system, where the enterprise data were stored, can work as an active database system if it is connected with ECAPNSim by its JDBC driver.
We believe CCPN would be very promising in the near future. As of the date, there is no formal model to specify these compositions. In many information systems, composite events and reactive activities exist widely and popularly. However, till now, there is no formal model to specify these compositions. We believe CCPN would be very promising in the near future.

V. CONCLUSION

This paper introduces a new application area for Petri nets: active database systems. A high-level Petri net model CCPN is proposed for ECA rule modeling and simulation. Its implementation ECAPNSim can be used as an engine for active database systems.

In the future, we are trying to apply CCPN to other areas where ECA rules and composite events are used such as workflow management systems, multiagent systems, etc. We emphasize that the advantage of CCPN is composite events and ECA rule modeling. In many information systems, composite events and reactive activities exist widely and popularly. However, till date, there is no formal model to specify these compositions. We believe CCPN would be very promising in the near future.

TABLE V
SPECIFICATION ON THE PLACES OF FIG. 12

<table>
<thead>
<tr>
<th>place [type]</th>
<th>event/or action</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0 [primitive]</td>
<td>e0</td>
</tr>
<tr>
<td>E1 [primitive]</td>
<td>e1</td>
</tr>
<tr>
<td>E2 [primitive]</td>
<td>e2</td>
</tr>
<tr>
<td>E3 [primitive]</td>
<td>e3</td>
</tr>
<tr>
<td>E4 [primitive]</td>
<td>e4</td>
</tr>
<tr>
<td>E5 [primitive]</td>
<td>e5</td>
</tr>
<tr>
<td>E6 [virtual]</td>
<td>OR(e4, e5)</td>
</tr>
<tr>
<td>E7 [primitive]</td>
<td>e7</td>
</tr>
<tr>
<td>E8 [virtual]</td>
<td>AND(e4, e7)</td>
</tr>
<tr>
<td>E9 [primitive]</td>
<td>e9</td>
</tr>
<tr>
<td>EC10 [composite]</td>
<td>NOT(e9)</td>
</tr>
<tr>
<td>EC11 [composite]</td>
<td>SEQ(e4, e10)</td>
</tr>
<tr>
<td>E14 [primitive]</td>
<td>e14</td>
</tr>
</tbody>
</table>

TABLE VI
SPECIFICATION ON THE TRANSITIONS OF FIG. 12

<table>
<thead>
<tr>
<th>transition [type]</th>
<th>condition</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0 [copy]</td>
<td></td>
<td>E4</td>
<td>EC6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C_E4**</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C_E4</td>
</tr>
<tr>
<td>T1 [copy]</td>
<td></td>
<td>E5</td>
<td>EC6</td>
</tr>
<tr>
<td>T2 [composite]</td>
<td>not specified</td>
<td>E7</td>
<td>EC8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C_E4</td>
</tr>
<tr>
<td>T3 [composite]</td>
<td>not specified</td>
<td>E9</td>
<td>EC10</td>
</tr>
<tr>
<td>T4 [composite]</td>
<td>not specified</td>
<td>EC10</td>
<td>EC11</td>
</tr>
<tr>
<td>T5 [Rule 000]</td>
<td>no condition</td>
<td>E0</td>
<td>E3</td>
</tr>
<tr>
<td>T6 [Rule 001]</td>
<td>no condition</td>
<td>E1</td>
<td>E2</td>
</tr>
<tr>
<td>T7 [Rule 002]</td>
<td>cond_Rule 002</td>
<td>E2</td>
<td>E3</td>
</tr>
<tr>
<td>T8 [Rule 003]</td>
<td>cond_Rule 003</td>
<td>E3</td>
<td>E2</td>
</tr>
<tr>
<td>T9 [Rule 004]</td>
<td>cond_Rule 004</td>
<td>EC6</td>
<td>E3</td>
</tr>
<tr>
<td>T10 [Rule 005]</td>
<td>cond_Rule 005</td>
<td>EC8</td>
<td>E14</td>
</tr>
<tr>
<td>T11 [Rule 006]</td>
<td>no condition</td>
<td>EC11</td>
<td>E9</td>
</tr>
</tbody>
</table>

Generally, rules in an ECA rule base are highly interrelated, which complicates the ECA rule management. There are quite a few results on the complexity of ECA rule interrelation. We have given some results by the CCPN approach [18]. In this paper, we focus on ECA rule modeling and simulation (for more details on the complexity analysis of ECA rules, see [17] and [18]).

REFERENCES

Joselito M. Medina (M’07) received the B.S. degree in computer science from the Guerrero Autonomous University, Guerrero, Mexico, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV-IPN), Mexico City, Mexico, in 2002 and 2005, respectively.

His current research interests include Petri net theory, active database systems, artificial intelligence, and programming languages.

Sergio V. Chapa received the B.S. and M.S. degrees in physics and mathematics from the National Institute Polytechnic in 1973 and 1980, respectively, and the Ph.D. degree in electrical engineering from Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV-IPN), Mexico City, Mexico, in 1991.

He worked as a Senior Software Engineer with the National Institute for Nuclear Research, Mexico City, where he was involved in designing and implementation of the scientific software for Laguna Verde Nuclear Power Station. He is currently a Research Professor with Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN. His research interests include visual language, database and knowledge systems, cellular automata, and scientific software Sturm–Liouville singular problems.

Prof. Vergara is a member of the American Mathematical Society and the Society for Industrial and Applied Mathematics.