

**CENTRO de INVESTIGACIÓN
y de ESTUDIOS AVANZADOS
del IPN**

**DEPARTAMENTO DE INGENIERIA ELECTRICA
SECCION DE COMPUTACION**

***Desarrollo de reglas ECA en Bases de Datos,
un enfoque de Red de Petri***

Tesis que Presenta el:

M. en C. Joselito Medina Marín¹

**para obtener el grado de Doctor en Ciencias en
la especialidad de Ingeniería Eléctrica Opción
Computación**

Director de la tesis

Dra. Xiaoou Li

México, D.F.

17 de octubre de 2005

¹Becario de CONACYT no. 158604

Desarrollo de reglas ECA en Base de Datos Activas,
Un enfoque de Red de Petri

Joselito Medina Marín

17 de octubre de 2005

Índice general

. Agradecimientos	1
. Resumen	3
. Abstract	5
1. Introducción	7
1.1. Estado del Arte	9
1.1.1. Sistemas Manejadores de BD comerciales	9
1.1.2. Sistemas Manejadores de BD no comerciales	11
1.2. Motivación	13
1.3. Planteamiento del problema	14
1.4. Objetivos	15
1.4.1. Objetivo general	15
1.4.2. Objetivos específicos	16
1.5. Metodología	16
1.6. Descripción del trabajo	17
1.6.1. Contribuciones	17
1.6.2. Organización de la tesis	18
2. Sistema de Base de Datos Activa	21
2.1. Sistema de Base de Datos	22
2.1.1. Modelos de Datos	23

2.2.	Comportamiento activo	29
2.2.1.	Modelo de conocimiento	30
2.2.2.	Modelo de ejecución	34
2.2.3.	Análisis de reglas ECA	36
2.3.	Sintaxis de reglas en BDA existentes	38
2.3.1.	Postgres	39
2.3.2.	SAMOS	40
2.3.3.	Ariel	41
2.3.4.	Starburst	42
2.3.5.	A-RDL	43
2.3.6.	Chimera	44
2.3.7.	HiPAC	45
2.3.8.	Ode	46
2.3.9.	Oracle	47
2.3.10.	Informix	47
2.4.	Comentarios	48
3.	Conceptos básicos de Redes de Petri	49
3.1.	Definición	49
3.2.	Representación de la estructura de la Red de Petri	52
3.3.	Disparo de transiciones	53
3.4.	Poder de representación de las PN	55
3.5.	Propiedades de PN	58
3.6.	Métodos de análisis de PN	62
3.6.1.	Simulación de una PN	64
3.7.	Red de Petri Coloreada	65
3.8.	Comentarios	68
4.	Red de Petri Coloreada Condicional	71
4.1.	Definición de Red de Petri Coloreada Condicional	72
4.2.	Elementos de la CCPN	75

4.3. Modelación de reglas ECA con CCPN	79
4.3.1. Algoritmo de conversión de reglas ECA a CCPN	82
4.4. Ejecución de la CCPN	92
4.5. Ejemplo	96
4.6. Comentarios	98
5. Modelación de Eventos Compuestos	103
5.1. Estado del arte	103
5.1.1. SAMOS	104
5.1.2. ODE	106
5.1.3. Snoop	106
5.1.4. EPL	107
5.1.5. CEDAR	107
5.1.6. Sistemas comerciales	109
5.2. Eventos Compuestos en CCPN	109
5.2.1. Conjunción	109
5.2.2. Disyunción	111
5.2.3. Negación	111
5.2.4. Secuencia	113
5.2.5. Simultáneo	115
5.2.6. Historia	117
5.2.7. Primero	118
5.2.8. Último	120
5.2.9. Alguno	120
5.3. Algoritmo de conversión de un evento compuesto a una CCPN	122
5.4. Ejemplo	135
5.5. Comentarios	137
6. Análisis de Terminación y Confluencia	141
6.1. Problemas de análisis estático	142
6.1.1. No terminación	142

6.1.2. Confluencia	142
6.2. Estado del arte	143
6.3. Matriz de Incidencia de CCPN	145
6.4. Analisis de Terminación y Confluencia con CCPN	148
6.4.1. Rutas	149
6.4.2. Análisis de Terminación	151
6.4.3. Ejemplo	156
6.4.4. Análisis de Confluencia	158
6.5. Comentarios	158
7. ECAPNSIM	161
7.1. Ambiente de desarrollo	162
7.2. Arquitectura de ECAPNSim	162
7.2.1. Kernel de ECAPNSim	164
7.2.2. Herramientas de ECAPNSim	166
7.3. Diagrama de clases	168
7.4. Diseño e Implementación	170
7.5. Plataforma independiente	171
7.5.1. Base de reglas	171
7.5.2. MS Access	173
7.5.3. FoxPro	176
7.5.4. Postgresql	185
7.6. Comentarios	191
8. Caso de estudio	195
9. Conclusiones y trabajo futuro	207
. Publicaciones	213
. Bibliografía	215

Agradecimientos

A Dios, por darme la oportunidad de existir y fuerzas para salir adelante.

En especial a mi esposa, quien me motivó y me dió ánimos para seguir adelante, por su apoyo en los momentos difíciles y amor incondicional.

A mis hijos, no me imagino mi vida sin ellos, gracias por existir.

A mis padres, por haberme dado esta educación e inculcado valores maravillosos.

A mis hermanos, gracias por su apoyo que siempre me han brindado.

A toda mi familia que es muy numerosa, pero agradezco su apoyo hacia mi persona, a mi esposa y a mis hijos.

A mi asesora la Dra. Xiaou Li, por sus brillantes ideas para el desarrollo de éste trabajo de tesis, por su amabilidad al atenderme para discutir temas sobre la tesis y en ocasiones distintos de la tesis.

A los Doctores Sergio V. Chapa y Pedro Mejía, por sus comentarios a mi documento de tesis, lo cual mejoró la estructura de la misma.

Al Dr. Adolfo Guzmán Arenas, que aún en momentos difíciles de su vida se tomó la molestia de atenderme y platicar conmigo.

al Dr. Luis Enrique Rocha Mier, por sus valiosos comentarios y observaciones, y por el tiempo dedicado a mis consultas.

Al Dr. Ernesto López Mellado, por compartir conmigo sus amplios conocimientos en el área de redes de Petri, lo cual me ayudó mucho para mejorar la escritura de mi documento.

A todos mis maestros de la Sección de Computación, que compartieron conmigo sus conocimientos en sus respectivas áreas, en particular a los Dres. Arturo Díaz Pérez, Luis Gerardo de la Fraga, Jorge Buenabad Chávez, Feliú Sagols Troncoso y Sergio Chapa Vergara. A las secretarias de la

Sección, Sofi, Flor y Feli, porque sin ellas no realizaríamos trámite administrativo alguno.

A mis hermanos de redes de Petri, por los debates que mantuvimos sobre el tema de investigación. A mis compañeros de la sección, por su amistad y compañerismo.

Al personal de las biblioteca de Ingeniería Eléctrica y de Matemáticas, así como a las personas que laboran en Servicios Escolares.

Al CONACYT por apoyarme con la beca de maestría y al CINVESTAV en general.

Resumen

Las bases de datos activas (BDA) son extensiones de las bases de datos (BD), las cuales, además de tener un comportamiento pasivo (modificar ú obtener información solicitada por el usuario), reaccionan ante la presencia de uno o más eventos en la BD. El comportamiento activo de una BD puede modelarse con las reglas evento-condición-acción (reglas ECA). La mayoría de las BDA comerciales utilizan el esquema de reglas ECA y cada una de ellas proporciona al usuario una sintaxis de definición de reglas. Sin embargo, el administrador de la BDA no puede llevar a cabo una simulación del comportamiento de la base de reglas ECA antes de su implementación en la BDA. Existen herramientas, tales como autómatas y redes de Petri (PNs, Petri nets), con las cuales puede llevarse a cabo la representación de reglas ECA; sin embargo, la representación de eventos mediante autómatas es una tarea complicada; en cambio, las PNs modelan fácilmente los sistemas basados en eventos y es posible desarrollar extensiones para sistemas que no pueden representarse con PN tradicionales. Una base de reglas ECA es considerada como un sistema basado en eventos y es posible representarla con una PN extendida, así como los eventos que las disparan.

En este trabajo de tesis se propone un marco teórico para el desarrollo y análisis de reglas ECA, basado en un modelo extendido de PN que contiene las características de los elementos (evento, condición y acción) de una regla ECA, a la que hemos denominado Red de Petri Coloreada Condicional (CCPN, Conditional Colored Petri Net). La CCPN tiene la capacidad de detectar eventos primitivos y nueve tipos diferentes de eventos compuestos (conjunción, disyunción, secuencia, simultáneo, negación, primero, último, historia y alguno). En este marco teórico se aprovecha la matriz de incidencia de la teoría de PN para llevar a cabo el análisis estático de la base de reglas ECA y determinar la presencia de ciclos potenciales en el disparo de reglas. Por otro lado, se desarrolló la interfaz gráfica ECAPNSim (ECA & PN Simulator) para diseñar la CCPN a partir de

reglas ECA. ECAPNSim lleva a cabo la simulación del comportamiento de una base de reglas ECA, o bien, se conecta a la BD para detectar eventos y ejecutar las acciones especificados en la CCPN.

A diferencia de otros trabajos relacionados al área, tanto proyectos de investigación como sistemas comerciales de BDA, en esta propuesta doctoral se ofrece una base teórica para el desarrollo, análisis, simulación y ejecución de reglas ECA, aplicadas en sistemas activos de BD.

Abstract

Active data bases (ADB) are database (DB) extensions, which, besides to have a passive behavior (to modify or to obtain data asked for by the user), react with the presence of one or more events in the DB. The active behavior of a DB can be modeled with event-condition-action (ECA) rules. Most of the commercial ADB use the ECA rules approach, and each one of them provides to the user a syntax for rule definition. Nevertheless, the ADB administrator cannot carry out a simulation of the ECA rules behavior before its implementation in the ADB. On the other side, there exists a graphical and mathematical tool of modeling and very powerful simulation, the Petri Nets (PN). PN can model easily systems based on events and it is possible to develop extensions for systems that cannot be modeled with traditional PN. An ECA rule base is considered as an event driven system and it can be represented with an extended PN.

In this thesis work is proposed a framework in order to develop and analyze ECA rules, which is based in an extended PN is proposed, named Conditional Colored Petri Net (CCPN), and it contains the characteristics of ECA rule elements (event, condition and action). CCPN has the ability to detect primitive and nine types of composite events (conjunction, disjunction, sequence, simultaneous, not, first, last, times, and any), moreover, it takes advantage of incidence matrix from PN theory, in order to perform the static analysis of ECA rule base and then to detect the presence of potentially infinite rule triggering. On the other hand, we have developed a graphical interface, named ECAPNSim, used to design the CCPN from ECA rules. ECAPNSim carries out the simulation of the ECA rule base behavior, or, it can be connected to a DB in order to detect events and to execute the actions specified in the CCPN.

Unlike related work, in this proposal a theoretical base is provided for the development, analysis, simulation, and execution of ECA rules, applied in active database systems.

Capítulo 1

Introducción

Desde la introducción de los sistemas informáticos en las actividades del hombre, se ha buscado la manera de almacenar información, para posteriormente ser reutilizada, ya sea como medio de consulta o para hacerle modificaciones a la misma y mantenerla actualizada. Ante la necesidad de almacenar toda la información posible, se requerían espacios cada vez mas grandes de almacenamiento, así como metodologías para almacenar la información y poder manejarla de manera eficiente. Además, en los esquemas tradicionales de manejo de información, cada persona u organización desarrollaba su propia aplicación que se ajustaba a sus propias necesidades, sin embargo, en este tipo de esquema existía una incompatibilidad total si se requería compartir información.

En vista de lo anterior, se desarrollaron las bases de datos (BD). Una BD es un conjunto de datos relacionados entre sí; los datos son los hechos conocidos que pueden registrarse y que tienen un significado implícito [2]. Una BD representa algún aspecto del mundo real, en ocasiones llamado **minimundo** o **universo de discurso**. Las modificaciones del minimundo se reflejan, a su vez, en la base de datos.

La creación de una BD, así como las modificaciones, agregaciones, eliminaciones y consultas de datos se hacen mediante un programa o un conjunto de programas que accederán a la BD para realizar estas operaciones. A este programa o conjunto de programas se les conoce como Sistema Manejador de Bases de Datos (DBMS, Database Management System). En conjunto la BD y el DBMS se les conoce como Sistema de Base de Datos.

Un Sistema de Base de Datos tradicional realiza las operaciones mencionadas sobre los datos, a

través de comandos o instrucciones definidas por el administrador de la Base de Datos o los usuarios de la misma, es decir, si un usuario de la BD desea agregar, modificar, eliminar o simplemente hacer una consulta a la BD, entonces debe ejecutar los comandos necesarios para llevar a cabo estas operaciones.

Sin embargo, existen sistemas en el mundo real, donde de acuerdo a lo que esté ocurriendo en el universo de discurso, es necesario realizar una acción o procedimiento dependiendo de los cambios que hayan ocurrido, si de antemano se conocen las acciones que se deben ejecutar y el momento en que éstas deben ejecutarse, entonces es factible definir una serie de procedimientos para incorporarle a la base de datos este comportamiento.

Los esquemas tradicionales de sistemas de bases de datos no tienen la capacidad de soportar a los sistemas descritos en el párrafo anterior, en los cuales es necesario especificarle a la BD un comportamiento de reacción automática ante los cambios que se susciten dentro del universo de discurso de la BD, ya que de hacerlo un usuario humano es posible que incurra en errores en la detección de eventos.

Por esta razón surgieron los sistemas activos de bases de datos, mejor conocidos como Sistemas de Bases de Datos Activas (SBDA), los cuales deben ser capaces de detectar los eventos que ocurren en el universo de discurso, verificar el estado de la base de datos durante la ocurrencia de eventos, y además, ejecutar acciones o procedimientos dentro de la BD sin la intervención directa de un usuario humano.

Para proporcionar el comportamiento activo a una BD es necesario definir reglas que describan la reacción automática que debe seguir el manejador de la BD, también conocidas como reglas activas. La forma más general para lograr la definición de las reglas activas es mediante la aplicación del modelo de *regla Evento-Condición-Acción* (ó regla ECA), en la cual se define el *evento* que debemos detectar, la *condición* que debemos evaluar contra el estado que guarde en esos momentos la BD, así como la *acción* o grupo de *acciones* que se ejecutarán si la evaluación de la condición es verdadera. La mayoría de los sistemas comerciales (SYBASE, ORACLE, SQL Server, etc.[3], [4]) ofrecen funciones para darle comportamiento activo a una base de datos, por medio de la definición de “triggers” o “disparadores”.

El área de aplicación de un SBDA es muy amplia, desde sistemas de bases de datos administrativos, hasta sistemas de bases de datos de control en aerolíneas y de control de hospitales,

así como monitoreo de transacciones financieras, identificación de actividades inusuales en el sistema, cumplimiento de restricciones de integridad, mantenimiento de datos derivados, generación oportuna de reportes, realización de procesos periódicos, entre otros.[5]

Además, se han utilizado SBDA para la detección de intrusos en redes, utilizando el sistema VenusIDS [40].

Sin embargo, el desarrollo de bases de reglas ECA se vuelve una actividad complicada cuando el número de reglas se incrementa. Los problemas con que se encuentran los desarrolladores de reglas son conocidos como el problema de *No terminación* y el problema de *confluencia*. En el problema de No terminación, la interrelación existente entre las reglas puede originar un disparo en cadena de la base de reglas y si en algún momento esta cadena regresa a la que comenzó con el disparo de reglas, entonces se provoca un disparo infinito de reglas que no terminan de dispararse. Por otro lado, una base de reglas ECA es confluyente siempre y cuando el disparo de un conjunto de reglas siempre obtenga un mismo estado final de la BD, sin importar el orden en que estas reglas sean disparadas.

Actualmente existen sistemas que pretenden incorporar un comportamiento activo a una BD, tanto comerciales como proyectos de investigación, sin embargo, cada uno de ellos ofrecen su propia sintaxis y plataforma de trabajo, convirtiéndolos en sistemas dependientes de la BD, además de que no ofrecen en su totalidad el análisis de reglas. A continuación se describen algunos de los más difundidos SBDA.

1.1. Estado del Arte

En esta sección se mencionan sistemas de BD que ofrecen un comportamiento activo, la cual está dividida en dos secciones, considerando por un lado a los sistemas de BD comerciales que proveen la definición de un sistema de reglas activas. Posteriormente se describen los sistemas de BD que no son sistemas comerciales, pero que también ofrecen la definición de reglas activas.

1.1.1. Sistemas Manejadores de BD comerciales

Los Sistemas Manejadores de BD comerciales soportan la implementación de “triggers” en varios niveles. Sin embargo, generalmente presentan ciertas limitaciones. Entre estas limitaciones

pueden mencionarse las siguientes: el evento del “trigger” solamente pueden construirse a partir de expresiones de SQL (como `update`, `insert`, `delete` o `select`) en una sola tabla de la BD; además, los “triggers” no pueden anidarse, es decir, que dentro de un “trigger” no puede invocarse a otro “trigger”.

Por ejemplo, en SYBASE [6], un sistema manejador de bases de datos relacionales, la parte condicional de un “trigger” solo puede referirse a una sola tabla, específicamente a la tabla donde se define el evento a detectar; un “trigger” no puede crearse sobre una vista (una vista es una tabla lógica, que muestra una parte de la base de datos; las vistas permiten “almacenar” de manera lógica los resultados de los queries o consultas a la BD); solamente un “trigger” puede asociarse con una operación en una tabla; la acción de un “trigger” está limitada a una secuencia de expresiones escritas en SQL; además, el disparo de “triggers” está limitado a un solo nivel, donde las acciones disparadas no provocan la activación de otros “triggers”.

INFORMIX [7] es un sistema manejador de base de datos basado en el modelo relacional el cual consiste en un conjunto de módulos que forman el sistema completo. De manera similar a SYBASE, en este sistema manejador de base de datos, un usuario puede crear un “trigger” en una tabla de la BD actual; un usuario no puede crear un “trigger” en una tabla temporal, una vista o en una tabla de catálogo del sistema. Además, en INFORMIX, si un usuario define en cada uno de los eventos de los “triggers” más de un elemento, los elementos de los eventos entre los diferentes “triggers” deben de ser mutuamente exclusivos, es decir, un mismo evento no puede disparar a dos o más “triggers”. Por ejemplo, en la siguiente definición de dos “triggers” en INFORMIX podemos apreciar que `trig2` es ilegal debido a que en su lista de campos que serán actualizados aparece uno (`num_paquete`) que se consideró para `trig1`.

```
CREATE TRIGGER trig1 UPDATE num_articulo, num_paquete ON Articulos
FOR EACH ROW(EXECUTE PROCEDURE proc1)
CREATE TRIGGER trig2 UPDATE num_orden, num_paquete ON Articulos
FOR EACH ROW(EXECUTE PROCEDURE proc2)
```

ORACLE [8] es un sistema manejador de bases de datos relacional con características que permiten la implementación del paradigma orientado a objetos, en donde los usuarios primeramente deben definir que tipo de evento activará a los “triggers”, además de escribir “triggers” con una especificación explícita para esos eventos. Los “triggers” se ejecutan implícitamente cuando un

comando `INSERT`, `UPDATE`, `DELETE` o `select` es utilizado sobre una tabla asociada, o en algunos casos sobre una vista, o cuando ocurren determinadas acciones en el sistema de BD. Un “trigger” en ORACLE es un bloque instrucciones en PL/SQL o Java y almacenado en la BD, o pueden ser escritos en subrutinas de lenguaje C, que se dispara cuando una determinada instrucción en SQL se va a ejecutar sobre dicha tabla. PL/SQL es un lenguaje de programación procedural utilizado en los sistemas manejadores de bases de datos ORACLE. La estructura de este lenguaje es similar al lenguaje de programación Pascal porque está estructurado en bloques y utiliza los mismos comandos (`BEGIN` y `END`) para delimitar a éstos bloques.

Microsoft SQL Server [9], uno de los productos que componen a la plataforma BackOffice de Microsoft, es un servidor para BD’s relacionales que utiliza como lenguaje de manipulación de datos una extensión del lenguaje SQL (Structured Query Language ó Lenguaje Estructurado de Consulta) llamado *Transact SQL*. Microsoft SQL Server también utiliza “triggers” para definir un comportamiento activo en su BD. Los eventos que se consideran para el disparo de los “triggers” en este manejador de BD son las operaciones de inserción, borrado o actualización de datos. Los “triggers” basados en instrucciones creadas enteramente en Transact SQL tienen una serie de limitaciones que vienen impuestas por el diseño de SQL Server: para evitar el disparo infinito o un ciclo infinito de disparo de “triggers”, un “trigger” no puede nunca disparar a otro “trigger” en su interior, además por estas mismas razones, un “trigger” no puede ejecutar una instrucción DDL (definición, modificación o eliminación de estructuras de tablas, índices, BD’s, etc.)

1.1.2. Sistemas Manejadores de BD no comerciales

Dentro de los sistemas no comerciales que proporcionan la definición de reglas activas podemos mencionar a Ariel, HiPAC, Startburst y POSTGRES.

El sistema Ariel [10] es la implementación de una base de datos relacional con un sistema de reglas tipo “built-in”. El enfoque tomado en el diseño de Ariel adopta el modelo de sistemas de producción, tal como el OPS5 (Official Production System 5) [11]. OPS5 es un lenguaje para ingeniería cognoscitiva que soporta el método de representación del conocimiento en forma de reglas, el cual incorpora un módulo unificador, un intérprete que incluye un mecanismo de encadenamiento progresivo, y herramientas para edición y depuración de los programas [12]. El OPS5 fue modificado para mejorar la funcionalidad y rendimiento de un sistema de producción en un ambiente de BD.

Estos cambios incluyeron una extensión del lenguaje de reglas de POSTQUEL, con una sintaxis semejante a un lenguaje de consultas, una red discriminatoria para evaluar la parte condicional de la regla diseñada de acuerdo al ambiente de la BD y medidas para integrar el procesamiento de reglas con transacciones y comandos de actualización a la BD orientados a conjuntos. El mecanismo que utiliza Ariel para validar la parte condicional de la regla, llamado A-TREAT, es una variación del algoritmo conocido como TREAT, al cual se le agregaron características para aumentar la velocidad de búsquedas en la BD para la evaluación de la parte condicional de la regla. Las reglas que se definen en Ariel pueden tener condiciones basadas en una mezcla de patrones, eventos y transiciones. Una característica importante de Ariel es su capacidad para manejar reglas de tipo condición-acción.

HiPAC [5] es una BDA orientada a objetos. El proyecto HiPAC es el pionero de muchas de las más importantes ideas en BDA's, tales como los modos de acoplamiento de reglas y la composición de eventos, aunque el diseño final de HiPAC no fue implementado en su totalidad. HiPAC utiliza el modelo relacional como marco de trabajo y el modelo de transacciones anidadas como el marco para la ejecución de reglas [13]. Cada regla en HiPAC está estructurada de acuerdo al paradigma evento-condición-acción. Los eventos pueden ser operaciones de la BD genéricos. La acción de la regla ECA, dentro de HiPAC, pueden contener operaciones de la BD, operaciones de transacción, operaciones de reglas, señales para identificar que un evento definido por el usuario a ocurrido o llamadas a los procedimientos de la aplicación. HiPAC estuvo asociado con la Base de Datos Orientada a Objetos (BDOO) pasiva PROBE, donde los objetos de este modelo fueron usados para almacenar a las reglas ECA de la parte extendida correspondiente a la BDA. HiPAC tiene la capacidad de ejecutar de manera simultánea ó paralela el disparo de reglas como subtransacciones.

Starburst [14] es un sistema de BD relacional desarrollado en el Centro de Investigaciones de Almaden - IBM, el cual fue complementado con el sistema de reglas Starburst. El lenguaje de reglas de Starburst difiere de la mayoría de los lenguajes de reglas de BDA en que está basado en transiciones de estado de BD arbitrarias en lugar de tuplas. La característica del sistema de reglas de Starburst que más llama la atención es su modelo de ejecución, en el cual las reglas se disparan por el efecto final de un conjunto de cambios o modificaciones realizadas sobre los datos almacenados en la BD [5]. Sin embargo, la semántica en la ejecución de las reglas es demasiado compleja, probablemente esto se deba a que al proveer más facilidades conduzca al desarrollo de

conjuntos de reglas difíciles de entender y mantener.

El sistema manejador de BD POSTGRES [15] utiliza un subsistema de reglas para ofrecer una reacción automática a la ocurrencia de eventos en el espacio de estados de la BD. POSTGRES, a pesar de ser un sistema manejador de BD relacional, ofrece facilidades para el manejo de objetos y de operaciones definidas por el usuario. El sistema de reglas POSTGRES está orientado a tuplas, además, la ejecución de las reglas definidas en él se efectúan inmediatamente después de que alguna modificación a una tupla dispara y satisface la condición de una o más reglas [3].

Entre los sistemas de BD que incorporan la definición de reglas, para proveer comportamiento activo a la BD, se encuentran SYBASE, ORACLE, INFORMIX, SQL Server, Progress y Visual Fox Pro.

1.2. Motivación

Debido a la necesidad de utilizar sistemas con capacidad de reacción a la ocurrencia de eventos, también conocidos como sistemas reactivos, en el área de BD se hizo necesaria la implementación de las BDA, sin embargo, cada una de las propuestas y sistemas manejadores de BD activas que se han analizado tienen una definición de reglas ECA exclusiva de cada sistema. En el caso de los sistemas de BDA comerciales podemos mencionar que solo manejan la definición de eventos que modifican, agregan, eliminan o seleccionan datos (por medio de los comandos update, insert, delete y select, respectivamente) y de los eventos compuestos (eventos que ocurren a partir de la ocurrencia de dos o más eventos utilizando un álgebra de eventos: conjunción, disyunción, negación, etc.) solo soporta la definición de la disyunción de eventos, además de que no debe existir una relación directa entre la acción de una regla y el disparo de otra [6], [7], [8]. Los sistemas de BDA que son aún prototipos de investigación ofrecen una mayor cantidad de propiedades de BDA que los sistemas comerciales, sin embargo, al igual que los anteriores, la sintaxis que utilizan en la definición de reglas es exclusiva para tales sistemas. Por otro lado, en el diseño y desarrollo de una base de reglas activas pueden presentarse problemas como el de No-terminación (disparo infinito de reglas al caer en un ciclo de disparo de reglas) y el de confluencia (cuando en el disparo simultáneo de dos reglas, el estado final de la BD depende del orden en que éstas se disparan).

Cada sistema de BDA ofrece su propia sintaxis de especificación de reglas ECA existiendo una incompatibilidad entre los diferentes manejadores de SBDA.

La fortaleza de un modelo depende de la capacidad que ofrezca para ser utilizado, sin ningún problema, en el área para la cual fue diseñado y en diferentes situaciones de aplicación donde sea requerido.

En el área de BDA se han propuesto diferentes plataformas, prototipos y sintaxis para la especificación de las reglas ECA, sin embargo hay características importantes, que están inherentes dentro del desarrollo de reglas ECA [5] [25], que en algunas propuestas no son consideradas, en algunas otras sí y viceversa. Además, cada una de las propuestas está pensada en el diseño de un SBDA, no en un modelo que cubra las características propias de una base de reglas ECA y que pueda ser aplicado a cualquier diseño de SBDA.

Los SBDA se utilizan para sistemas de control de misiones de aeronaves, sistemas de navegación de a bordo, en la industria manufacturera de automóviles, en aplicaciones de telecomunicaciones, sistemas de simulación, subastas por comercio electrónica y ciertos aspectos de workflows.

En vista de lo anterior, estamos proponiendo un marco teórico basado en Teoría de red de Petri, en el cual se ofrece un modelo formal con características de una red de Petri Coloreada al que hemos denominado Red de Petri Coloreada Condicional (CCPN, Conditional Colored Petri Net).

En el modelo propuesto en este marco es posible almacenar suficiente información de la base de reglas ECA para proporcionar el comportamiento activo adecuado a una BD. Además, el modelo de conocimiento de una regla ECA se puede incorporar dentro de la CCPN, y el modelo de ejecución de la regla ECA se ajusta al disparo de transiciones de la teoría de PN. Finalmente, dentro de la CCPN es posible simular, ejecutar y analizar una base de reglas ECA. aunado a lo anterior, dentro de la propuesta se realiza el análisis de las reglas, ofreciendo una herramienta de análisis de reglas basado en la matriz de incidencia de PNs.

1.3. Planteamiento del problema

En esta tesis doctoral se propone un modelo formal basado en PN. Con este modelo se hará la definición de una base de reglas ECA visualizando de manera gráfica la CCPN correspondiente a las reglas ECA en desarrollo.

En el modelo se toman características de la red de Petri coloreada, como el manejo de información dentro de los tokens (colores) para la conformación de eventos compuestos y la evaluación de la parte condicional de la regla ECA.

Cada uno de los elementos de la regla ECA (el evento, la condición y la acción) pueden mapearse a elementos de una PN (lugares y transiciones).

Las PNs se utilizan en la modelación y simulación de sistemas manejados por eventos y los sistemas de bases de datos activas también se consideran como sistemas manejados por eventos. Por lo tanto, el proceso de disparo y ejecución de las reglas ECA puede realizarse a través de PN's.

Con la CCPN obtenida de la base de reglas ECA, es posible hacer la simulación de la base de reglas y posteriormente, con la misma CCPN, realizar su implantación sobre una base de datos pasiva.

Para el manejo de eventos compuestos, se proponen estructuras de CCPN que fungen como constructores de los eventos compuestos basados en CCPN.

El análisis de la CCPN para detectar problemas presentes en bases de reglas ECA, puede realizarse con la matriz de incidencia de la teoría de PN, donde pueden determinarse las rutas que se forman a partir de la conexión de reglas.

La aplicación de la CCPN puede mostrarse a través de la interfaz gráfica ECAPNSim, en la cual puede realizarse la modelación, ejecución y análisis de una base de reglas ECA, representada como una CCPN.

Utilizando el ECAPNSim, se llevan a cabo pruebas con diferentes manejadores de bases de datos relacionales y con diferentes bases de datos, para proporcionarles un comportamiento activo a partir de la definición de reglas ECA en CCPN.

1.4. Objetivos

Los objetivos que se pretenden alcanzar con este trabajo de tesis doctoral los dividimos en un objetivo general y en cinco objetivos específicos.

1.4.1. Objetivo general

- Proponer un marco teórico formal basado en Teoría de Redes de Petri, para el desarrollo de reglas Evento-Condición-Acción (reglas ECA), que permita la implementación de Sistemas de Bases de Datos Activas y el análisis estático de reglas ECA.

1.4.2. Objetivos específicos

1. Desarrollar el modelo de Red de Petri Coloreada Condicional (CCPN, Conditional Colored Petri Net), que proporcione las características del modelo de conocimiento y de ejecución de las reglas ECA.
2. Realizar el análisis estático de las reglas ECA, utilizando como representación de reglas la propia CCPN, para detectar problemas como el de No Terminación y Confluencia, presentes en el desarrollo de bases de reglas ECA.
3. Implementar una interfaz gráfica derivada de los conceptos funcionales de la CCPN, para modelar y simular el comportamiento de una base de reglas ECA, además, de proporcionarle un comportamiento activo a un SBD pasivo.
4. Probar la independencia de plataforma de la CCPN con el uso de la interfaz gráfica descrita en el punto anterior.
5. Desarrollar herramientas necesarias para el desarrollo de bases de reglas ECA, basadas en la interfaz gráfica, tales como herramientas de análisis estático y dinámico, generador de la matriz de incidencia de la CCPN, editor de reglas ECA, compilador de reglas y un módulo de explicación para establecer la comunicación con el desarrollador de reglas ECA.

1.5. Metodología

La metodología a seguir para lograr los objetivos específicos, y en consecuencia el objetivo principal, se describe a continuación:

- Estudio y comprensión de los conceptos que forman parte del área de Bases de Datos Activas, con la finalidad de tener presentes las características fundamentales que un sistema de base de datos debe ofrecer, para ser considerado como un sistema activo.
- Evaluación del estado del arte para conocer los trabajos relacionados en el área de BDA, así como las ventajas y desventajas que presentan los trabajos propuestos en el área.

- Estudio y análisis de la teoría de red de Petri para identificar los elementos de PN que serán utilizados en el modelo propuesto.
- Estudio de las herramientas de análisis de la teoría de PN, para proponer un modelo basado en PN, para detectar los problemas de No terminación y Confluencia.
- Desarrollo del marco teórico basado en PN para darle funcionalidad activa a una base de datos.
- Desarrollo del modelo formal extendido de una PN que modele reglas ECA.
- Desarrollo de algoritmos para la detección del problema de No terminación y de Confluencia.
- Implementación del modelo propuesto mediante el uso de un lenguaje de programación orientado a objetos e independiente de plataforma.
- Llevar a cabo la conexión con diferentes sistemas manejadores de bases de datos (como PostgreSQL, Visual FoxPro y MS Access), y mostrar la independencia de plataforma que, la interfaz implementada, soporta.
- Implementación de módulos para llevar a cabo el análisis estático de la base de reglas.

1.6. Descripción del trabajo

1.6.1. Contribuciones

De las contribuciones que ofrecemos en el desarrollo de la presente tesis doctoral consideramos como la más importante la siguiente:

- Un marco teórico basado en un modelo extendido de red de Petri formalizado, al que hemos denominado Red de Petri Coloreada Condicional (CCPN, Conditional Colored Petri Net), en el cual es posible realizar la modelación, análisis, simulación y ejecución de una base de reglas ECA, utilizadas para el desarrollo de bases de reglas activas en BDA's, en un mismo ambiente de operación.

Sin embargo, a partir del modelo de CCPN es posible utilizar las propiedades inherentes en una PN para soportar propiedades de las reglas ECA dentro del modelo de PN. Por lo tanto, las contribuciones secundarias de nuestro trabajo serían:

- Patrones para la definición de eventos compuestos utilizando estructuras de CCPN para su representación y detección.
- Método de análisis estático de una base de reglas ECA para detectar los problemas de No Terminación y Confluencia. El análisis se realiza en la misma definición de la base de reglas ECA como una CCPN, utilizando como herramienta de análisis a la *matriz de incidencia*, propia de la teoría de PN tradicional.
- Ambiente gráfico para la definición de reglas ECA.
- Interfaz gráfica, al que hemos denominado ECAPNSim (ECA - PN Simulator) en el cual puede modelarse, simularse y ejecutarse una base de reglas ECA, utilizando como repositorio de datos cualquier manejador de BD relacional que soporte una conexión ODBC-JDBC.

1.6.2. Organización de la tesis

La estructura de esta trabajo está dividida en nueve capítulos y se encuentra organizada de la siguiente manera:

En el capítulo 1 se presenta el estado del arte en que se encuentran los sistemas de BDA's actuales más conocidos, y en base a las carencias que presentan se da una descripción de la motivación que propició el desarrollo de la presente tesis doctoral. Se plantea el problema que se va a atacar, así como la solución que se propone. Más adelante, se muestran las contribuciones generadas en esta investigación y al final se muestra, en términos generales, el contenido de todo el documento.

Un panorama que engloba la descripción y el funcionamiento de una BDA es mostrado en el capítulo 2. En este capítulo se define el concepto de regla ECA y cada uno de los elementos que la componen. Se describe qué es un evento compuesto y cuales son las diferentes modalidades en que puede presentarse, dentro de una base de reglas activas. Por otro lado, se presenta la descripción de las propiedades que debe tener una BDA y las diferentes BDA que se conocen, tanto comerciales como libres y prototipos de investigación.

En el capítulo 3 se presentan los conceptos fundamentales de la teoría de red de Petri, así como los métodos de análisis de la teoría de PN y las extensiones al modelo que se han desarrollado con la finalidad de representar sistemas cuya modelación con PNs tradicionales sería un trabajo complejo.

La solución propuesta al desarrollo de base de reglas activas utilizando la CCPN, se describe en el capítulo 4. Se comienza por una definición general de la teoría de PN, así como sus propiedades y herramientas de análisis con que cuenta. Posteriormente, debido a que se toman elementos de una red de Petri Coloreada (CPN) para el desarrollo de la CCPN, se da un bosquejo sobre conceptos de CPN. Con las definiciones anteriores, se procede a definir formalmente la CCPN, la manera en que modela a una base de reglas ECA y se muestran ejemplos de aplicación.

En el capítulo 5 se describe la manera en que los *eventos compuestos* son modelados utilizando la CCPN, donde se presentan las estructuras de CCPN para cada uno de ellos y los algoritmos utilizados para la generación de las estructuras de CCPN que denotan a los eventos compuestos. Los análisis de terminación y confluencia, de las bases de reglas activas, se presentan como parte del capítulo 6, donde se mencionan a diferentes autores y a su forma en que ellos realizan el análisis de terminación y el análisis de confluencia. Se retoma el concepto de matriz de incidencia, herramienta de análisis que ofrece la teoría de PN, que utilizamos en esta investigación para realizar la búsqueda de rutas en la CCPN, útiles para llevar a cabo los análisis de terminación y confluencia. Posteriormente se describe cómo se utiliza la matriz de incidencia en la CCPN para realizar éstos análisis.

En el capítulo 7 se detalla la implementación del modelo CCPN en una interfaz gráfica diseñada en el lenguaje de programación orientado a objetos Java, su diagrama de clases, la creación, visualización, manipulación, simulación y ejecución de la CCPN para proporcionarle comportamiento activo a un repositorio de datos. En el capítulo 8 se muestran casos de estudio donde se aplica el modelo propuesto en sistemas de BD, proporcionándoles comportamiento activo. Las conclusiones sobre los resultados de la investigación doctoral se listan en el capítulo 9, además del trabajo futuro con lo que se pretende seguir sobre esta línea de investigación.

Capítulo 2

Sistema de Base de Datos Activa

Los esquemas tradicionales de sistemas de bases de datos fueron creados para almacenar información vital dentro de una organización. De esta manera, se representa una parte del mundo real que es importante conservar. Así, si existe algún cambio en la parte del mundo real que se tiene almacenada en el sistema de base de datos, éste también se refleja dentro del sistema de base de datos.

Sin embargo, algunas situaciones no pueden modelarse siguiendo este patrón. Por ejemplo, imaginemos el sistema que se tiene en la Central de Autobuses, en los días de vacaciones y días festivos existe mucha afluencia de personas para abordar autobuses y llegar a distintos destinos; por lo que es necesario agregar salidas de autobuses adicionales, de acuerdo a la cantidad de usuarios que arriben a la terminal. Si el número de usuarios sobrepasa un cierto umbral, entonces se programan más salidas de autobuses.

El problema lo podemos atacar de dos maneras:

1. Instalar en cada terminal de trabajo un módulo que verifique el número de usuarios; pero tendría que instalarse el mismo módulo en cada terminal, propiciando duplicidad en los programas.
2. Otra manera sería hacer consultas periódicas a la BD, sin embargo, si se realiza una verificación del estado de la BD por períodos muy cortos se estaría consumiendo mucha potencia de cómputo. Por el otro lado, si se realiza una verificación del estado de la BD por períodos

largos, podría ser demasiado tarde para llevar a cabo una acción, pueden perderse eventos que ocurran dentro del intervalo de búsqueda y nuestro mecanismo de verificación no podrá detectarlos, además de que el orden de los eventos no es posible determinarlos.

En los enfoques descritos se presentan problemas que afectan completamente el desempeño de nuestro sistema, por tal motivo, es necesario implementar mecanismos que reaccionen automáticamente en tiempo y forma ante la ocurrencia de eventos en el interior y exterior de la BD, de ahí el surgimiento de los sistemas de bases de datos activas.

Un sistema de Base de Datos Activa (SBDA), debe de ofrecer la funcionalidad de un SBD tradicional, además de proporcionar un comportamiento activo al SBD, es decir, además de definir el modelo de datos y el conjunto de programas que lo manipularán (esquema tradicional de BD), es necesario definir una base de reglas activas donde se especifique el comportamiento que esperamos sea realizado por el SBDA. A continuación se describen los conceptos fundamentales de un SBD, útiles para la comprensión de los conceptos de SBDA que más adelante se describen.

2.1. Sistema de Base de Datos

Un sistema de Base de Datos (SBD) se forma a partir de la unión de una base de datos (BD) ó repositorio de datos, y de un conjunto de programas que se encargarán de manipular a los datos almacenados en la BD [26]. En la figura 2.1 se muestra el esquema de un SBD.

Los SBD's fueron diseñados para manejar grandes volúmenes de información. El manejo de datos involucra la definición de las estructuras donde serán almacenados los datos, así como de proveer mecanismos que manipulen eficaz y adecuadamente a éstos datos.

Una BD no es mas que el conjunto datos que tienen relación entre sí y mantienen un significado implícito. Como se mencionó anteriormente, en la BD se refleja una parte del mundo real, solo aquella parte que es de nuestro interés, llamada **minimundo** o **universo de discurso**. Si el estado del mundo real es modificado, tales modificaciones también son realizadas dentro de la BD, para mantener una coherencia de información entre el mundo real y el minimundo.

Los mecanismos de manipulación de datos, en lo que se refiere a la creación de la BD, las modificaciones, agregaciones, eliminaciones y acceso a los datos, se realizan mediante un programa o un conjunto de programas conocido como Sistema Manejador de Bases de Datos (SMBD).

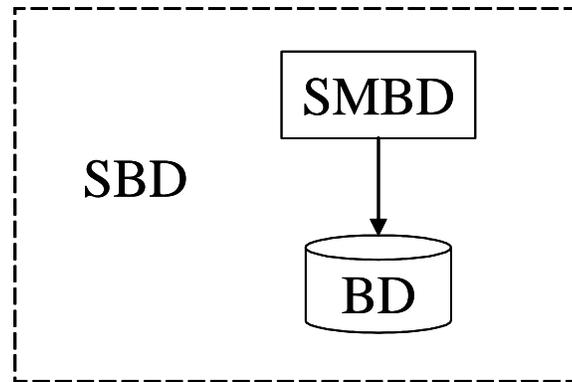


Figura 2.1: Esquema tradicional de un SBD.

La estructura de la BD está definida por el *esquema conceptual de la BD*. Los esquemas son especificados utilizando un *lenguaje de definición de datos* (DDL) y el acceso a la BD es realizado a partir de un *lenguaje de manipulación de datos* (DML). La unión del DDL y del DML forman el *modelo de datos*. [27]

Un punto muy importante en la creación de la BD es el modelo de datos que se utilizará. El modelo de datos con que esté diseñada la BD influye fuertemente en la definición de la base de reglas ECA, ya que la detección de eventos, verificación de la condición y la ejecución de las acciones de las reglas serán llevadas a cabo en coordinación con el modelo de datos con que se encuentre estructurada la información.

Los modelos de datos más ampliamente utilizados se clasifican en *modelos lógicos basados en objetos*, *modelos lógicos basados en registros* y *modelos físicos*. [26], los cuales se describen a continuación:

2.1.1. Modelos de Datos

Bajo la estructura de una base de datos está el *modelo de datos*, que es una colección de herramientas conceptuales para realizar la descripción de los datos, definir relaciones entre los datos, establecer la semántica de los datos y verificar las restricciones para consistencia del sistema. Los diferentes modelos de datos que han sido propuestos caen dentro de tres grupos diferentes: modelos lógicos basados en objetos, modelos lógicos basados en registros y modelos físicos. [26]

Modelos lógicos basados en objetos

Los modelos lógicos basados en objetos son utilizados para describir datos en los niveles lógico y de visualización. Están caracterizados por el hecho de que proporcionan mucha flexibilidad en su estructura y permiten especificar restricciones de datos. Hay muchos modelos diferentes, entre los más conocidos se encuentran:

- El modelo de entidad-relación.
- El modelo orientado a objetos.
- El modelo de datos semánticos.
- El modelo de datos funcionales.

A continuación se da una breve descripción de los dos primeros, los dos restantes son poco utilizados, por lo tanto son omitidos.

El modelo de entidad-relación.

El modelo de datos de entidad-relación (E-R) está basado en una percepción del mundo real que consiste de una colección de objetos básicos, llamados *entidades*, y de *relaciones* entre estos objetos. Una entidad es una "cosa" ó un "objeto" en el mundo real que es distinguible de otros objetos. Una *relación* es una asociación entre varias entidades. El conjunto de todas las entidades del mismo tipo y el conjunto de todas las relaciones del mismo tipo forman un *conjunto de entidades* y un *conjunto de relaciones*, respectivamente.

La estructura lógica puede ser expresada gráficamente mediante un *diagrama E-R*, el cual se construye utilizando los siguientes componentes:

- Rectángulos, representan conjuntos de entidades.
- Elipses, representan atributos.
- Rombo, representan las relaciones que existen entre las entidades.
- Líneas, conectan los atributos a los conjuntos de entidades y los conjuntos de entidades a las relaciones.

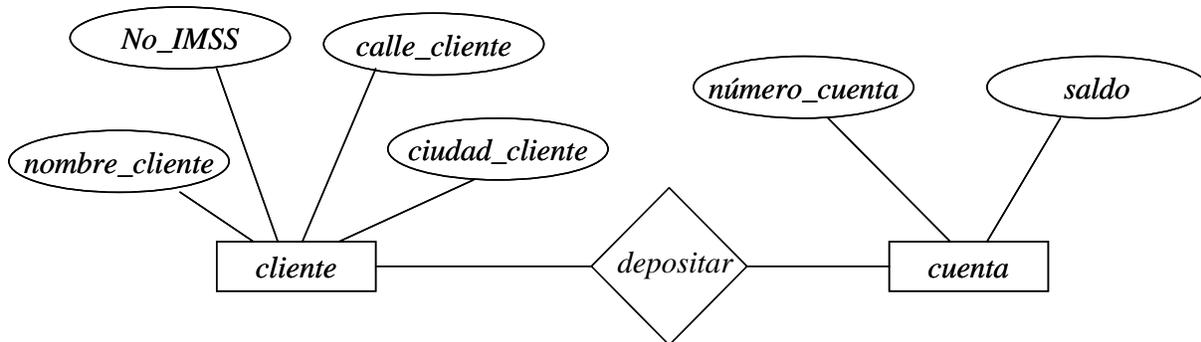


Figura 2.2: Ejemplo de un diagrama E-R.

Cada componente está etiquetado con el nombre de la entidad o de la relación a la que representa. Como un ejemplo, consideramos parte de un sistema de base de datos de un banco, el cual consiste de clientes y de las cuentas que tienen. Su diagrama E-R se muestra en la figura 2.2.

El modelo orientado a objetos.

Al igual que el modelo E-R, el modelo orientado a objetos está basado en una colección de objetos. Un objeto tiene valores almacenados en *variables de instancia* que se encuentran dentro de él. Además, contiene segmentos de código que operan como parte del objeto, a estos segmentos de código se les llama *métodos*.

Los objetos que contienen el mismo tipo de valores y los mismo métodos están agrupados en *clases*. Una clase puede ser vista como la definición de un tipo de dato para objetos. La combinación de datos y métodos formando una definición de tipo de dato es semejante a un tipo de dato abstracto en lenguajes de programación.

El único medio por el cual un objeto puede acceder a los datos de otro objeto es a través de la invocación de un método del otro objeto. Esta acción se le conoce como el *envío de mensajes* de un objeto a otro.

A diferencia de las entidades en el modelo E-R, cada objeto tiene una identidad única, independientemente de los valores que contenga. Por lo tanto, dos objetos son distintos aunque contengan los mismos valores. La distinción entre los objetos se lleva a cabo en el nivel físico a través de la asignación de identificadores diferentes para cada objeto.

Modelos lógicos basados en registros

Los modelos lógicos basados en registros son usados en la descripción de datos en los niveles lógico y de visualización. A diferencia de los modelos de datos basados en objetos, éstos se utilizan tanto para especificar la estructura lógica de la base de datos como para proporcionar una descripción de alto nivel de la implementación.

A los modelos basados en registros se les conoce así porque la base de datos está estructurada en registros de formato fijo de varios tipos. Cada tipo de registro define un número fijo de campos, ó atributos, y cada campo generalmente tiene una longitud fija. Los tres modelos de datos basados en registros más aceptados son el modelo relacional, el modelo de red y el modelo jerárquico.

Modelo relacional.

El modelo relacional usa una colección de tablas para representar datos y las relaciones que existen entre ellos. Cada tabla tiene múltiples columnas, y cada columna tiene un nombre único. En la figura 2.3 se presenta un ejemplo de una base de datos relacional, la cual se compone de dos tablas: una muestra los clientes de un banco y la otra muestra las cuentas que pertenecen a los clientes.

Modelo de red.

En el modelo de red, los datos son representados por colecciones de *registros*, y las relaciones entre los datos son representados por *ligas*, las cuales pueden considerarse como punteros. Los registros en la base de datos están organizados como colecciones de gráficas arbitrarias. La figura 2.4 presenta un ejemplo de una base de datos de red que utiliza la misma información presentada en la figura 2.3.

Modelo jerárquico.

El modelo jerárquico es parecido al modelo de red en el sentido de que los datos y las relaciones entre ellos están representados por registros y ligas, respectivamente. Pero difiere del modelo de red en que los registros están organizados como colecciones de árboles, en lugar de gráficas arbitrarias. La figura 2.5 presenta un ejemplo de una base de datos jerárquica que utiliza la misma información presentada en la figura 2.4.

<i>nombre_cliente</i>	<i>No_IMSS</i>	<i>calle_cliente</i>	<i>ciudad_cliente</i>	<i>número_cuenta</i>
José Andrade	9202-75-0085	Insurgentes	México	6071268270
Rogelio Marín	0653-65-1685	Corregidora	Guadalajara	4598632458
Francisco Medina	0564-45-7894	Ticomán	México	1684523457
Julio Durán	6854-68-1972	Cuauhtémoc	Acapulco	7851338945
José Andrade	9202-75-0085	Insurgentes	México	5674354251
Guadalupe Serna	1548-31-6454	Rubén Mora	Acapulco	6848673154
Angélica Sánchez	9684-83-4211	Loma Bonita	Acapulco	4361843466
Rogelio Marín	0653-65-1685	Corregidora	Guadalajara	5674354251

<i>número_cuenta</i>	<i>saldo</i>
6071268270	500
4598632458	700
1684523457	400
7851338945	650
5674354251	900
6848673154	750
4361843466	700

Figura 2.3: Ejemplo de una base de datos relacional.

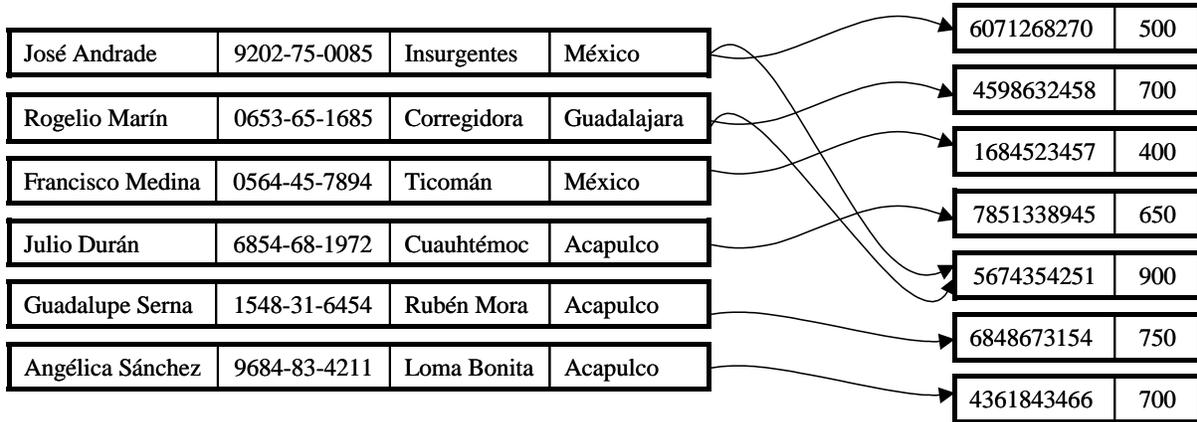


Figura 2.4: Ejemplo de una base de datos de red.

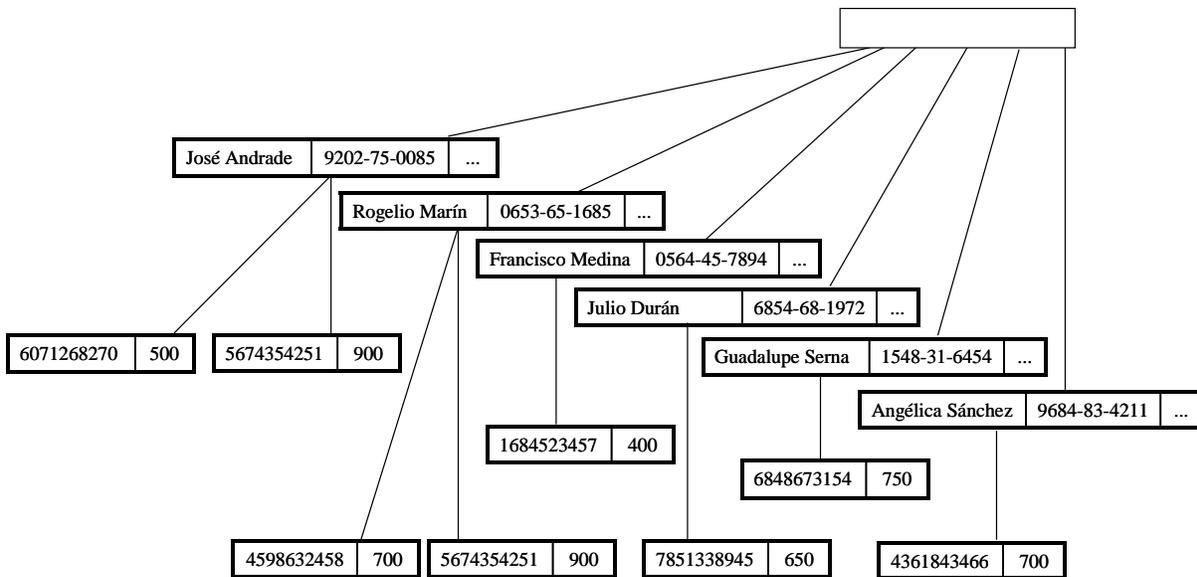


Figura 2.5: Ejemplo de una base de datos jerárquica.

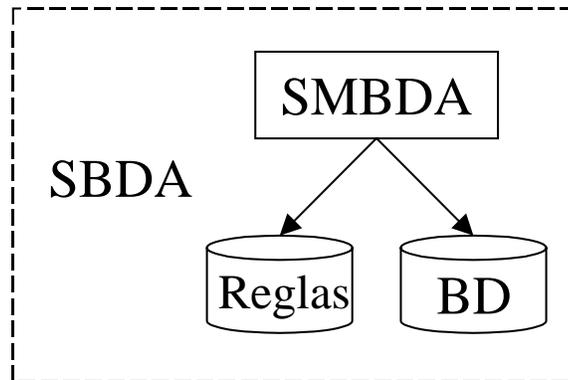


Figura 2.6: Esquema de un SBDA.

Modelos físicos

Los modelos de datos físicos son usados para describir datos en el nivel más bajo. A diferencia de los modelos lógicos de datos, son pocos los modelos físicos de datos que se utilizan. Dos de los más conocidos son el *modelo unificado* y el *modelo de estructura de memoria*.

Sin embargo, la definición del modelo de datos con que se va diseñar un SBD así como el conjunto de programas generados para la manipulación de los datos no son suficientes para representar universos de discurso donde se requiere de una reacción automática por parte del SBD. Para tal efecto, se introdujeron los Sistemas de Bases de Datos Activas.

2.2. Comportamiento activo

Un SDBA es una extensión de un SBD “pasivo” con la posibilidad de especificar un comportamiento “reactivo”. Parte fundamental en la especificación del comportamiento reactivo es la definición de una base de reglas activas, que proporcionen los mecanismos de reacción al SBD. En la figura 2.6 se presenta un esquema de la arquitectura general que un SBDA debe contener.

Algunas de las aplicaciones de SBDA son [27], [5], [26]:

- monitoreo de transacciones financieras,

- identificación de actividades inusuales en el sistema,
- cumplimiento de restricciones de integridad,
- mantenimiento de datos derivados,
- generación oportuna de reportes,
- realización de procesos periódicos,
- entre otras.

En vista de lo anterior, un SDBA debe ofrecer un *modelo de conocimiento* y un *modelo de ejecución* en el manejo de las reglas activas.

2.2.1. Modelo de conocimiento

El modelo de conocimiento del SBDA [5] indica al sistema la manera en que éste debe comportarse ante la presencia de ciertos eventos relevantes y el estado que guarde la BD en el momento en que ocurran estos eventos [27].

El enfoque más utilizado para el modelo de conocimiento del SBDA es la definición de *reglas Evento-Condición-Acción* ó simplemente *reglas ECA* [5].

Regla ECA

La definición de la base de reglas ECA es la que provee de la funcionalidad *reactiva* al SBDA y está estrechamente ligada a la sintaxis del lenguaje de reglas que tenga el SBDA donde se deseen implementar. [27]

Una regla ECA está formada por tres elementos: el *evento*, la *condición* y la *acción*. La forma general para representar a una regla ECA es la siguiente:

```
on evento  
if condición  
then acción
```

El *evento* de la regla describe un suceso al cual la regla debe de ser capaz de responder. La *condición* de la regla examina el contexto de la base de datos en el cual el evento ocurrió. Y finalmente, la

acción de la regla describe la tarea que será llevada a cabo por la regla si el evento correspondiente ha tomado lugar y la evaluación de la parte condicional de la regla resulta verdadera. [5]

La mayoría de los SBDA soportan reglas ECA con sus tres componentes. En algunas propuestas de SBDA la parte de evento o de condición pueden ser omitidas o estar implícitas en los otros elementos de la regla. Si no se proporciona el evento de la regla, entonces estaríamos manejando una regla del tipo *condición-acción*, mejor conocida como *regla de producción*. Si la condición de la regla no es especificada, entonces se trata de una regla *evento-acción*, donde a partir de la ocurrencia del evento de la regla se ejecuta la acción especificada.

Evento Un evento es algo que ocurre en un punto del tiempo. Por lo que la especificación de un evento involucra la descripción de un suceso que va a ser monitoreado. La naturaleza de la descripción y la manera en que el evento puede detectarse, depende en gran medida de la fuente ó generador de eventos. Las diferentes alternativas de generadores de eventos son:

- **Operaciones de estructura.** En las cuales el evento es generado a partir de una operación sobre algún elemento de la estructura de la BD. Por ejemplo agregar una tupla, modificar un atributo, etc.
- **Transacción.** Donde el evento es generado a partir de un comando de transacción; por ejemplo *abort*, *commit*, *begin-transaction*.
- **Definidas por el usuario.** En este caso, un mecanismo de programación es utilizado para permitir que un programa de aplicación genere explícitamente la ocurrencia de un evento; por ejemplo, se genera automáticamente un señal en respuesta de la introducción de datos por parte del usuario de la BD.
- **Reloj.** Un evento de reloj es generado en un punto determinado de tiempo. Existen tres tipos de eventos de reloj que deben considerarse: *absoluto* (por ejemplo, 15 de diciembre de 2005 a las 15:25 horas); *relativo* (por ejemplo, 5 horas después de que se agregaron ciertos registros) y *periódico* (por ejemplo, respaldar la BD cada viernes a las 23:00 horas).
- **Externos.** En este caso, el evento es generado a partir de un suceso externo a la BD (por ejemplo, la temperatura de un paciente está por arriba de los 40 grados).

Además, un evento puede ser de dos tipos, *primitivo* y *compuesto*.

Un evento primitivo es alcanzado cuando se da una sola ocurrencia de un evento que pertenece a la una de las posibilidades de fuente de eventos mencionado anteriormente.

Por otro lado, un evento compuesto se da a partir de alguna combinación de eventos primitivos y/o compuestos, utilizando una serie de constructores de eventos compuestos, a los cuales se les conoce como el *álgebra de eventos*.

Los constructores u operadores del álgebra de eventos varían de acuerdo a los diferentes SBDA. Entre los más importantes se encuentran los siguientes:

Disyunción, $e_1 \vee e_2 \vee \dots \vee e_n$, donde el evento compuesto de la disyunción ocurre cuando ha ocurrido uno de los eventos $e \in \{e_1, e_2, \dots, e_n\}$.

Conjunción, $e_1 \wedge e_2 \wedge \dots \wedge e_n$, donde el evento compuesto de la conjunción ocurre cuando han ocurrido todos los eventos $e_i \in \{e_1, e_2, \dots, e_n\}$, $i = 1, 2, \dots, n$, en cualquier orden.

Secuencia, $seq(e_1, e_2, \dots, e_n)$, donde el evento compuesto *secuencia* es alcanzado cuando ocurre en primera instancia el evento e_1 , posteriormente ocurre el evento e_2 y así sucesivamente hasta que ocurra el evento e_n .

Simultáneo, $sim(e_1, e_2, \dots, e_n)$, este evento compuesto es alcanzado cuando ocurren los eventos e_1, e_2, \dots, e_n en el mismo instante de tiempo.

Negación, $\sim e_1$ en *intervalo*, es la no ocurrencia del evento e_1 en el *intervalo* de tiempo especificado. Se dice que el evento ha tomado lugar al término del intervalo si e_1 no ocurre.

Primero, $first(e_1)$ en *intervalo*, este evento compuesto ocurre cuando en determinado *intervalo* se presenta en al menos una ocasión el evento e_1 , tomando para efecto de evaluación de la condición de la regla a la primera ocurrencia de e_1 dentro del *intervalo*.

Ultimo, $last(e_1)$ en *intervalo*, a semejanza del evento anterior, este evento compuesto ocurre cuando ha tomado lugar al menos una vez el evento e_1 dentro del *intervalo*, sin embargo, en este caso, se considera a la última ocurrencia de e_1 dentro del *intervalo* especificado.

Historia, $times(n, e_1)$ en *intervalo*, ocurre cuando el evento e_1 ha ocurrido n veces dentro del *intervalo* especificado.

Alguno, $any(m, e_1, e_2, \dots, e_n)$, $m < n$, este evento compuesto ocurre cuando han ocurrido m eventos e_i distintos que pertenecen al conjunto $\{e_1, e_2, \dots, e_n\}$.

En la detección de eventos compuestos otro problema que debe tomarse en cuenta es qué eventos serán considerados para formar un evento compuesto si los eventos que lo constituyen ocurren más de una vez. Por ejemplo, para el evento compuesto $seq(e_1, e_2)$, supongamos que se dan dos ocurrencias del evento e_1 , e_1^1 y e_1^2 , y posteriormente ocurre el evento e_2 , e_2^1 . Las posibilidades para generar el evento compuesto *secuencia*, a partir de la ocurrencia de los eventos e_1 y e_2 serían $sec(e_1^1, e_2^1)$, $sec(e_1^2, e_2^1)$, o bien, la unión de las dos posibilidades anteriores. La manera en que debe de considerarse la ocurrencia de eventos debe especificarse mediante *políticas de consumo de eventos*. En [28] se proponen cuatro contextos para el consumo de eventos:

- **Contexto reciente:** Considera al conjunto de eventos más recientes que hayan ocurrido para formar al evento compuesto. Para el caso de ejemplo anterior, la posibilidad que se tomaría sería $sec(e_1^2, e_2^1)$, omitiéndose a e_1^1 y e_2^1 para futuras composiciones del evento compuesto *secuencia*.
- **Contexto cronológico:** En el cual se consumen a los eventos en orden cronológico. En nuestro ejemplo, la alternativa que se tomaría para este contexto sería $sec(e_1^1, e_2^1)$, y en futuras composiciones del evento compuesto *secuencia* ya no se considera a e_1^1 y e_2^1 .
- **Contexto continuo:** En este tipo de contexto, para cada evento constituyente e , del evento compuesto, que toma lugar se comienza con la composición de un nuevo evento compuesto. Utilizando este contexto, en el ejemplo se tomaría a $sec(e_1^2, e_2^1)$ y $sec(e_1^1, e_2^1)$ como la ocurrencia de dos eventos compuestos distintos.
- **Contexto acumulado:** A diferencia de los anteriores, en este contexto se toma al conjunto de ocurrencias de eventos en lugar de cada uno de ellos, es decir, se acumulan todos los eventos constituyentes hasta que se forme el evento compuesto. Tomando nuevamente nuestro ejemplo, en un contexto acumulado, el evento compuesto *secuencia* estaría formado por ambas ocurrencias de e_1 y la ocurrencia de e_2 , es decir $sec(\{e_1^1, e_1^2\}, e_2^1)$.

Solamente en el caso del contexto continuo, la ocurrencia de un evento constituyente e puede participar en la composición de dos o más eventos compuestos. Para tener a la mano la información sobre los eventos que han ocurrido, es necesario el uso de base de datos temporales, donde se almacenen todos los cambios de estado que ha sufrido la BD con el paso del tiempo.

Condición La parte condicional de una regla es una expresión lógica la cual determina si la acción de la regla será ejecutada. La condición se evalúa contra el estado que guarde el estado de la BD en el momento en que el evento de la regla a ocurrido, si la evaluación de dicha expresión resulta verdadera entonces la acción ó conjunto de acciones especificadas en la acción de la regla serán ejecutadas.

En algunos sistemas, la especificación de la condición es opcional, formando de esta manera reglas de tipo Evento-Acción, donde a partir de la ocurrencia del evento se dispara la regla y la acción es ejecutada [5].

Acción La acción de una regla ECA describe el conjunto de tareas que serán realizadas si la evaluación de la parte condicional de la regla resulta verdadera.

La acción de la regla puede ser la modificación de la estructura en la BD (operaciones de modificación, inserción ó eliminación de datos), el envío de una señal a un dispositivo externo a la BD, informar a un usuario o al administrador del sistema sobre alguna situación que esté ocurriendo en el estado de la BD, abortar una transacción o bien, tomar alguna acción alternativa utilizando el comando **do-instead**. [15]

La acción de la regla puede a su vez ser el evento que active alguna regla ECA o formar parte de un evento compuesto que active a una regla ECA.

Además de especificar en un SBA el modelo de conocimiento correspondiente a los elementos necesarios para especificar correctamente a cada uno de ellos, es de vital importancia saber la manera en que serán manejados en tiempo de ejecución, por lo cual se hace necesaria la definición de un *modelo de ejecución*.

2.2.2. Modelo de ejecución

Además de un modelo para la definición de cada una de los atributos y características con que deben contar los elementos de la regla ECA en el *modelo de conocimiento*, también se cuenta con un *modelo de ejecución* para el SBDA.

El *modelo de ejecución* es la manera en que la base de reglas ECA serán manejadas en tiempo de ejecución, es decir, como se detectarán los eventos para activar reglas, realizar la verificación de la parte condicional de las reglas activadas, así como la ejecución de las acciones dentro o fuera de

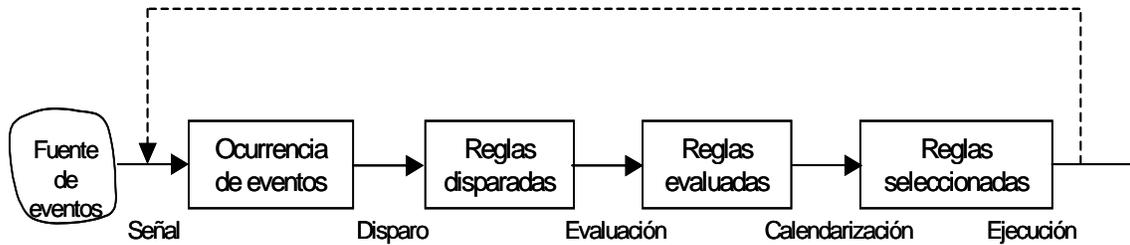


Figura 2.7: Principales pasos que toman lugar durante la ejecución de reglas.

la BD.

El modelo de ejecución está muy relacionado con cada SBDA, pero de manera general en todos los casos siguen las mismas fases en la ejecución de las reglas ECA [5].

La primera fase se refiere a la aparición de los eventos que dispararán a las reglas (*signaling*). En la segunda fase se toman los eventos encontrados en la fase anterior y dispara a aquellas reglas cuyo evento de la regla hayan ocurrido (*triggering*). La tercera fase realiza la evaluación de la parte condicional de la regla (*evaluation*), formando un conjunto con aquellas reglas cuya evaluación de la condición haya resultado verdadera. En la cuarta fase (*scheduling*) se toma el conjunto de reglas formado en la tercera fase y se decide el orden en que serán procesadas. Finalmente, en la última fase (*execution*) se llevan a cabo las acciones correspondientes a las reglas disparadas. Figura 2.7. Las acciones ejecutadas en la última fase, pueden a su vez generar eventos a ser considerados dentro de la primera etapa, produciendo un disparo de reglas en cadena.

El modelo de ejecución determina el momento en que una regla ECA, cuyo evento activador ha ocurrido, será disparada; además del momento en que su parte condicional será evaluada; así como el tiempo en que la acción será ejecutada. Existen tres tipos de *modos de acoplamiento* utilizados más frecuentemente entre los elementos de la regla ECA **evento-condición** y **condición-acción** [27] [5]:

- *Modo inmediato*.- En esta modalidad, la segunda parte de la pareja de elementos de regla ECA se lleva a cabo inmediatamente después de que la primera es terminada. Es decir, en **evento-condición**, la condición es evaluada inmediatamente después de que el evento es detectado; y en **condición-acción**, la acción es ejecutada inmediatamente después de que la

condición es evaluada.

- *Modo pospuesto.*- Esta modalidad es utilizada cuando las operaciones del SBD se hacen mediante transacciones. En esta modalidad, la segunda parte de la pareja de elementos de regla ECA se lleva a cabo hasta que la transacción donde se ocurrió la primera ha terminado. Es decir, en **evento-condición**, la condición es evaluada después de que termina la transacción donde el evento es detectado; y en **condición-acción**, la acción es ejecutada después de que termina la transacción donde la condición es evaluada.
- *Modo separado.*- Al igual que en la modalidad anterior, ésta también es utilizada cuando las operaciones del SBD se realiza mediante transacciones. En el *modo separado*, la segunda parte de la pareja de elementos de regla ECA se lleva a cabo inmediatamente después de la primera es terminada, sin embargo, se realiza en una transacción distinta. Es decir, en **evento-condición**, la condición es evaluada inmediatamente después de que el evento es detectado, pero en una transacción distinta a aquella donde el evento es detectado; y en **condición-acción**, la acción es ejecutada inmediatamente después de que la condición es evaluada, pero en una transacción distinta a la de la evaluación de la condición.

Ya teniendo una definición de los modelos de conocimiento y de ejecución que un SBDA debe ofrecer, es necesario observar algunas de las vicisitudes que se presentan durante el desarrollo de base de reglas ECA. En este sentido es necesario realizar análisis de las bases de reglas ECA.

2.2.3. Análisis de reglas ECA

Durante el procesamiento, las reglas interactúan de diferentes maneras, desde formas muy simples de interacción, hasta comportamientos complejos de interrelación entre los elementos de la regla ECA.

Debido a esto, el desarrollo de una base de reglas ECA suele ser una tarea muy complicada de realizar, debido a la naturaleza impredecible que tienen las reglas ECA durante su procesamiento.

El análisis de reglas ECA nos ayuda a determinar, hasta cierto punto, el comportamiento que presentará una determinada base de reglas ECA en ciertos estados de la BD. Este análisis puede llevarse a cabo durante la etapa de desarrollo de la base de reglas ECA (*análisis estático*), o bien, durante la ejecución de la base de reglas obre la BD (*análisis dinámico*).

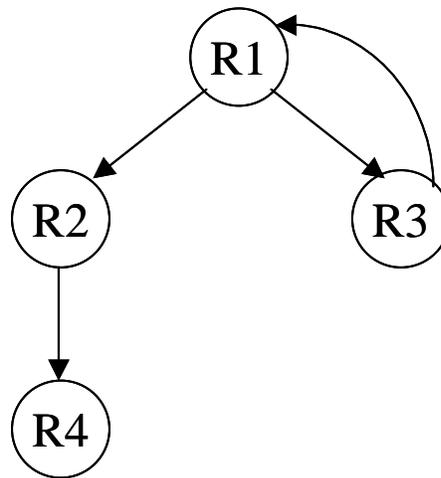


Figura 2.8: Diagrama del disparo infinito de reglas, que produce el problema de No-Terminación.

Existen, principalmente, dos propiedades que toda base de reglas debe ofrecer, las cuales son **Terminación** y **Confluencia**. En términos generales, éstas propiedades suelen ser difíciles de determinar en una base de reglas ECA.

Terminación

La propiedad de *Terminación* garantiza que el disparo en cadena de un conjunto de reglas ECA llega a terminar, es decir, si para algún estado inicial de la BD durante el procesamiento de reglas, se garantiza que el disparo de reglas no se realiza de manera infinita.

En la figura 2.8 se muestra gráficamente el problema que existe si una base de reglas ECA no cumple con la propiedad de Terminación. En la cual, a partir del disparo de la regla $R1$ se produce el disparo de las reglas $R2$ y $R3$. Siguiendo la secuencia de disparo de reglas para $R2$, se observa que $R2$ a su vez dispara a la regla $R4$, terminando en ese punto el disparo de reglas en cadena. Sin embargo, por el otro lado tenemos que el disparo de $R1$ produce el disparo de $R3$, pero el disparo de $R3$ produce el disparo, nuevamente, de $R1$; obteniendo con esto un disparo infinito de las reglas $R1$ y $R3$.

Este procesamiento infinito de reglas utiliza mucha capacidad de procesamiento del equipo de cómputo, volviendo inestable al SBD.

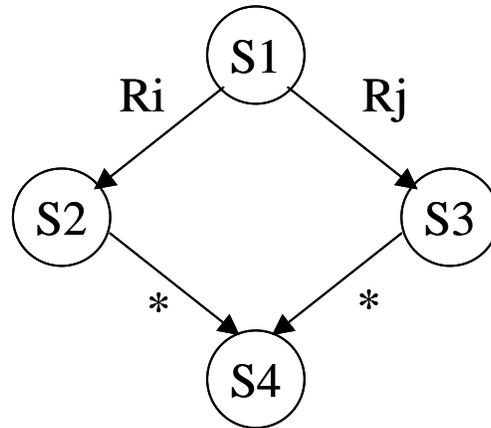


Figura 2.9: Confluencia en el disparo de un conjunto de reglas ECA.

Confluencia

La propiedad de *Confluencia* garantiza que a partir de un estado inicial de la BD y un conjunto de reglas R cuyo evento activador ha ocurrido, el estado final de la BD es el mismo independientemente del orden en que las reglas $r \in R$ sean disparadas. Es decir, un conjunto de reglas es confluente si siempre se obtiene un estado final único después del disparo de un conjunto de reglas.

Considerando que el disparo de una regla produce un nuevo estado de la BD; si más de una regla es disparada al mismo tiempo, entonces tendríamos más una posibilidad de estado de la BD. Como se muestra en la figura 2.9, se tiene un estado inicial $S1$, a partir del cual se disparan las reglas R_i y R_j obteniendo los estados $S2$ y $S3$ respectivamente. Una base de reglas ECA es confluente si se tiene un conjunto de reglas listas para dispararse R y para cualesquier regla $R_i, R_j \in R$ disparada en el estado inicial $S1$, se garantiza que siempre se obtendrá el mismo estado final $S4$, sin tener en cuenta el orden en que sean disparadas el resto de las reglas (representadas por $*$ en la figura 2.9) en R , $\{R_i \mid R_j\} - R$.

2.3. Sintaxis de reglas en BDA existentes

Existen diversos manejadores de BDA que ofrecen una sintaxis para la definición de reglas ECA. Entre los que podemos mencionar se encuentra el Postgres, Ariel, Starburst, A-RDL, Chimera,

HiPAC, Ode y algunos sistemas de BD comerciales como Oracle, Informix, Ingres, Rdb, Allbase/SQL e InterBase.

2.3.1. Postgres

Postgres ofrece un lenguaje de consulta denominado POSTQUEL, el cual está basado en el lenguaje de consulta QUEL de INGRES, al que se le hicieron algunas extensiones y modificaciones para que pudiera aceptar las nuevas características de Postgres [15].

En Postgres existen dos posibles implementaciones de reglas. La primera implementación, el *Sistema a Nivel de Tuplas (TLS, Tuple Level System)*, procesa las reglas en cada una de las tuplas que pertenecen a una relación. Este sistema utiliza marcadores especiales denominados *candados de reglas*. Cuando una regla es definida bajo esta implementación, los candados de reglas son almacenados en las tuplas que satisfacen los requisitos definidos en la regla (es decir, la parte condicional). Cuando un evento apropiado ocurre en una tupla que tiene el candado de regla apropiado, la acción correspondiente es ejecutada. [17]

En la segunda forma de implementación, el *Sistema de Rescritura de Consultas (QRS, Query Rewrite System)*, está basado en un mecanismo en el que se rescriben las consultas. En este enfoque, una consulta es combinada con todas las tuplas relevantes y se genera un conjunto con todas las consultas modificadas. La ejecución de este nuevo conjunto de consultas proporciona el resultado deseado. [17]

La sintaxis para la definición de reglas en Postgres es la siguiente.

```
define [tuple | rewrite] rule nombre_regla is
on evento to objeto
where predicado
do [instead] acción
```

El *evento* puede ser alguna operación generada por un comando en POSTQUEL: *retrieve*, *append*, *delete* o *replace*. El *objeto* se refiere a una relación (por ejemplo *Empleado*) o un atributo de una relación (por ejemplo *Empleado.antigüedad*). El *predicado* es una expresión de evaluación del estado de la BD expresado en POSTQUEL y la *acción* es un conjunto de operaciones denotadas con comandos de POSTQUEL o el comando especial *abort*. Si el evento especificado ocurre dentro del objeto, y si el predicado de evaluación es verdadero, entonces se ejecuta la acción de la regla.

Si la acción de la regla ejecuta el comando *abort*, la transacción es cancelada. Las palabras clave *tuple* y *rewrite* especifican cual tipo de implementación de regla que será utilizado, TLS o QRS respectivamente. Si no se especifica ninguna de las dos palabras clave, por default se toma al tipo TLS.

2.3.2. SAMOS

SAMOS es un proyecto de investigación desarrollado en la Universidad de Zurich, en Suiza. SAMOS es un acrónimo de *Swiss Active Mechanism Based Object Oriented Database Systems*. También podría considerarse el nombre de la isla griega donde el filósofo y matemático Pitágoras vivió.

SAMOS agrega características de una BDA en un modelo de datos orientado a objetos, en la cual conceptos como clases, objetos y herencia son considerados para su diseño. Además del lenguaje de definición de datos que proporciona la BD sobre la que trabaje SAMOS, éste provee un lenguaje de definición de reglas para la especificación de reglas ECA. [38]

La arquitectura de SAMOS persigue una arquitectura de capas, el cual es colocado como una capa superior sobre un sistema de BD orientado a objetos. La implementación de SAMOS se realizó en C++, sobre un Sistema Manejador de Base de Datos Orientado a Objetos comercial, el ObjectStoreTM, el cual también está implementado en C++.

La sintaxis de SAMOS para la definición de reglas tiene la siguiente forma:

```
DEFINE RULE nombre_regla
ON clausula_evento
IF condición
THEN acción
COUPLING MODE (coupling, coupling)
PRIORITIES (BEFORE | AFTER) nombre_regla
```

En esta definición de regla se especifica una descripción del evento que será monitoreado, una condición, una acción y restricciones para la ejecución de las reglas (prioridades y modos de acoplamiento). [38]

La parte correspondiente a la condición es opcional, mientras que el evento y la acción de la regla son obligatorios, es decir, en SAMOS pueden definirse reglas de tipo EA (*evento-acción*).

SAMOS utiliza el parámetro **same transaction**, para especificar que en la composición de eventos compuestos, cada uno de sus componentes hayan ocurrido en la misma transacción.

SAMOS soporta el modo de consumo de eventos cronológico. Los autores argumentan que el modo de consumo continuo puede modelarse en SAMOS como un evento compuesto definido con un intervalo de monitoreo.

SAMOS no verifica la confluencia, debido a que utiliza prioridades en reglas y esto evita que a partir del disparo de un conjunto de reglas, se obtengan estados finales distintos.

2.3.3. Ariel

El sistema Ariel es una implementación de un SBD relacional con un sistema para la construcción de reglas. El sistema de reglas de Ariel está basado en el modelo de sistemas de producción [10]. El mecanismo para la verificación de la parte condicional de las reglas en Ariel, denominado A-TREAT, es una variación del mecanismo TREAT presentado en [19].

Ariel ofrece un subconjunto del lenguaje de consultas POSTQUEL, de POSTGRES, para la especificación de comandos de definición, consultas y actualizaciones de datos. Ariel soporta los comandos de POSTQUEL **retrieve**, **append**, **delete** y **replace**, además de otros comandos utilizados en la creación y destrucción de tablas e índices, carga de relaciones, entre otras.

El lenguaje para la definición de reglas en Ariel es un lenguaje mejorado de reglas de producción para la definición de reglas con condiciones que pueden contener las operaciones del álgebra relacional *select* y *join*, así como la especificación de eventos y transiciones. La sintaxis del lenguaje de reglas de Ariel está basado en la sintaxis del lenguaje de consultas. La forma general de una regla en Ariel es la siguiente: [18]

```
define rule nombre_regla [in nombre_conjunto_reglas]
  [priority valor_prioridad]
  [on evento]
  [if condición]
then acción
```

Se requiere de un nombre de regla único para cada regla, porque las reglas son referenciadas posteriormente por el usuario. El nombre del conjunto de reglas en el que asignará la regla es opcional; los conjuntos de reglas se utilizan como una manera de agrupamiento de reglas, las cuales

pueden manejarse simultáneamente. Si no se especifica un nombre de conjunto de reglas cuando una regla es definida, la regla se asigna en el conjunto de reglas definidas por el sistema *default_rules*.

La cláusula **priority** también es opcional y permite la especificación de una prioridad numérica para controlar el orden de ejecución de reglas, cuando muchas reglas se seleccionan para ejecutarse.

La cláusula **on** permite la especificación del evento que disparará la regla. La parte condicional de la regla, la cláusula **if**, tiene una sintaxis semejante a la cláusula **where**, utilizada para las consultas a la BD, con algunas modificaciones. La cláusula **then** de la regla contiene la acción de la regla que será ejecutada cuando la regla dispare.

2.3.4. Starburst

El sistema *Starburst* es un prototipo de un SBD relacional extendido desarrollado en el Centro de Investigaciones de Almaden - IBM[14]. La extensión que tiene Starburst permite que el SBD sea personalizado para aplicaciones avanzadas y no tradicionales de BD. Una de las extensiones que ofrece Starburst es un sistema de procesamiento de reglas activas denominado *Sistema de Reglas de Starburst*.

El lenguaje para la definición y manipulación de reglas ECA en Starburst consiste de cinco comandos: **create rule**, **alter rule**, **deactivate rule**, **activate rule** y **drop rule**. El comando **create rule** es utilizado para la definición de una regla nueva. La sintaxis de este comando es: [20]

```
create rule nombre on tabla
when operaciones_de_disparo
[if condición]
then lista_de_acciones
[precedes lista_de_reglas]
[follows lista_de_reglas]
```

El parámetro *nombre* es el nombre distintivo de cada regla, y cada regla debe definirse sobre una *tabla*. La cláusula **when** especifica uno o más eventos, algunos de los cuales será el que dispare a la regla (no es más que el evento compuesto *disyunción*). Los eventos posibles que dispararían a una regla corresponden a las tres operaciones de modificación del álgebra relacional: *insertar*, *eliminar* y *actualizar*. En el caso de un evento de actualización, debe especificarse, además de la tabla, el nombre de la columna que sufrirá la modificación, si no se especifica ninguna columna,

entonces la regla se dispara cuando sea modificada cualesquier columna de la tabla.

La cláusula **if** especifica la condición de la regla, la cual se especifica como una sentencia de SQL. La condición es verdadera si el resultado de la sentencia de SQL produce como resultado una o más tuplas. Si ésta cláusula es omitida, se considera que la cláusula **if** siempre será verdadera.

La cláusula **then** especifica la acción de la regla. Cada acción puede ser alguna operación sobre la BD, incluyendo comandos de manipulación de datos de SQL (**select**, **insert**, **delete**, **update**), comandos de definición de datos (por ejemplo, **create table**, **drop rule**) y el comando **rollback**. Las acciones son especificadas como una lista que serán ejecutadas en el orden en que sean denotadas.

Las cláusulas opcionales **precedes** y **follows** son usadas para especificar el orden de prioridad entre las reglas.

2.3.5. A-RDL

El proyecto de investigación A-RDL es una continuación del proyecto RDL1, cuyo objetivo principal fue el desarrollo de un lenguaje de reglas de producción para BD.

El lenguaje A-RDL tiene las siguientes características. Primero, combina los módulos de reglas que derivan datos de otros datos y módulos de reglas activas que reaccionan a eventos de modificación de datos. Segundo, los eventos que disparan a las reglas activas no necesitan ser proporcionados explícitamente por el usuario, sino que deben ser generados automáticamente por el sistema. Tercero, la semántica del lenguaje está definida formalmente usando operadores de punto fijo. [21]

La descripción de la forma general de definición de reglas utiliza la forma BNF. Dentro de ésta gramática, los símbolos no terminales están encerrados entre los símbolos \langle y \rangle ; los nodos terminales, que son palabras reservadas del lenguaje están en negritas; las opciones alternativas dentro de la regla se especifican mediante el símbolo $|$; la notación $[e]$ manifiesta que la expresión e es opcional; y la notación $\{e\} \dots$ manifiesta que la expresión e se repite una o mas veces.

En la siguiente tabla se describe la sintaxis utilizada en la definición de reglas para A-RDL:

```

<rule_definition>  rules [<coupling_mode>] <rule> [{,<rule>}...]
                    rule_name is [{c_code}] [<coupling_mode>]
                    if <condition_part>
                    <then_mode> <action_part> {c_code};

<then_mode>       then | thenone
<coupling_mode>  immediate | deferred

```

Cada regla tiene un nombre distintivo y consiste de una declaración **if-then**. Los modos de acoplamiento pueden ser definidos en una sola ocasión para todas las reglas después de la palabra reservada **rules** o puede definirse para cada regla después de la palabra reservada **is**. A-RDL solamente soporta dos modos de acoplamiento, el modo inmediato y el modo pospuesto.

La sintaxis de reglas acepta la utilización de código escrito en lenguaje C, el cual es útil para realizar un trazado de la ejecución de programas de reglas, establecer relaciones con los usuarios o para asignar valores a las variables.

La parte del **if** en una regla, o parte condicional, es una expresión del cálculo relacional de tuplas. La parte de **then** en una regla, o parte de la acción, es un conjunto de acciones elementales, tales como actualizaciones a la BD, una sentencia de **rollback**, una asignación a alguna variable o una llamada a un procedimiento. La palabra reservada **then** especifica la manera tradicional en que la cláusula then es utilizada, sin embargo, si se especifica la palabra reservada **thenone** especifica que la regla solamente será disparada una sola vez durante el procesamiento de las reglas.

2.3.6. Chimera

Chimera es un lenguaje de BD que está conformado por un modelo de datos orientado a objetos, un lenguaje de consultas declarativo basado en reglas deductivas y un lenguaje de reglas activas para el procesamiento reactivo [5]. Chimera utiliza reglas activas orientadas a conjuntos, las cuales son activadas por el efecto de varios eventos, lógicamente distinguibles, que afectan a múltiples objetos instancias de clase.

Las reglas activas en Chimera son denominadas *triggers* o disparadores. La sintaxis para la definición de reglas en Chimera es la siguiente: [22]

```

trigger_rule ::=
    "define" [opciones_de_disparo] "trigger" nombre_trigger ["for" nombre_clase]

```

```
“events” eventos_de_disparo
“condition” formula_de_condición
“action” reacción
[opción_de_prioridad]
“end”
```

Las reglas activas de Chimera pueden definirse en el contexto de una sola clase o en el contexto de múltiples clases. Cada regla activa consiste de cuatro componentes: eventos, condiciones, acciones y la prioridad.

Los eventos indican las operaciones primitivas que son monitoreados por las reglas activas, los cuales son denotados por el nombre de la operación primitiva y el elemento donde la operación es aplicada. Las operaciones primitivas son consultas, creación, modificación y eliminación de objetos, migración de objetos dentro de jerarquías de generalización y el cambio del estado de persistencia de los objetos.

La condición es una conjunción de fórmulas atómicas, las cuales son interpretadas como como una expresión del cálculo de predicados sobre variables tipeadas. La condición es evaluada contra el estado de la BD y eventos relevantes. Chimera soporta también el valor para la condición *true*, que denota que la condición siempre será verdadera.

La reacción es una secuencia de llamadas a procedimientos, incluyendo primitivas del lenguaje de modificación de datos y despliegue de resultados, procedimientos definidos externamente y comandos de transacción como *savepoint* y *rollback*.

Para controlar la ejecución de un conjunto de reglas activadas en tiempo de ejecución, se establece una cláusula opcional de prioridad. En esta cláusula se indican las reglas que tendrán prioridades menores o mayores que la regla que se está definiendo, con esta especificación de prioridad se establece un orden parcial de ejecución de reglas.

2.3.7. HiPAC

HiPAC es un SMBD orientado a objetos, el cual extiende a un SMBD orientada a objetos básico con la definición de reglas ECA. Las reglas ECA, al igual que otras formas de datos en HiPAC, son tratadas como entidades. Existe un tipo de entidad denominada **rule** para denotar a la entidad para las reglas y cada regla es una instancia de este tipo de entidad. Funciones especiales son definidas

sobre el tipo de entidad **rule** para **disparar**, **activar** y **desactivar** reglas.[13]

El tipo de entidad **rule** está definida como un subtipo de un tipo más genérico denominado **entity**. El evento, la condición y la acción de una regla están definidos por tres funciones del tipo de entidad **rule**: [23]

- **event**, mapea la entidad **rule** a una entidad **event**.
- **condition**, mapea la entidad **rule** hacia una pareja de elementos conformada por un modo de acoplamiento y un conjunto de consultas.
- **action**, mapea la entidad **rule** hacia una pareja de elementos conformada por un modo de acoplamiento y un programa.

Las consultas utilizadas en la parte de la condición de la regla son realizadas mediante el lenguaje de consultas de HiPAC y los programas usados para la acción de la regla pueden ser escritos en alguno de los lenguajes que manejan comandos de manipulación de datos en BD.

Los modos de acoplamiento que ofrece HiPAC son inmediato, pospuesto, separado y casualmente-dependiente-separado.

2.3.8. Ode

La BD orientada a objetos Ode está basada en el paradigma de programación orientada a objetos de C++. La interfaz primaria de la BD Ode es el lenguaje de programación de BD O++, el cual es una extensión compatible de C++. O++ ofrece facilidades adecuadas para el desarrollo de aplicaciones de SBD, incluyendo la asociación de restricciones y disparadores (triggers) con objetos. [71]

Los disparadores están asociados con los objetos y son activados explícitamente después de que un objeto ha sido creado. Un disparador T_i asociado con un objeto cuyo identificador es *object-id* es activado por la llamada *object-id* $\rightarrow T_i(\text{argumentos})$. La activación del disparador regresa un identificador para el disparador si ésta fue exitosa, en caso contrario regresa un valor nulo (null).

Un disparador activado “dispara” cuando su predicado llega a ser verdadero. Posteriormente, la acción asociada con el disparador es ejecutada en una transacción diferente. Los disparadores pueden ser desactivados explícitamente antes de que sean disparados utilizando la función *deactivate(identificador_disparador)*. [24]

2.3.9. Oracle

Oracle soporta disparadores que son ejecutados antes o después de la operación de disparo. Un disparador puede monitorear múltiples operaciones en la misma tabla, pero cada operación (incluyendo las actualizaciones a una columna) puede ser monitoreada solamente por un solo disparador. La sintaxis para la creación de disparadores en Oracle es la siguiente: [8]

```
<disparador> ::= {CREATE | REPLACE} TRIGGER <nombre del disparador>
                {BEFORE | AFTER} <eventos del disparador>
                ON <nombre de tabla>
                [[REFERENCING <referencias>]]
                FOR EACH ROW
                [WHEN <condición>]]
                <PL/SQL procedimiento>
```

```
<evento> ::= INSERT | DELETE | UPDATE [OF <nombres de columnas>]
```

```
<referencia> ::= OLD AS <nombre para el valor anterior de la tupla>
                NEW AS <nombre para el nuevo valor de la tupla>
```

La condición es soportada solamente en conjunción con la opción `FOR EACH ROW` y está restringida a ser un simple predicado en la tupla modificada. La acción disparada es un procedimiento escrito en el lenguaje PL/SQL, un lenguaje de programación de BD soportado por Oracle. Las referencias `OLD` y `NEW` se utilizan para denotar el valor de un atributo antes de la modificación (`OLD`) y después de la modificación (`NEW`).

2.3.10. Informix

Informix proporciona disparadores que son definidos con una sintaxis no estándar, donde varios disparadores pueden definirse dentro de una sola regla. La definición de un trigger en INFORMIX se realiza como se muestra en el siguiente ejemplo:

```
CREATE TRIGGER nombreTrigger UPDATE salario, rango ON Empleado
FOR EACH ROW(EXECUTE PROCEDURE actualizacion)
```

2.4. Comentarios

Un SBD está conformado por una BD, o repositorio de datos, y por un conjunto de programas, encargados de manipular los datos almacenados. Una BD es el conjunto de datos que tienen una relación entre sí y mantienen un significado implícito. En una BD se busca representar una parte del mundo real al que se le denomina minimundo o universo de discurso.

Entre los modelos de datos utilizados en la definición de una BD están los modelos lógicos basados en objetos (modelo Entidad-Relación, modelo orientado a objetos, modelo de datos semánticos y modelo de datos funcionales), modelos lógicos basados en registros (modelo relacional, modelo de red, modelo jerárquico) y los modelos físicos.

Sin embargo, existen universos de discurso que no pueden ser modelados con el esquema tradicional de BD, porque necesitan de una reacción automática del propio SBD, ante la ocurrencia de eventos, para realizar ciertas acciones dentro y fuera de la BD. Por tal motivo se introdujeron los Sistemas de Bases de Datos Activas (SBDA).

Los SBDA son útiles cuando se desea representar un minimundo que esté controlado por la ejecución y disparo automático de acciones. Un SBDA se logra mediante la conjunción de un sistema tradicional, o pasivo, de BD mas un conjunto de reglas activas.

Un SBDA ofrece un modelo de conocimiento y un modelo de ejecución. El modelo de conocimiento indica al SBDA la manera en que debe de comportarse ante la presencia de eventos relevantes y al estado que guarde la BD en un momento determinado. El enfoque más utilizado para el modelo de conocimiento de un SBDA es a partir del modelo de reglas Evento-Condición-Acción ó reglas ECA. Por otro lado, el modelo de ejecución indica la manera en que la base de reglas ECA será manejada en tiempo de ejecución, es decir, cómo serán detectados los eventos que activarán a las reglas, cómo se realizará la verificación de la parte condicional de la regla y la manera en que serán ejecutadas las acciones de la regla.

Durante el desarrollo de las reglas ECA pueden presentarse problemas como el de No terminación, en donde se da el disparo infinito de un conjunto de reglas; y el de confluencia, donde a partir de un conjunto de reglas, el estado final de la BD depende del orden en que las reglas sean ejecutadas. Estos tipos de problemas llevan a un SBD a un estado inconsistente.

Existen diferentes SBDA que ofrecen una sintáxis para la definición de reglas, sin embargo cada uno de ellos dependen del ambiente y de la BD pasiva sobre la que definen las reglas.

Capítulo 3

Conceptos básicos de Redes de Petri

Esta propuesta doctoral está basada en teoría de redes de Petri (PN's) [30] [32] [31] [33] [35] [29] [37], la cual nos ofrece un ambiente gráfico y matemático para realizar la modelación de sistemas manejados por eventos y el análisis de sus propiedades y comportamiento, razón por la cual es factible su uso dentro del campo de las BDA's.

En este capítulo se presentan los conceptos fundamentales de la teoría de PN's, posteriormente se presenta el concepto de la red de Petri Coloreada (CPN) [68], de la cual se tomaron algunos elementos para formar el modelo propuesto.

3.1. Definición

El concepto de PN tiene su origen en el trabajo de tesis de Carl Adam Petri, que presentó en el año de 1962 en la Facultad de Física y Matemáticas de la Universidad Técnica de Darmstadt, de la antigua República Federal Alemana. En ese trabajo, C.A. Petri estableció las bases para una teoría de comunicación entre componentes asíncronos de un sistema de cómputo. Definió una herramienta matemática de propósito general para describir las relaciones que existen entre condiciones y eventos. [30]

Una PN es un tipo particular de grafo dirigido bipartito, compuesto por dos tipos de objetos. Estos objetos son **lugares y transiciones**, los arcos que los unen están dirigidos de lugares a transiciones o de transiciones a lugares. Gráficamente, los lugares son representados por círculos y

las transiciones son representadas por barras o por rectángulos. En su forma más simple, una PN se representa por una transición unida con su lugar de entrada y su lugar de salida. Esta red básica se utiliza para representar varios aspectos de un sistema que se pretende modelar. Con la finalidad de estudiar el comportamiento dinámico de los sistemas modelados, desde el punto de vista del estado que presentan en un momento dado y los cambios de estado que pueden ocurrir, cada lugar puede no tener tokens o tener un número positivo de tokens. Los tokens se representan gráficamente por pequeños círculos rellenos. La presencia o ausencia de un token dentro de un lugar indica si una condición asociada con este lugar es falsa o verdadera, ó también nos indica si un dispositivo que se está modelando se encuentra disponible o no.

Formalmente, una PN puede definirse como sigue: [32]

Definición 3.1 Una PN es una 5-tupla, $PN = \{P, T, F, W, M_0\}$, donde

$P = \{p_1, p_2, \dots, p_m\}$ es un conjunto finito de lugares.

$T = \{t_1, t_2, \dots, t_n\}$ es un conjunto finito de transiciones.

$F \subseteq (P \times T) \cup (T \times P)$ es un conjunto de arcos (relación de flujo)

$W : F \rightarrow \{1, 2, 3, \dots\}$ es una función de peso.

$M_0 : P \rightarrow \{0, 1, 2, 3, \dots\}$ es el marcado inicial.

$$P \cap T = \emptyset \text{ y } P \cup T \neq \emptyset.$$

El *marcado* es la asignación de *tokens* a los lugares de una PN. Un token es un concepto primitivo que forma parte de las PN's (como los lugares y las transiciones), el cual puede ser asignado a un lugar de la red de Petri para denotar el estado del sistema modelado. Los tokens son usados para definir la ejecución de una PN, por lo tanto, la cantidad y la posición de los tokens puede cambiar durante la ejecución.

Algunas interpretaciones típicas de las transiciones y de sus lugares de entrada y de salida se muestran en la tabla VI. [32]

Lugares de Entrada	Transiciones	Lugares de Salida
Precondiciones	Evento	Postcondiciones
Datos de entrada	Procesamiento	Datos de salida
Señales de entrada	Procesamiento de señales	Señales de salida
Requerimiento de recurso	Tarea	Liberación del recurso
Condiciones	Cláusula en lógica	Conclusiones
"Búfer"	Procesador	"Búfer"

Tabla VI. Algunas interpretaciones de transiciones y lugares.

La PN es una herramienta de modelación gráfica y matemática que puede aplicarse a muchos sistemas. Las PN pueden realizar la descripción y el análisis de sistemas que procesan información, los cuales se caracterizan por ser concurrentes, asíncronos, distribuidos, paralelos, indeterministas, y/o estocásticos [32]. Para la modelación de éstos sistemas, podemos utilizar las PN de dos maneras: gráficamente y matemáticamente.

Gráficamente las PN pueden utilizarse como una ayuda de comunicación visual parecidas a los diagramas de flujo, diagramas de bloques y redes. Además, los tokens son utilizados en este tipo de redes para simular las actividades dinámicas y concurrentes de un sistema. Como una herramienta matemática, es posible obtener ecuaciones de estado, ecuaciones algebraicas, y otros modelos matemáticos que gobiernen el comportamiento de los sistemas.

Entonces, considerando a las PN como herramienta gráfica o matemática, y de acuerdo a su definición, una gráfica de PN tiene dos tipos de nodos. Un *círculo* que representa a un lugar (*place*); una *barra* o *rectángulo* que representa una transición (*transition*). Los arcos (ó flechas) dirigidos conectan a los lugares y a las transiciones, algunos arcos se conectan desde lugar hacia una transición, y otros se conectan desde una transición hacia un lugar. Un arco dirigido desde un lugar p_j a una transición t_i define a p_j como un lugar de entrada de t_i , el cual se denota como (p_j, t_i) . Un arco dirigido desde una transición t_i hacia un lugar p_j define a p_j como un lugar de salida de t_i , el cual se describe como (t_i, p_j) . El peso de un arco se denota como $w(p, t)$ ó $w(t, p)$ para especificar el peso de un arco que conecta a $p \in P$ con $t \in T$, ó a $t \in T$ con $p \in P$, respectivamente.

Si $w(p_j, t_i) = k$ (o $w(t_i, p_j) = k$), entonces existen k arcos dirigidos (paralelos) que conectan el lugar p_j con la transición t_i (o que conectan la transición t_i con el lugar p_j). Generalmente, en un esquema gráfico, los arcos paralelos que conectan a un lugar (transición) con una transición (lugar)

se representan por un sólo arco dirigido, etiquetado con el número de arcos que representa, o su peso k . Un círculo que contiene un punto en su interior representa a un lugar que contiene un token. [32]

El conjunto de lugares de entrada para una transición $t \in T$ se representan por el conjunto $\bullet t = \{p \mid p \text{ es un lugar de entrada de } t\}$; y el conjunto de lugares de salida para t se representan por el conjunto $t^\bullet = \{p \mid p \text{ es un lugar de salida de } t\}$. Por otro lado, el conjunto de transiciones que se conectan hacia un lugar $p \in P$ se representa por $\bullet p = \{t \mid p \in t^\bullet\}$; y el conjunto de transiciones hacia donde se conecta p se representa por $p^\bullet = \{t \mid p \in \bullet t\}$.

Ejemplo 3.1 *La figura 3.1 muestra una PN sencilla, en esta PN tenemos:*

$$P = \{p_1, p_2, \dots, p_7\};$$

$$T = \{t_1, t_2, \dots, t_5\};$$

$$w(p_1, t_1) = 2, w(p_i, t_1) = 0, \text{ para } i = 2, 3, \dots, 7;$$

$$w(p_2, t_2) = 1, w(p_7, t_2) = 1, w(p_i, t_2) = 0, \text{ para } i = 1, 3, 4, 5, 6;$$

.....

$$w(t_1, p_2) = 1, w(t_1, p_3) = 2, w(t_1, p_i) = 0, \text{ para } i = 1, 4, 6, 7;$$

$$w(t_2, p_4) = 1, w(t_2, p_i) = 0, \text{ para } i = 1, 2, 3, 5, 6, 7;$$

.....

$$M_0 = (2 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)^T.$$

3.2. Representación de la estructura de la Red de Petri

La estructura de una PN puede ser descrita en términos formales a través de la *matriz de incidencia*. La matriz de incidencia de una PN denota la relación existente entre lugares y transiciones, además de expresar las reglas de habilitación y disparo de transiciones. A partir de una matriz de incidencia puede crearse la PN que la originó.

La *matriz de incidencia* $A = \{a_{ij}\}$ para una PN con n transiciones y m lugares es una matriz de $n \times m$ de enteros y sus entradas están dadas por:

$$a_{ij} = a_{ij}^+ - a_{ij}^-$$

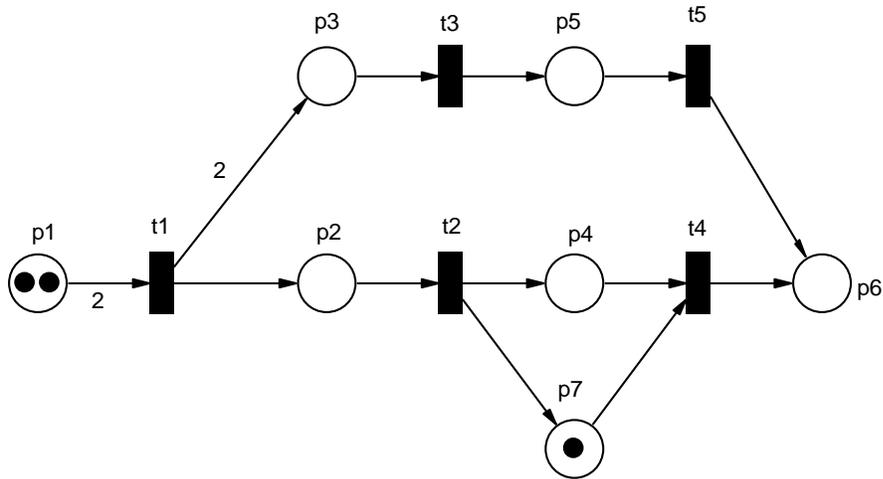


Figura 3.1: Una red de Petri sencilla.

donde $a_{ij}^+ = w(t_i, p_j)$ es el peso del arco desde la transición i a su lugar de salida j y $a_{ij}^- = w(p_j, t_i)$ es el peso del arco a la transición i desde su lugar de entrada j .

Es fácil de observar que a_{ij}^- , a_{ij}^+ , a_{ij} , representan el número de tokens eliminados, agregados, y modificados, respectivamente, en el lugar p_j cuando la transición t_i es disparada. La transición t_i se habilita en una marca M si y sólo si

$$a_{ij}^- \leq M(p_j), j = 1, 2, \dots, m.$$

3.3. Disparo de transiciones

La ejecución de una PN es controlada por el número de tokens y su distribución en la red. Los tokens se alojan en los lugares y controlan el disparo de las transiciones que forman la red. Cambiando la distribución de los tokens en los lugares, podremos estudiar el comportamiento dinámico que puede alcanzar el sistema en diferentes estados. Una PN es ejecutada a través del disparo de transiciones, que se lleva a cabo con la aplicación de la regla de habilitación (*enabling rule*) y posteriormente se aplica la regla de disparo (*firing rule*), las cuales gobiernan el flujo de los tokens por la red. [29]

1. *Regla de habilitación de transiciones:* Una transición t se dice que será *habilitada* si cada lugar de entrada p de t contiene al menos un número de tokens igual al peso k del arco dirigido que conecta a p con t , es decir, que $M(p) \geq w(p, t), \forall p \in \bullet t$.
2. *Regla de disparo de transiciones:*
 - a) Una transición habilitada t puede dispararse o no dependiendo de la interpretación adicional que tenga la transición, y
 - b) El disparo de una transición habilitada t elimina de cada lugar de entrada p el mismo número de tokens que el valor del peso k del arco dirigido que conecta a p con t . Además, agrega a cada lugar de salida p el número de tokens que sea igual al peso k del arco dirigido que conecta a la transición t con el lugar p .

Matemáticamente, el disparo de t en M alcanza una marca nueva:

$$M'(p) = M(p) - w(p, t) + w(t, p), \forall p \in P$$

Solamente las transiciones habilitadas pueden dispararse, cada lugar siempre mantendrá un número no negativo de tokens después que una transición haya sido disparada.

Consideremos ahora las siguientes situaciones que pueden presentarse en una PN: una transición sin ningún lugar de entrada se le llama *transición fuente* (*source transition*) y una sin lugar de salida alguno se le llama *transición sumidero* (*sink transition*). Observemos que una transición fuente siempre está habilitada, y que el disparo de una transición sumidero consume tokens, pero no los produce porque no tiene lugar de salida a quien mandárselos. Una PN formada solamente por un lugar p y una transición t se dice que es *cíclica*, si p es el lugar de entrada, así como el lugar de salida de la transición t . Si una PN no tiene ciclos, entonces se trata de una PN *pura*. [32]

Para mostrar un ejemplo de la habilitación y disparo de reglas tomemos la PN de la figura 3.1. Bajo una marca inicial, $M_0 = (2 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)^T$, solamente t_1 está habilitada. Al realizar el disparo de t_1 obtenemos una nueva marca, digamos M_1 . De acuerdo con la regla disparada obtenemos entonces:

$$M_1 = (0 \ 1 \ 2 \ 0 \ 0 \ 0 \ 1)^T.$$

La distribución de los tokens resultantes se presenta en la figura 3.2. Ahora, a partir de la marca M_1 , las transiciones t_2 y t_3 se encuentran habilitadas. Si se dispara a t_2 se obtiene una nueva marca, digamos M_2 :

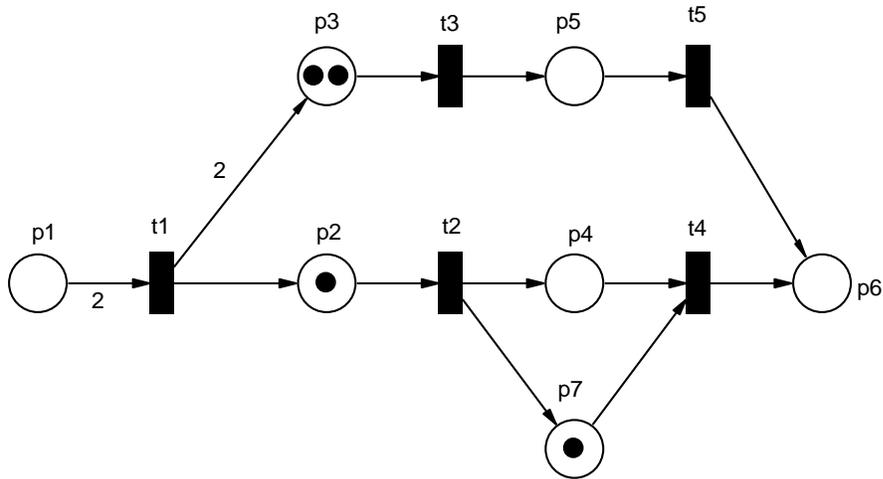


Figura 3.2: Red de Petri resultante al disparar t_1 en la red del ejemplo 1.

$$M_2 = (0 \ 0 \ 2 \ 1 \ 0 \ 0 \ 0)^T.$$

Y si la transición que dispara es t_3 , entonces la nueva marca, digamos M_3 , es:

$$M_3 = (0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1)^T.$$

3.4. Poder de representación de las PN

Las características típicas manifestadas por las actividades de un sistema dinámico de eventos discretos (DEDS), como concurrencia, toma de decisiones, sincronización y prioridades, pueden modelarse efectivamente con PN. A continuación se describen las estructuras de PN para representar las características de las actividades que realizan los DEDS. En la figura 3.3 se muestran estas estructuras. [30]

1. *Ejecución secuencial.* En la figura 3.3(a), la transición t_2 puede disparar solamente después de que t_1 sea disparada. Lo que obliga a que se cumpla la restricción de precedencia " t_2 después t_1 ". Estas restricciones de precedencia son comunes en la ejecución por partes de un DEDS. Además, esta estructura de PN modela la relación causal que existe entre las actividades del

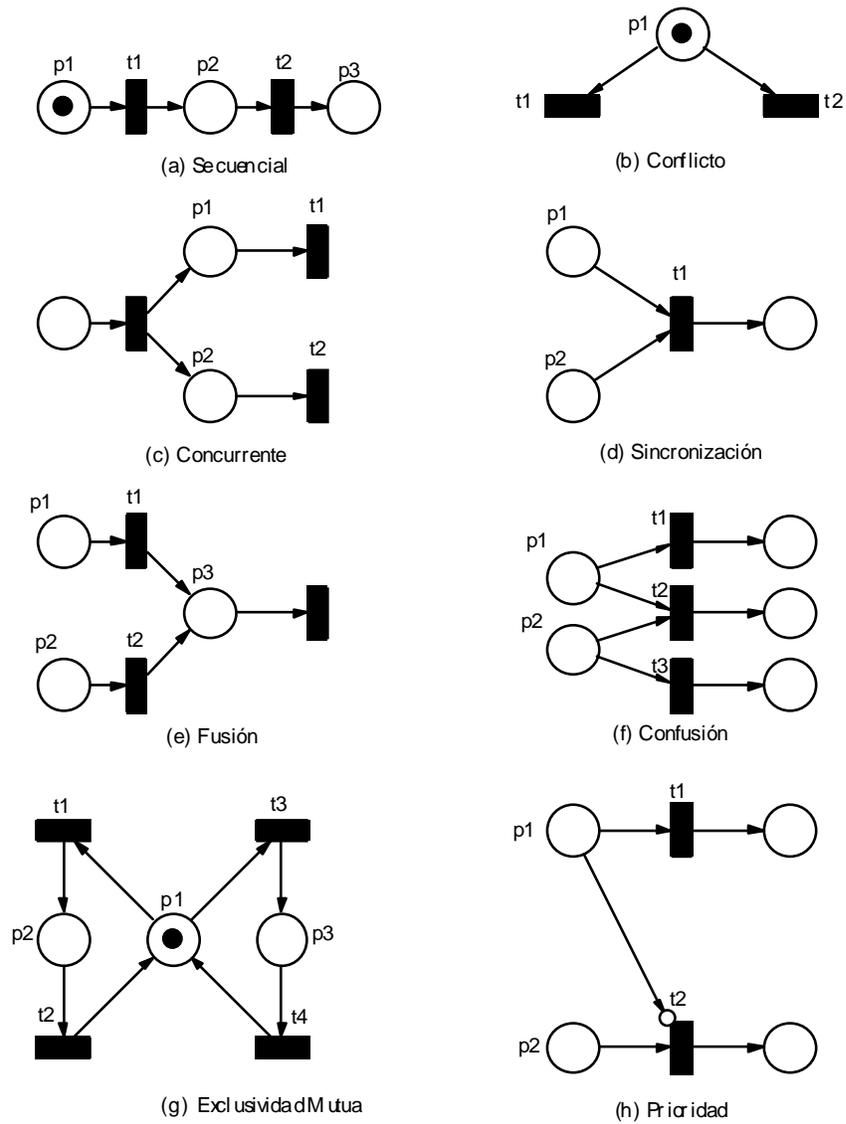


Figura 3.3: Primitivas de redes de Petri para representar características de los sistemas a modelar.

sistema.

2. *Conflicto*. Las transiciones t_1 y t_2 están en conflicto en la figura 3.3(b). Ambas están habilitadas, pero el disparo de una de ellas conduce a la deshabilitación de la otra. Esta situación se presenta, por ejemplo, cuando una máquina tiene que escoger entre varios tipos de partes, ó cuando una parte tiene que escoger entre máquinas diferentes. Estos conflictos pueden ser resueltos de una manera no determinista, o con probabilidades, asignando probabilidades apropiadas a las transiciones en conflicto.
3. *Concurrencia*. En la figura 3.3(c), las transiciones t_1 y t_2 son concurrentes. La concurrencia es un atributo muy importante en la interacción de los DEDS. Para que una transición sea concurrente es necesario la existencia de transiciones con bifurcaciones, las cuales colocan tokens en dos o más lugares de salida.
4. *Sincronización*. A menudo, las piezas de un DEDS esperan por algún recurso, y los recursos esperan por la llegada piezas o mensajes apropiados (como en una línea de montaje o un sistema que combina información). La sincronización producida en estas actividades puede ser capturada por el tipo de transición mostrada en la figura 3.3(d). Aquí, t_1 es habilitada solamente cuando p_1 y p_2 reciben un token. La llegada de un token en cada uno de los lugares p_1 y p_2 puede ser el resultado de una secuencia de operaciones en el resto de la PN. Básicamente, la transición t_1 modela la operación de unión.
5. *Fusión*. Cuando las piezas de diferentes flujos llegan por un servicio a la misma máquina, la situación resultante puede ser representada como la que se muestra en la figura 3.3(e). Otro ejemplo es la llegada de varias piezas desde diferentes orígenes a un almacén central.
6. *Confusión*. La confusión es una situación donde coexisten la concurrencia y las situaciones de conflicto. Un ejemplo es representado en la figura 3.3(f). t_1 y t_3 son concurrentes, mientras que t_1 y t_2 están en conflicto, y t_2 y t_3 también están en conflicto.
7. *Exclusividad mutua*. Dos procesos son mutuamente exclusivos si no se pueden llevar a cabo al mismo tiempo debido a las restricciones estipuladas para los recursos compartidos. En la figura 3.3(g) se muestra esta estructura. Por ejemplo, un robot puede ser compartido por dos máquinas para cargar y para descargar.

8. *Prioridades.* Las PN's no tienen mecanismos para representar prioridades, pero la implementación de prioridad en una PN se puede lograr con el uso de un *arco inhibidor*. El arco inhibidor conecta un lugar de entrada a una transición, y gráficamente es representado por un arco que termina con un pequeño círculo. La presencia de un arco inhibidor conectando a un lugar de entrada con una transición cambia completamente las condiciones para habilitar a la transición. En la presencia de un arco inhibidor, una transición es considerada como habilitada si cada lugar de entrada, conectado a la transición con un arco normal (un arco que termina con una flecha), contiene al menos el mismo número de tokens que el especificado en el peso del arco, y no existen tokens en cada lugar de entrada conectado a la transición con un arco inhibidor. La regla de disparo de transiciones es la misma como normalmente se aplica a los lugares conectados. Sin embargo, el disparo no cambia la marca en los lugares de entrada conectados por medio del arco inhibidor. Una PN con un arco inhibidor se muestra en la figura 3.3(h). t_1 es habilitada si p_1 contiene un token, mientras t_2 es habilitada si p_2 contiene un token y p_1 no tiene tokens. Esto le da una mayor prioridad a t_1 sobre t_2 .

3.5. Propiedades de PN

Como una herramienta matemática, las PN presentan algunas propiedades. Estas, cuando son interpretadas en el contexto del sistema modelado, permiten al diseñador del sistema identificar la presencia o ausencia de características funcionales específicas en el dominio de aplicación del sistema que se está diseñando. Dos tipos de propiedades de PN pueden resaltarse: de comportamiento, las cuales dependen del estado inicial o de la marca inicial de la PN; y de estructura, las cuales no dependen de la marca inicial de una PN, sino de la topología ó estructura de la PN [31]. A continuación se da una descripción de algunas de las propiedades de comportamiento más importantes: alcanzabilidad (reachability), limitada (boundedness), conservatividad (conservative) y vivacidad (liveness).

Alcanzabilidad.

Una cuestión importante en el diseño de DEDES es saber si un sistema puede alcanzar un estado específico, o manifestar un comportamiento funcional particular. En general, la pregunta es si el sistema modelado con una PN muestra todas las propiedades deseadas, como se especifica en los requerimientos del sistema, y no aquellas propiedades que no están especificadas.[31]

Para detectar si el sistema modelado puede alcanzar un estado específico, es necesario encontrar una secuencia de disparo de transiciones que transforme una marca M_0 en una marca M_i , donde M_i representa el estado al que se quiere llegar y la secuencia de disparos representa el comportamiento funcional solicitado. Una marca M_i se dice que es *alcanzable* desde una marca M_0 si existe una secuencia de disparo de transiciones que transforma una marca M_0 en una marca M_i . Una marca M_1 se dice que es *inmediatamente alcanzable* desde M_0 si el disparo de una transición habilitada en M_0 da como resultado la marca M_1 . [29]

El conjunto de todas las posibles marcas alcanzables desde un estado inicial dado se le conoce como *conjunto alcanzable* denotado por $R(M_0)$. El conjunto de todas las secuencias de disparo posibles desde M_0 es denotado por $L(M_0)$. Dado $\sigma \in L(M_0)$, el caso en que la red alcanza la marca M_i desde la marca inicial M_0 al disparar σ es denotado por $M_0[\sigma > M_i]$. Formalmente tenemos:

Definición 3.2 Para una $PN = \{P, T, F, W, M_0\}$ dada, si hay un σ que pertenece a $L(M_0)$ tal que $M_0[\sigma > M_i]$, entonces se dice que M_i es alcanzable desde M_0 .

Limitada y segura.

En una PN, a menudo los lugares son usados para representar áreas de almacenamiento de información en comunicación y sistemas de cómputo, áreas de almacenamiento de productos y herramientas en sistemas de manufactura, etc. Es de gran importancia, tener la capacidad para determinar si las estrategias de control propuestas previenen de desbordamientos a estas áreas de almacenamiento [31]. La propiedad de PN que ayuda a identificar la existencia de desbordamientos en los sistemas modelados es el concepto de *limitación* o *acotamiento* (*boundedness*).

Definición 3.3 Se dice que un lugar p es k -limitado si el número de tokens en p siempre es menor o igual a k (k es un número entero no negativo) para cada marca M_i alcanzables desde la marca inicial M_0 , es decir, $M_i \in R(M_0)$. Es seguro si es 1-limitado.

Definición 3.4 Una $PN = \{P, T, F, W, M_0\}$ es k -limitada (segura) si cada lugar en P es k -limitado (seguro).

Ejemplo 3.2 La PN de la figura 3.1 es 2-limitada, y la PN de la figura ?? es 1-limitada, lo que significa que ambas PN son seguras.

Conservatividad.

Los tokens en una PN pueden representar recursos. La cantidad de recursos que aparecen en un sistema real generalmente es un número fijo, por lo tanto, el número de tokens en un modelo de PN para este sistema debe permanecer igual sin consideración a la marca que alcanza la red. Cuando las PN son usadas para representar sistemas de asignación de recursos, esta propiedad se torna importante. [31]

Definición 3.5 Una $PN = \{P, T, F, W, M_0\}$ es estrictamente conservativa si para todas $M \in R(M_0)$,

$$\sum_{p_i \in P} M(p_i) = C, \text{ donde } C = \text{constante}$$

Esta propiedad es muy estricta, la cual indica que hay exactamente el mismo número de tokens en cada marca alcanzable de una PN. Desde el punto de vista estructural de la red, esto sólo puede pasar cuando el número de arcos de entrada para cada transición es igual al número de arcos de salida. Sin embargo, en sistemas reales los recursos frecuentemente están unidos a ciertas tareas que pueden ser ejecutadas. En ese caso, los recursos son separados después de que la tarea es terminada. Para cubrir este problema, los pesos pueden estar asociados con los lugares, permitiendo que la suma de tokens en una red sea siempre un número constante. Esto da como resultado una definición más amplia de esta propiedad:[31]

Definición 3.6 Se dice que una $PN = \{P, T, F, W, M_0\}$ es conservadora si existe un vector $w = (w_1, w_2, \dots, w_m)$ donde m es el número de lugares, y $w_i > 0$ para cada $p_i \in P$, tal que

$$\sum_{i=1}^m w_i M(p_i) = C, \text{ donde } C = \text{constante}$$

La figura 3.4 muestra el modelo de PN de otro sistema de manufactura sencillo: una máquina procesa dos tipos de piezas, la pieza tipo A y la tipo B. En este modelo de PN, p_1 representa que la máquina está disponible. p_2 y p_3 representan que las piezas de tipo A y B están disponibles, respectivamente. p_4 y p_5 representan que las piezas de tipo A y B están en procesamiento, respectivamente. Esta PN no es estrictamente conservadora, porque hay tres tokens en la marca inicial y en la marca que le precede al disparar t_1 o t_3 (la máquina comienza con el procesamiento de la pieza de tipo A o la de tipo B), hay solamente dos tokens (puesto que los recursos de la máquina y las piezas de trabajo están combinados en un solo elemento). Sin embargo, esta PN es conservadora con respecto a $w = (1 \ 1 \ 1 \ 2 \ 2)$. [29]

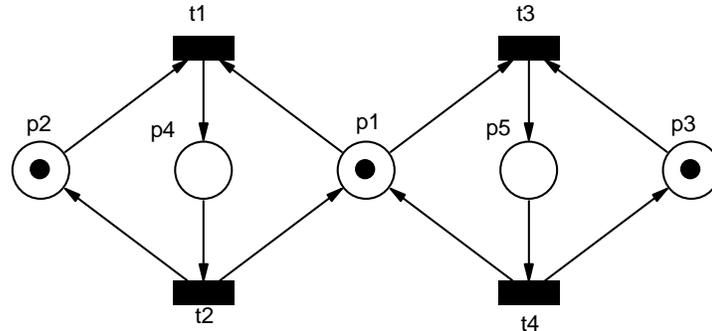


Figura 3.4: Un modelo de red de Petri para un sistema de manufactura, el cual no es conservativo.

Vivacidad.

El concepto de "vivacidad" está estrechamente relacionado con la situación de *bloqueo* (*dead-lock*), el cual ha sido situado en el ámbito de los sistemas operativos. [30]

Una PN P , con un estado inicial x_0 , se dice que está *viva* si existe alguna ruta de disparo de transiciones, tal que una transición pueda dispararse eventualmente desde algún estado alcanzado por x_0 . [36]

Por lo que se tiene la siguiente clasificación de una PN viva.

Definición 3.7 Se dice que una transición t en una red de Petri $PN = \{P, T, F, W, M_0\}$ es:

1. *L0-viva (o muerta)* si t nunca será disparada a partir de este estado.
2. *L1-viva* si hay alguna secuencia de disparo desde x_0 tal que t pueda dispararse en al menos una ocasión..
3. *L2-viva* si t puede ser disparada al menos k veces para un valor entero positivo de k .
4. *L3-viva* si existe una secuencia de disparo infinita donde aparezca t de manera infinita.
5. *L4-viva (o viva)* si t es *L1-viva* para cada estado posible alcanzado por x_0 .

Definición 3.8 Se dice que una red de Petri $PN = \{P, T, F, W, M_0\}$ es *Lk-viva*, para una marca M_0 , si cada transición en la red es *Lk-viva*, para $k = 0, 1, 2, 3, 4$.

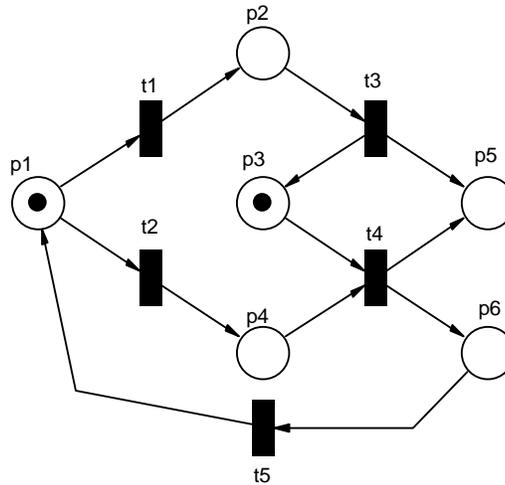


Figura 3.5: Una red de Petri no viva. Pero es estrictamente L1-viva.

Ejemplo 3.3 La PN mostrada en la figura 3.5 es estrictamente L1-viva puesto que cada transición puede ser disparada exactamente una vez en el orden de t_2 , t_4 , t_5 , t_1 y t_3 . Las transiciones t_1 , t_2 , t_3 y t_4 en la figura 3.6 son L0-viva (muertas), L1-viva, L2-viva y L3-viva, respectivamente.

3.6. Métodos de análisis de PN

Los métodos para analizar PN pueden clasificarse en los siguientes grupos: 1) el método de árbol de cobertura, 2) la ecuación de estado, que utiliza a la matriz de incidencia, 3) la técnica de simplificación de PN y 4) la simulación de la PN. El primer método tiene que ver con todas las marcas alcanzables de la marca inicial. Este método se podría aplicar a todas las clases de redes, pero está limitado a redes "pequeñas" debido a la complejidad en el incremento del espacio de estados. Por otro lado, el enfoque de ecuación de matrices y la técnica de simplificación de PN son muy poderosos, pero en muchos casos se pueden aplicar sólo subclases especiales de PN o en situaciones especiales. Para modelos de PN complejos, la simulación de eventos discretos es otra de las formas con que se pueden revisar las propiedades del sistema. [31]

El árbol de cobertura

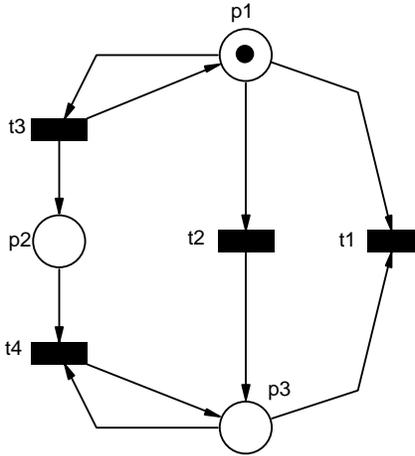


Figura 3.6: Las transiciones $t1$, $t2$, $t3$ y $t4$ son transición muerta ($L0$ -viva), $L1$ -viva, $L2$ -viva y $L3$ -viva, respectivamente.

Dada una PN, desde su marca inicial M_0 , podemos obtener igual número de marcas "nuevas" como el número de transiciones que se encuentren habilitadas. Con cada marca nueva podemos alcanzar más marcas. En el árbol obtenido, los nodos representan las marcas generadas desde M_0 y sus sucesores, y cada arco representa el disparo de una transición, el cual transforma una marca en otra.

Sin embargo, esta representación de árbol crecerá infinitamente si el número de tokens en cada lugar de la red aumenta de manera ilimitada. Para mantener un árbol finito, se incluye un símbolo especial ω , el cual representará a aquellos lugares cuyo número de tokens crezca de manera infinita. Además, ω tiene la propiedad de que para cada n , $\omega > n$, $\omega \pm n = \omega$ y $\omega \geq \omega$.

El árbol de cobertura para una PN se construye a partir del algoritmo descrito en [31]. Para PN limitada, el árbol de cobertura también se le conoce como árbol de alcanzabilidad, dado que este árbol contendrá todas las posibles marcas alcanzables.

Ecuación de estado

El comportamiento dinámico de muchos sistemas estudiados en ingeniería puede ser descrito por ecuaciones diferenciales o por ecuaciones algebraicas. Sería agradable si pudiésemos describir

y analizar completamente el comportamiento dinámico de las PN con algunas ecuaciones. Con este propósito se desarrollaron ecuaciones de estado que rigen el comportamiento dinámico de los sistemas concurrentes que son modelados con PN. Sin embargo, la factibilidad de estas ecuaciones es limitada, debido a la naturaleza no determinista inherente a las PN y a la restricción de que en las soluciones deben encontrarse sólo números no negativos. [31]

Para escribir las ecuaciones de estado, representaremos a una marca M_k como un vector columna $m \times 1$. La j -ésima entrada de M_k denota el número de tokens en el lugar j inmediatamente después del k -ésimo disparo en alguna de las secuencias de disparo. El k -ésimo disparo o *vector de control* u_k es un vector columna $n \times 1$ de $(n-1)$ 0's y una entrada diferente de cero, un 1 en la i -ésima posición indicando que la transición i dispara en el k -ésimo disparo. Puesto que el i -ésimo renglón de la matriz de incidencia A representa un cambio en la marca como resultado de disparar la transición i , podemos escribir la ecuación de estado siguiente para una PN:

$$M_k = M_{k-1} + A^T u_k, \quad k = 1, 2, \dots$$

Análisis de invariancia

Los arcos describen las relaciones entre lugares y transiciones, y pueden representarse por dos matrices $\{a_{ij}^-\}$ y $\{a_{ij}^+\}$. Al estudiar ecuaciones lineales basadas en la regla de ejecución de las PN y matrices, podemos encontrar subconjuntos de lugares en los cuales la suma de los tokens permanece sin cambio alguno. También podemos encontrar que una secuencia de disparo de transiciones lleva de regreso a una marca de inicio, es decir, después de una secuencia de disparo de transiciones regresamos a la marca con la cual iniciamos la secuencia. [31]

Reglas de simplificación de PN

Para facilitar el análisis de sistemas grandes, es común reducir el modelo del sistema a uno más sencillo, mientras se mantengan las propiedades del sistema que se está analizando. Por el contrario, las técnicas para transformar un modelo abstracto en un modelo más refinado de manera jerárquica se pueden utilizar para síntesis. Existen muchas técnicas de transformación en PN, las cuales se describen en [32].

3.6.1. Simulación de una PN

Para modelos de PN complejos, la simulación de eventos discretos es otra forma de verificar las propiedades del sistema. La idea es simple, se disparan las transiciones que se encuentren

habilitadas, actualizando el número de tokens en los lugares de entrada y salida de las transiciones disparadas. Se sigue este esquema hasta que ya no exista alguna transición habilitada o se establezca un criterio de parada. De esta manera podemos checar el comportamiento que sigue el sistema al establecer ciertas condiciones de entrada en la marca de inicio.

Las extensiones de las PN fueron desarrolladas de acuerdo a las necesidades de los investigadores, hay algunas PN que presentan ciertas variaciones contra la estructura original. Estas variaciones fueron hechas con la finalidad de cubrir las características de sistemas que no manejan solamente eventos y condiciones; además, existen sistemas que necesitan incluir variables u otra propiedad en el diseño de la PN. En la literatura se pueden encontrar tres tipos de PN: abreviaciones, extensiones y estructuras particulares.

En una PN ordinaria el peso de todos los arcos siempre es 1, solamente hay un tipo de token y la capacidad de un lugar para albergar tokens es infinita. En estas redes, una transición puede dispararse si cada lugar que la precede contiene al menos un token y el tiempo no se considera. [33]

Las extensiones de PN están conformadas por modelos en los cuales se han agregado reglas de funcionalidad, para mejorar el modelo original. Dentro de las extensiones podemos considerar tres subclases importantes; la primera subclase corresponde a modelos que tienen el poder de descripción de máquinas de Turing: PN con arco inhibidor y PN con prioridad. La segunda subclase corresponde a las extensiones que pueden modelar PN continuas y PN híbridas. La tercer subclase corresponde a las PN no-autónomas, las cuales describen el funcionamiento de aquellos sistemas cuya evolución es considerada por eventos externos y/o por tiempo: PN sincronizadas, PN con tiempo, PN interpretadas y PN estocásticas.

Entre las extensiones existentes de PN's, algunas de las propiedades que ofrecen las redes de Petri coloreadas son utilizadas en nuestro modelo. A continuación se describe a ésta extensión de PN.

3.7. Red de Petri Coloreada

La red de Petri Coloreada (CP-net o CPN) es un lenguaje orientado a gráficas para el diseño, especificación, simulación y verificación de sistemas. Son adecuadas para modelar sistemas donde la comunicación, sincronización y los recursos compartidos son importantes. Las áreas de aplicación que tienen las CPN son en protocolos de comunicación, sistemas distribuidos, sistemas embebidos,

sistemas de producción automatizada, análisis de workflow y chips VLSI. [68]

La CPN se le denomina “coloreada” porque permite el uso de tokens que contienen valores, y cada token puede ser diferenciado de los demás - en contraste con los tokens de las PN originales, ya que por convención son puntos negros ó puntos “sin color”. En un inicio, solamente conjuntos de colores pequeños y sin estructura fueron utilizados. Posteriormente, se generalizó esta teoría, incluyendo la definición de tipos de datos complejos como parte de los tokens. No hay una diferencia muy pronunciada entre un conjunto de colores y un tipo de dato, como tampoco existe diferencia entre el color de un token y el valor de un token. Para que una transición sea disparada, debe de tener tokens suficientes en sus lugares de entrada y estos tokens deben contener valores que coincidan con las expresiones de los arcos. Las elipses y círculos representan a los lugares de la CPN, los cuales describen a los estados del sistema. Las transiciones son dibujadas como rectángulos, los cuales representan las acciones que se llevan a cabo. Las expresiones de arco describen que es lo que ocurre cuando una transición es disparada. [68]

Un pequeño ejemplo de CPN se muestra en la figura 3.7, donde se describe un protocolo de transporte sencillo, el cual transfiere un número de paquetes de información a través de una red poco confiable desde un origen hacia un destino. Cada lugar contiene un conjunto de marcas (tokens). Como se mencionó antes, cada uno de estos tokens lleva un dato, el cual pertenece a un **tipo** dado. El lugar *Enviar* (en la esquina superior izquierda de la figura 3.7) tiene siete tokens en su estado inicial. Todos los valores de los tokens pertenecen al tipo *INTxDATA*, y representan siete paquetes que están listos para ser enviados. El primer elemento en cada pareja de datos es el número del paquete, y el segundo elemento es la información que se envía. Todos los 1's al frente de los apóstrofes indican que hay exactamente un token por cada uno de los paquetes definidos (en general, un lugar puede tener varios tokens con la misma información). Los lugares *SigEnvío* y *SigReg* inician con un solo token con valor 1 (que pertenece al tipo de dato *INT*). Estos lugares representan dos contadores, almacenando el número del siguiente paquete que será enviado/recibido. El lugar *Recibido* inicia con un token que contiene la cadena vacía "" (que pertenece al tipo de dato *DATA*). En este token se estarán concatenando las cadenas que se vayan recibiendo. Los lugares restantes *A-D* no tienen tokens en el estado inicial. Estos lugares representan los buffers de entrada y salida en el proceso de transmisión de la red. [68]

La ventaja principal de este tipo de PN es que, a diferencia de las otras, puede manejar datos

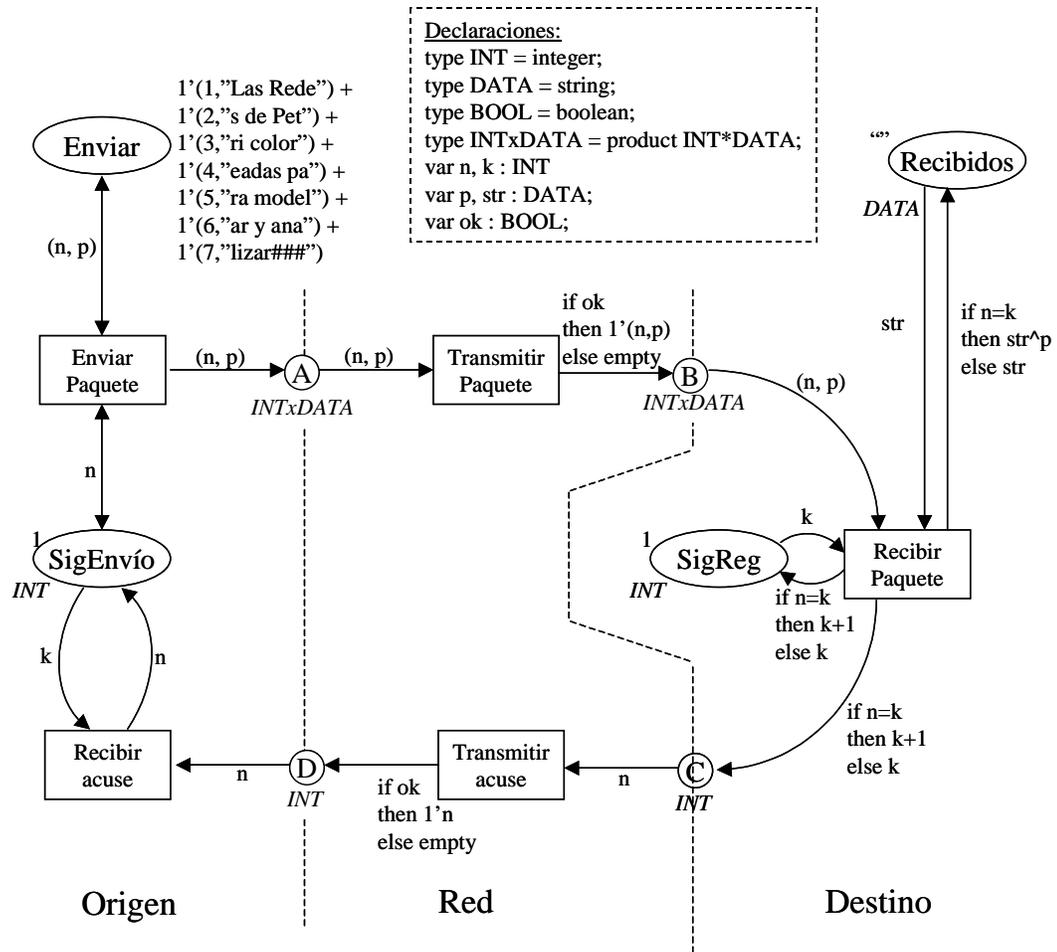


Figura 3.7: Ejemplo de una red de Petri Coloreada.

o información en los tokens. Esta característica es importante para nuestro trabajo, porque para modelar las reglas ECA necesitamos conocer los datos que provocan un cambio de estado en la base de datos.

Una de las diferencias que existe entre la CCPN y las CPN es que en la CPN no se manejan eventos, sino que se consideran a los lugares como acciones, además en la CPN una transición representa el estado que guarda el sistema que se esté modelando, mientras que en la CCPN una transición almacena datos de una regla ECA.

Además, la **expresión de arco** le indica a la CPN cual será el cambio en la CPN cuando la transición que le precede sea disparada. Las expresiones de arco para los arcos de salida de transiciones pueden ser expresiones booleanas, o bien, la estructura o el tipo del token que está permitido pasar por este arco. Y en los arcos de entrada puede especificarse únicamente el tipo de dato que puede recibir la transición a la que conecta. En el caso de la CCPN, no utiliza expresiones de arco.

3.8. Comentarios

La teoría de PN cuenta con herramientas de análisis para llevar a cabo una verificación y validación de los sistemas modelados. Los métodos de análisis que tiene la teoría de PN son el árbol de cobertura, ecuación de estado, la simplificación de PN y la simulación de PN.

Sin embargo, existen sistemas cuya modelación con PN's no es posible o generan estructuras de PN demasiado grandes, haciendo difícil su manipulación. Por tal motivo se han creado extensiones de PN con características heredadas del modelo de PN original mas las características, agregadas por los propios autores , necesarias para la modelación de los diferentes sistemas para los que fueron diseñadas. Entre las extensiones de PN's podemos encontrar PN's difusas, PN's coloreada, PN's híbridas, PN's estocásticas, entre muchas otras.

La regla de habilitación y disparo de las PN's es similar a la de activación y disparo de las reglas ECA. Para estar en condiciones de modelar un conjunto de reglas ECA por medio de un modelo de PN, es necesario agregar elementos a la PN que permitan la definición de reglas ECA para un sistema activo de base de datos.

Las herramientas de análisis que ofrece la teoría de PN pueden ser utilizadas también para el análisis de las reglas ECA modeladas como una PN extendida.

En general, la teoría de PN ofrece más propiedades para el manejo de reglas ECA que otros enfoques como la teoría de autómatas, el poder de representación de las PN's es superior al de los autómatas, además, que con los autómatas se restringe el número de estados que pueden ser considerados en la modelación de un sistema, en cambio, con PN's es posible modelar sistemas donde la cantidad de estados tienda al infinito.

Por otro lado, y de manera muy particular con bases de datos activas, la modelación de los eventos compuestos sería una tarea complicada si utilizamos autómatas, ya que por definición un autómata describe una “secuencia” de eventos, permitiendo precisamente solamente la definición del evento compuesto *secuencia*, pero para el resto de los eventos compuestos, mencionados en el capítulo 2, su modelación con autómatas no es trivial.

Capítulo 4

Red de Petri Coloreada Condicional

Existen muchas propuestas para soportar comportamientos y mecanismos reactivos dentro de un SBD para generar lo que se conoce como SBDA. Sin embargo estas propuestas están diseñadas para un sistema en particular, sin permitir su migración a otros ambientes, además de que no existe una propuesta de SBDA formalizada.

En esta tesis doctoral se está proponiendo un modelo general para el desarrollo de reglas ECA, basado en teoría de PN, el cual puede utilizarse como un motor independiente sobre cualquier SBD.

Como se mencionó en el capítulo 2, un SBDA debe ofrecer un *modelo de conocimiento* y un *modelo de ejecución*. En el *modelo de conocimiento* se especifican cada uno de los elementos que conforman a la regla ECA, es decir, el **evento**, la **condición** y la **acción**. En la conversión de una regla ECA a una CCPN, sus elementos son convertidos en elementos de CCPN.

El **evento** activador de la regla ECA se convierte en una estructura de CCPN capaz de realizar la detección y conformación del evento. Si se trata de un evento de tipo *primitivo*, éste es representado por un lugar de la CCPN. Sin embargo, si el evento de la regla es un evento compuesto, se genera la estructura de CCPN apropiada para definir al evento compuesto. Ambos tipos de eventos, primitivos y compuestos proporcionan un lugar, el cual será utilizado como lugar de entrada para una transición.

El siguiente elemento de la regla ECA, la **condición**, es almacenada dentro de una transición de la CCPN, la cual, además de evaluar la presencia de tokens en su lugar de entrada (la ocurrencia del evento) evalúa la condición de la regla que tiene almacenada. Incrementándole a la regla de

disparo de transiciones de PN's, la evaluación de una expresión booleana.

Finalmente, el elemento de la regla **acción**, debido a que al ser ejecutado modifica el estado de la BD, se representa como un lugar de salida de la transición que tiene almacenada a la condición correspondiente.

El modelo de ejecución para la CCPN se basa en la regla de disparo de transiciones de la teoría de PN, proporcionando mecanismos para la generación de tokens con información, o color, correspondiente a los eventos que estén ocurriendo en la BD. Los tokens generados son colocados en los lugares que representen a esos eventos. De esta manera se procesan todas las reglas definidas y se detectan los eventos tanto primitivos como compuestos.

Con CPN's es posible representar a las reglas ECA, pero solo aquellas que manejan eventos primitivos, siendo incapaces de manejar la definición de eventos compuestos dentro de la base de reglas ECA.

4.1. Definición de Red de Petri Coloreada Condicional

La Red de Petri Coloreada Condicional (Conditional Colored Petri Net, CCPN) es una extensión de PN, la cual hereda atributos y la regla de disparo de transiciones de PN. Además, en la CCPN se toman conceptos que están presentes en la definición de la red de Petri coloreada (CPN), tales como la definición de tipos de datos, asignación de colores (valores) a los tokens, y la asignación de tipos de datos a los lugares. En el caso de CPN's la asignación de tipos de dato se hace hacia todos los lugares de la CPN, en el caso de la CCPN, la asignación de tipos de datos a lugares no es general, ya que en la CCPN se manejan lugares (*lugares virtuales*) con la capacidad de alojar tokens de diferentes tipos de datos.

En una regla ECA se evalúa la condición de la regla. En la CCPN se utiliza una función que realiza la evaluación de la parte condicional de la regla ECA almacenada en una transición.

Para el caso de los eventos compuestos donde se tiene que verificar un intervalo de tiempo, la CCPN ofrece una función para asignar los intervalos de tiempo a una transición, la cual verificará si determinados eventos ocurren dentro del intervalo, de manera similar a como realiza la evaluación de la condición de una regla. A este tipo de transiciones la denominamos *transición compuesta*.

Como se definió previamente, cada uno de los eventos ocurre en un punto del tiempo, por lo tanto, la CCPN proporciona un función que asigna a cada token, que representa la ocurrencia de

un evento, una estampa de tiempo, el cual especifica el momento en que éste ocurrió, para efectos de evaluación de intervalos y de eventos compuestos como *secuencia* y *simultáneo*.

Finalmente, cada vez que ocurre un evento, la CCPN contiene una función para inicializar los tokens, es decir, generar la estructura del token y la asignación de los valores correspondientes al evento detectado en la BD.

La CCPN es una extensión de PN que utiliza conceptos de la CPN, por lo que también está basada en el concepto matemático de multiconjunto [68]:

Definición 4.1 *Un multiconjunto m , sobre un conjunto no vacío S , es una función $m \in [S \rightarrow N]$ la cual puede representarse como la sumatoria $\sum_{s \in S} m(s)'s$. Donde S_{MS} es el conjunto de todos los multiconjuntos sobre S . Los enteros no negativos $\{m(s) \mid s \in S\}$ son los coeficientes del multiconjunto. $s \in m$ si y solo si su $m(s) \neq 0$.*

Para hacer más clara la definición de la CCPN, utilizaremos la siguiente notación:

- Los **elementos de un tipo**, T . El conjunto de todos los elementos en T es denominado utilizando la misma letra T .
- El **tipo de una variable**, v – representado por $Type(v)$.
- El símbolo \mathcal{B} es utilizado para denotar un tipo de dato **booleano** (el cual contendrá a los elementos $\{falso, verdadero\}$ y tendrá las operaciones estándar del álgebra booleana).
- Para denotar el i -ésimo token alojado en p_j utilizamos la notación $M(p_j)_i$.

Formalmente, podemos definir a la CCPN como:

Definición 4.2 *Una red de Petri coloreada condicional (conditional colored Petri net, CCPN) es una 11-tupla*

$$CCPN = (\Sigma, P, T, A, N, C, Con, Acción, D, \tau, I)$$

donde

- (i) Σ es un conjunto finito de tipos de datos, también llamados conjuntos de colores.

- (ii) P es un conjunto finito de lugares. Para una representación gráfica más adecuada, P está dividido en los siguientes subconjuntos,

$$P = P_{prim} \cup P_{comp} \cup P_{virtual} \cup P_{copy}$$

donde P_{prim} , P_{comp} , $P_{virtual}$ y P_{copy} representan a lugares primitivos, compuestos, virtuales y copia, respectivamente.

- (iii) T es un conjunto finito de transiciones. Existen tres tipos de transiciones,

$$T = T_{regla} \cup T_{comp} \cup T_{copy}$$

donde T_{regla} , T_{comp} y T_{copy} representan a transiciones tipo regla, compuestas y copia, respectivamente

- (iv) A es un conjunto finito de arcos, tales que

$$P \cap T = P \cap A = T \cap A = \emptyset.$$

donde A está dividido en dos subconjuntos, A_{inh} y A_{nor} , los cuales representan a los conjuntos de arcos inhibidores \hat{a} y arcos normales a , respectivamente.

- (v) $N : A \rightarrow P \times T \cup T \times P$ es una función nodo. Está definida desde A hacia $P \times T \cup T \times P$.

- (vi) $C : P \rightarrow 2^\Sigma$ es una función *color*. Está definida desde el conjunto P hacia 2^Σ .

- (vii) Con es una función para evaluar condiciones. Está definida desde una transición $t \in T_{regla} \cup T_{comp}$ hacia expresiones tales que

$$\forall t \in T_{regla} : [Type(Con(t)) = \mathcal{B}],$$

donde la función Con evalúa la condición de la regla ECA.

$$\forall t \in T_{comp} : [Type(Con(t)) = \mathcal{B}],$$

donde la función Con evalúa el intervalo de tiempo, almacenado en t , contra la estampa de tiempo de los tokens.

(viii) *Acción* es una función sobre la *acción* de la regla ECA. Está definida desde el conjunto T_{regla} hacia expresiones tales que:

$$\forall t_n \in T_{regla}, p \in t_n : [Type(Acción(t_n)) = C(p)_{MS}]$$

(ix) $D : T_{comp} \rightarrow \mathbb{R} \times \mathbb{R}$ es una función de intervalos de tiempo. Está definida desde el conjunto T_{comp} hacia intervalos de tiempo $[d1(t), d2(t)]$, donde $t \in T_{comp}$ y $d1(t), d2(t)$ son los instantes inicial y final, respectivamente, del intervalo de tiempo.

(x) $\tau : M(p) \rightarrow \mathbb{R}^+$ es una función de estampas de tiempo. Está definida desde la marca $M(p)$ de un lugar p , hacia $\{0\} \cup \mathbb{R}^+$, la cual asigna a cada token del lugar p una estampa de tiempo correspondiente a un instante del reloj en la forma *año : mes : día – hora : minuto : segundo*, por ejemplo, un token puede tener una estampa de tiempo como 2005 : 10 : 06 – 16 : 30 : 00.

(xi) $I : P \rightarrow C(p)_{MS}$ es una función de inicialización de valores en la CCPN. Está definida a partir del conjunto P hacia el multiconjunto de colores $C(p)_{MS}$.

4.2. Elementos de la CCPN

El conjunto de tipos de datos Σ , o conjuntos de colores, determinan los tipos de datos que serán utilizados dentro de la CCPN para efectos de manipular la información concerniente a eventos de la BD, evaluación de expresiones booleanas en la parte condicional y los datos necesarios para llevar a cabo la acción de las reglas ECA.

El conjunto de lugares P está dividido en los conjuntos P_{prim} , P_{comp} , $P_{virtual}$ y P_{copy} . P_{prim} es un conjunto finito de lugares, los cuales representan a los eventos primitivos de las reglas ECA, gráficamente son representados mediante un círculo de una sola línea. Figura 4.1 (a). Este tipo de lugares pueden ser lugares de entrada para cualquier tipo de transición $p \in P_{prim}, p \in \bullet t, t \in T$, pero solo pueden ser lugares de salida de las transiciones T_{regla} , debido a que representan operaciones primitivas y éstas son utilizadas para describir la acción de la regla ECA, $p \in P_{prim}, p \in t \bullet, t \in T_{regla}$.

P_{comp} es un conjunto finito de lugares, mediante los cuales se representan a los eventos compuestos de las reglas ECA, donde se manejan estampas de tiempo. Estos lugares son representados gráficamente con un círculo de doble línea. Figura 4.1 (b). Este tipo de lugares son lugares de

salida para las transiciones de tipo T_{comp} que se utilizan para representar a eventos compuestos, es decir, $\forall p \in P_{comp}[p \in t^\bullet, t \in T_{comp}]$. Pero son lugares de entrada para cualquier tipo de transición, $\forall p \in P_{comp}[p \in \bullet t, t \in T]$. Es decir, un lugar $p \in P_{comp}$ representa a un evento compuesto que dispara a una regla ECA ($p \in \bullet t, t \in T_{regla}$); o a un evento compuesto que se utiliza para disparar a dos o más reglas ECA ($p \in \bullet t, t \in T_{copy}$); o bien, a un evento compuesto que forma parte de otro evento compuesto ($p \in \bullet t, t \in T_{comp}$).

$P_{virtual}$ es un conjunto de lugares virtuales. Éstos lugares solamente se utilizan para almacenar tokens en la formación de los eventos compuestos *conjunción* y *disyunción*. Gráficamente se representan por un círculo punteado. Figura 4.1 (c). Los lugares virtuales son lugares de salida de transiciones T_{comp} para el caso del evento compuesto *conjunción* o bien, son lugares de salida de transiciones tipo T_{copy} para el caso del evento compuesto *disyunción*. Por otro lado, éste tipo de lugares puede ser lugar de entrada para cualquier transición, $\forall p \in P_{comp}[p \in \bullet t, t \in T]$. Como en el caso de P_{comp} , un lugar $p \in P_{virtual}$ representa a un evento compuesto que dispara a una regla ECA, a un evento compuesto que se utiliza para disparar a dos o más reglas ECA, o a un evento compuesto que forma parte de otro .

P_{copy} es un conjunto de lugares que son utilizados para replicar eventos en el disparo de dos o más reglas. Cuando un mismo evento participa en el disparo de dos o más reglas, el procesamiento de las reglas se realiza de manera separada, por lo que es necesario generar réplicas del evento para el proceso de evaluación. Las réplicas de los eventos que se generan, son representados por lugares tipo P_{copy} . Gráficamente, los lugares tipo P_{copy} son representados mediante un círculo dibujado con doble línea, cuya línea interior es una línea punteada, tal y como se muestra en la figura 4.1 (d). Los lugares tipo P_{copy} solamente son lugares de salida para transiciones de tipo T_{copy} , $\forall p \in P_{copy}[p \in t^\bullet, t \in T_{copy}]$; y son lugares de entrada para transiciones de tipo T_{regla} y T_{comp} , $\forall p \in P_{copy}[p \in \bullet t, t \in T_{regla} \mid t \in T_{comp}]$.

El conjunto de transiciones T está dividido en los conjuntos T_{regla} , T_{comp} y T_{copy} .

Una transición $t \in T_{regla}$ representa a una regla ECA, en la cual se almacena la parte condicional de la regla y está conectada con un lugar de entrada $p_1 \in P$, donde p_1 es el evento activador de la regla, el cual puede ser un lugar de tipo P_{prim} , P_{comp} , $P_{virtual}$ o P_{copy} ; y con uno o más lugares de salida $p_2 \in P_{prim}$, dependiendo del número de acciones de la regla ECA. Gráficamente, una transición $t \in T_{regla}$ es representada mediante un rectángulo. Figura 4.2 (a).

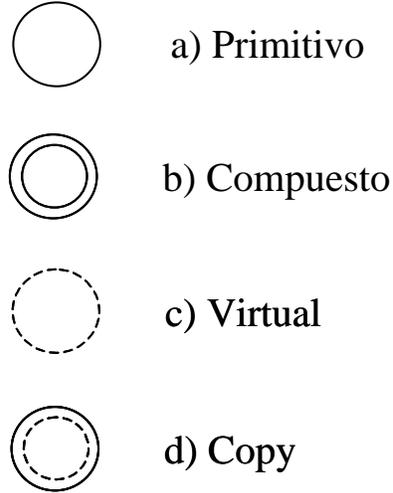


Figura 4.1: Clasificación de los lugares en la CCPN.

Una transición $t \in T_{comp}$ es utilizada para formar las estructuras de los eventos compuestos, teniendo como lugares de entrada el o los eventos que conforman al evento compuesto. Todos los tipos de lugares son válidos como lugares de entrada para este tipo de transición, es decir, $\bullet t = \{p \mid p \in P\}$; y los lugares de salida para t son lugares virtuales o compuestos, $t^\bullet = \{p \mid p \in P_{virtual} \cup P_{comp}\}$. Las transiciones $t \in T_{comp}$ se representan gráficamente mediante una doble barra. Figura 4.2 (b).

Una transición $t \in T_{copy}$ es utilizada para replicar el token de su lugar de entrada y se aplica en dos casos, *i*) cuando el lugar de entrada $p \in \bullet t$ representa a un evento, sea primitivo o compuesto, y es utilizado en dos o más ocasiones, ya sea como evento activador de una regla o para conformar eventos compuestos, teniendo como lugar de salida a un lugar $p_2 \in P_{copy}$, $t^\bullet = \{p_2\}$; y *ii*) para formar a los eventos compuestos *disyunción* y *alguno*, teniendo como lugar de salida a un lugar $p_i \in P_{virtual}$, es decir, $t^\bullet = \{p \mid p \in P_{virtual}\}$. Los lugares de entrada válidos para t son $\bullet t = \{p \mid p \in P_{prim} \cup P_{comp} \cup P_{virtual}\}$. Las transiciones $t \in T_{copy}$ se representan gráficamente mediante una barra. Figura 4.2 (c).

El conjunto de arcos A de la CCPN está dividido en los conjuntos A_{inh} y A_{nor} . El conjunto de arcos $a \in A_{nor}$ está formado por los arcos normales de teoría de PN y se utilizan para conectar

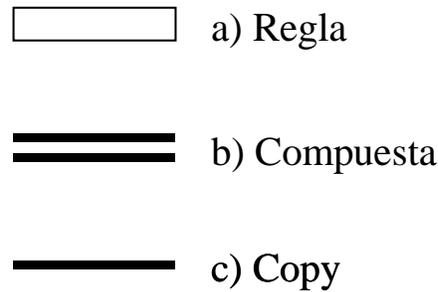


Figura 4.2: Clasificación de las transiciones en la CCPN.

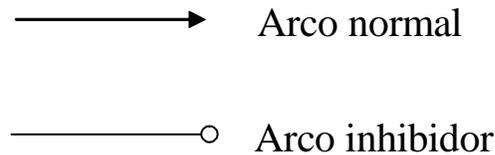


Figura 4.3: Clasificación de los arcos en la CCPN.

lugares y transiciones donde no se especifique el evento compuesto *negación*. Los arcos A_{nor} se representan gráficamente con una flecha indicando los nodos que conecta. Figura 4.3 (a).

El conjunto de arcos $a \in A_{inh}$ está formado por arcos inhibidores y se utiliza para formar la estructura del evento compuesto *negación*, el cual conecta a un lugar de entrada $p \in P$, de cualquier tipo, hacia una transición $t \in T_{comp}$. En teoría de PN, un arco inhibidor tiene la función de habilitar una transición cuyos lugares de entrada no cumplan con la regla de disparo de transiciones y viceversa. Un evento compuesto de negación es, precisamente, la no ocurrencia de un evento en determinado intervalo, de tal manera que el arco inhibidor es útil para este fin, ya que un evento lo estamos representando con un lugar de entrada para una transición. La representación gráfica de este tipo de arcos es una línea que conecta a un lugar de entrada con una transición $t \in T_{comp}$, la cual tiene un círculo pequeño en el extremo que conecta a t . Figura 4.3 (b).

La función nodo N mapea cada arco de la CCPN en pares ordenados de números enteros, los cuales indican los índices de las transiciones y los lugares que están conectadas mediante un arco

$a \in A$.

La función *color* mapea cada lugar $p \in P$ con un tipo de color $C(p)$, es decir, los datos almacenados dentro de un lugar, pertenecen a un tipo de color.

Cuando una función de condición *Con* evalúa la condición de la regla ECA, mapea cada transición $t \in T_{rule}$ a una expresión booleana donde todas las variables tienen un tipo de dato que pertenece a Σ . Si la evaluación de *Con* resulta verdadera, entonces significa que la regla se dispara, produciendo el disparo de t . Por otro lado, la función *Con* verifica si la estampa de tiempo de los tokens provenientes de los lugares de entrada, hacia una transición $t \in T_{comp}$, cumplen con el intervalo de tiempo de t .

La función *Acción* mapea cada transición $t \in T_{rule}$, hacia un conjunto de valores de algún tipo de datos $C(p)$, los cuales serán depositados en su correspondiente lugar de salida.

El intervalo de tiempo D es utilizado por la función *Con* para evaluar transiciones $t \in T_{comp}$.

Los tokens alojados en un lugar p pueden tener diferentes estampas de tiempo, así, utilizamos la función τ para cada token $M(p_i)$.

La función de inicialización *I* mapea cada lugar, p , hacia expresiones que deben de ser de tipo $C(p)_{MS}$.

Una parte importante dentro de una regla ECA es el evento que se desea detectar. La detección de eventos primitivos es una tarea sencilla, ya que solamente se estaría monitoreando un cambio específico de la BD. Sin embargo, el proceso de detección de eventos se complica a medida de que se desee incrementar la detección de la combinación de eventos primitivos, y en su caso de eventos formados a partir de la combinación de otros. La combinación de eventos generan un tipo de eventos denominados eventos compuestos.

Sin embargo, el manejo de los eventos compuestos representa un reto en cuanto a la semántica y eficiencia que aún no han sido cubiertos completamente.

4.3. Modelación de reglas ECA con CCPN

En la definición de una base de reglas ECA, utilizando la CCPN, se toma a cada una de las reglas para convertir sus elementos en elementos de la CCPN. Dado un conjunto de reglas ECA $R = \{r_1, r_2, \dots, r_n\}$, para cada regla $r_i(e_i, c_i, a_i), i = 1, 2, \dots, n$, el evento e_i es modelado con una estructura de CCPN, en el caso de eventos primitivos se utilizan lugares $p \in P_{prim}$, y para el caso de

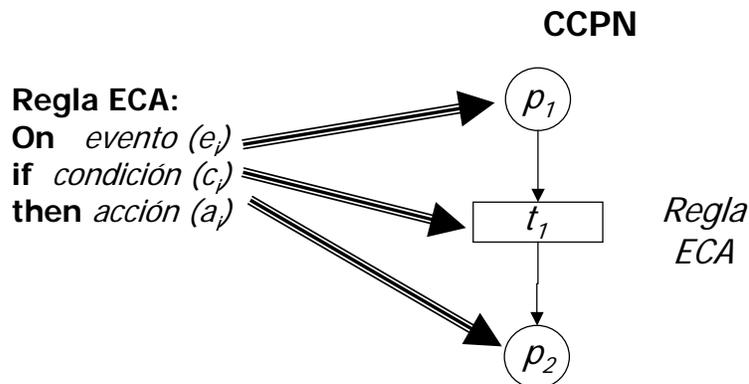


Figura 4.4: Migración de los elementos de la regla ECA (*evento-condición-acción*) a elementos de una CCPN.

eventos compuestos se generan estructuras correspondientes a cada evento compuesto, tal y como se describen en el siguiente capítulo. La condición de la regla c_i se almacena dentro de una transición $t \in T_{regla}$, la cual será evaluada durante el proceso de disparo de la transición t . La acción de la regla ECA es una operación que modifica al estado de la BD; esta operación puede ser cualquiera que se encuentra dentro de la fuente de eventos, y es representada por un lugar $p \in P_{prim}$. En la figura 4.4, se ilustra la migración del evento, la condición y la acción de una regla ECA hacia elementos de la CCPN. El lugar p_1 que representa al evento e_i de la regla puede ser de cualquier tipo ($p_1 \in P_{prim} \cup P_{comp} \cup P_{virtual} \cup P_{copy}$), por simplicidad se dibuja a un lugar P_{prim} . La condición c_i se almacena dentro de la transición $t_1 \in T_{regla}$. Finalmente, la acción de la regla ECA a_i se traduce en un lugar de salida $p_2 \in P_{prim}$ de t , tal que $t^\bullet = \{p_2\}$. Podemos decir que la regla ECA está siendo representada por t , porque en t se conocen los elementos de la regla: el evento es su lugar de entrada, la condición está almacenada en ella y la acción es su lugar de salida.

Sin embargo, existen bases de reglas donde un mismo evento (primitivo o compuesto) disparan a dos o mas reglas. Utilizando el modelo de CCPN, el evento que tiene que participar más de una ocasión en la conformación de eventos compuestos o como evento de una regla ECA, se replica para ser considerado en los lugares donde sea solicitado. En la figura 4.5 se muestra la participación de un evento primitivo p , el cual dispara a dos reglas distintas. Dado un conjunto de reglas R , dos reglas $r_1(e_1, c_1, a_1), r_2(e_2, c_2, a_3) \in R$ y $e_1 = e_2$, el evento de ambas reglas se mapea a un lugar p_1 ,

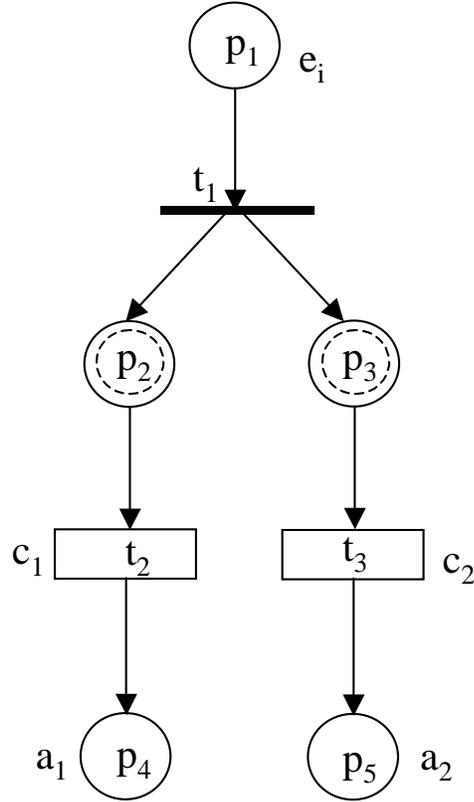


Figura 4.5: Estructura de una CCPN cuando un mismo evento e_1 dispara a dos reglas distintas.

pero como p_1 se utiliza en la evaluación de dos reglas, la información de p_1 se replica a los lugares p_2 y p_3 , utilizando una transición $t_1 \in T_{copy}$ y los lugares $p_1, p_2 \in P_{copy}$, es decir, $\bullet t_1 = \{p_1\}$ y $t_1^\bullet = \{p_2, p_3\}$. De esta manera, ambas reglas, representadas mediante t_2 y t_3 , utilizarán la información del mismo evento para evaluar sus condiciones, c_1 y c_2 , almacenadas en las transiciones $t_2, t_3 \in T_{regla}$, respectivamente, $\bullet t_2 = \{p_2\}$ y $\bullet t_3 = \{p_3\}$. Por último, la acción de ambas reglas se representan por los lugares p_4 y p_5 , $t_2^\bullet = \{p_4\}$ y $t_3^\bullet = \{p_5\}$.

En una base de reglas ECA existen relaciones entre reglas. Además de que dos reglas compartan el mismo evento, dos reglas se relacionan si el evento de una regla es a la vez la acción de la otra. Dado un conjunto de reglas ECA R , dos reglas $r_1(e_1, c_1, a_1), r_2(e_2, c_2, a_3) \in R$ y $a_1 = e_2$, entonces la estructura a que da lugar esta relación es como se muestra en la figura 4.6. La regla r_1 está

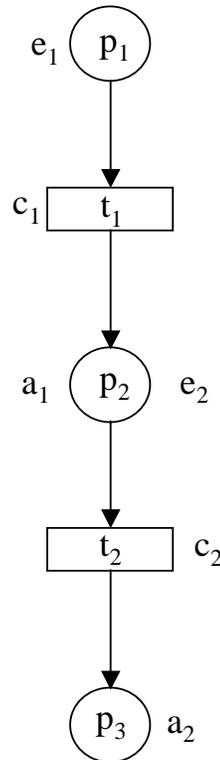


Figura 4.6: Relación existente entre dos reglas cuando la acción de una de ellas provoca el disparo de la otra.

representada por la transición t_1 y la regla r_2 está representada por la transición t_2 , $t_1, t_2 \in T_{regla}$. El evento e_1 está representado por el lugar p_1 , lugar de entrada para t_1 ($\bullet t_1 = \{p_1\}$) y la acción a_1 está representada por el lugar p_2 , lugar de salida de t_1 ($t_1^\bullet = \{p_2\}$). Dado que $a_1 = e_2$ entonces e_2 está representado también por p_2 , entonces $\bullet t_2 = \{p_2\}$ y $t_2^\bullet = \{p_3\}$. Por lo tanto $t_1^\bullet = \bullet t_2$.

4.3.1. Algoritmo de conversión de reglas ECA a CCPN

Para llevar a cabo la conversión de una base de reglas ECA a una CCPN se utiliza un algoritmo que toma la definición de la base de reglas ECA en su forma general *on - if - then* y genera los lugares y transiciones correspondientes.

Archivos eca

Los archivos ECA donde se define la base de reglas ECA contiene la definición del esquema conceptual de la BD sobre la que se definirán las reglas. En el esquema conceptual se definen las estructuras de las tablas que participan en la BD, así como las relaciones existentes entre las tablas. Además se describen los eventos que activan a las reglas ECA; finalmente se especifican las reglas en el formato *on evento, if condición, then acción*.

Los eventos primitivos se denotan utilizando los comandos básicos que se manejan en SQL para la modificación y el acceso a la información de la BD (*insert, update, delete, select*). Además, el nombre del comando se complementa con la tabla y, en su caso, el campo que afectan.

En el caso del comando *insert*, se utiliza el nombre del comando y el nombre de la tabla, el campo no es necesario porque en este caso no estamos insertando solo el campo sino todo el registro completo: *insert_tabla*.

Para nombrar a un evento primitivo que modifica a la BD con el comando *update*, si debemos especificar el nombre de la tabla y el nombre del campo, porque en este caso no necesariamente se modifica todo el registro sino que solamente puede modificarse un solo campo y para ubicar al evento debemos conocer la tabla y el campo que está siendo modificado: *update_tabla_campo*.

El comando que elimina un registro (*delete*), al igual que en el caso del comando *insert*, no necesita la especificación del campo porque se elimina el registro completo y no solamente el campo: *delete_tabla*.

El comando *select* no modifica el estado de la BD, sin embargo en ocasiones es importante conocer quién, cuando o como están accediendo a la BD, por lo tanto es necesario incluirlo como evento primitivo. En este caso el nombre para identificar a éste evento se forma, al igual que el comando *update*, con el nombre del comando, la tabla y el campo al que se está accediendo: *select_tabla_campo*.

En el caso de los eventos compuestos *conjunción, disyunción, secuencia, simultáneo, primero, último, negación, historia* y *alguno* se utilizan los comandos *and()*, *or()*, *seq()*, *sim()*, *clos()*, *last()*, *not()*, *times()* y *any()*, respectivamente. Los eventos que participan en el evento compuesto se anotan dentro de los paréntesis y se separan por dos puntos (:). Al final se especifica el intervalo de tiempo en que el evento compuesto estará escuchando la ocurrencia de eventos, además de la política de consumo que se desee aplicar al evento compuesto.

La condición de la regla se especifica como una expresión booleana donde participan valores de las tablas de la BD o valores que se encuentran dentro del mismo evento activador de la regla. Si no se quiere especificar la condición simplemente se establece un valor *true* en la parte del *if*.

La acción de la regla se define mediante instrucciones de SQL, especificando la acción o acciones que se realizarán en el dado caso de que se dispare la regla y la condición sea evaluada a verdadera.

La definición de la base de reglas ECA observando estos lineamientos se almacenan en un archivo de texto, asignándole la extensión *.eca*, para identificar los archivos que contienen definiciones de reglas ECA. Este archivo es utilizado por el algoritmo de conversión de reglas ECA a una CCPN.

Descripción del algoritmo

El diagrama de flujo del algoritmo de conversión de reglas ECA en una CCPN se muestra en la figura 4.7. Este diagrama de flujo utiliza dos módulos que más adelante se describen.

En este diagrama de flujo comenzamos con la definición de las estructuras que utilizaremos para almacenar los datos de las reglas ECA. En primer lugar tenemos el conjunto *BD* para almacenar el esquema conceptual de la base de datos, es decir, las tablas definidas en la base de datos (o relaciones si utilizamos los conceptos del modelo relacional de BD), sobre la que se aplicarán las reglas, son los elementos de éste conjunto. Además, se tienen los conjuntos de lugares (*P*), transiciones (*T*) y arcos (*A*).

Se recibe como entrada un archivo de texto (*Reglas.eca*) que contiene la definición del esquema conceptual de la BD y las reglas ECA en la forma *on – if – then*. Se llama al módulo “**Obtención esquema conceptual BD**”, el cual obtiene la definición de las tablas descritas en el archivo de texto leído. Posteriormente se llama el módulo “**Creación de CCPN para eventos**”, en donde se generan las estructuras correspondientes a los eventos a detectar por las reglas ECA. Este módulo se describe en el capítulo concerniente a eventos compuestos; en el caso de eventos primitivos se representan por un lugar $p \in P_{prim}$, sin embargo los eventos compuestos se representan por medio de estructuras de CCPN más complejas; por el momento se considera que para cada evento existe un lugar, el cual puede ser de tipo primitivo o compuesto.

El algoritmo comienza a leer cada una de las reglas ECA definidas hasta que ya no encuentra y termina su ejecución. Para cada regla ECA leída se crea una transición $t \in T_{regla}$, a la cual se le asigna la condición de la regla y es agregada al conjunto *T*. Se busca en *P* el lugar p_1 que

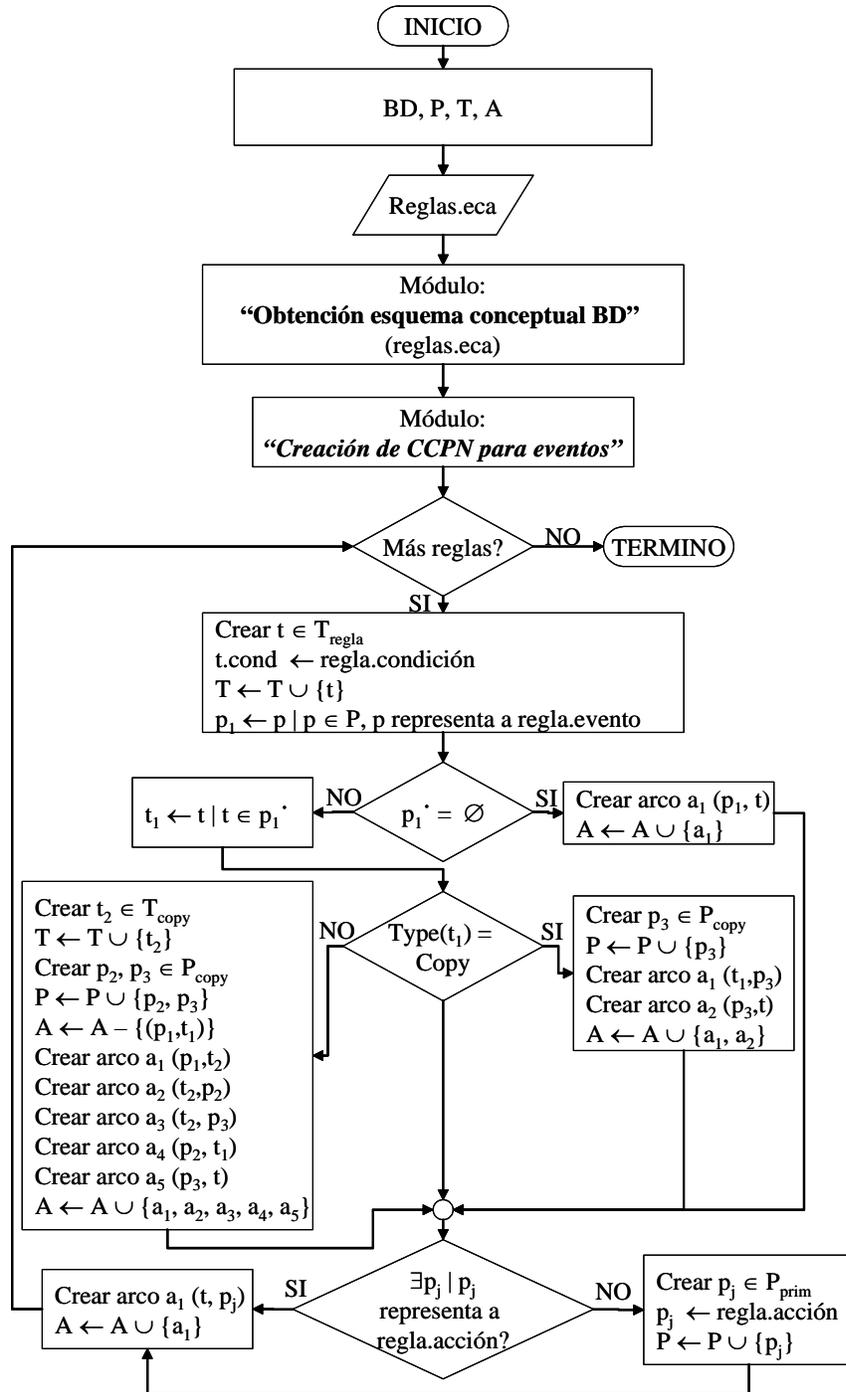


Figura 4.7: Diagrama de flujo del algoritmo utilizado para convertir una base de reglas ECA en una CCPN.

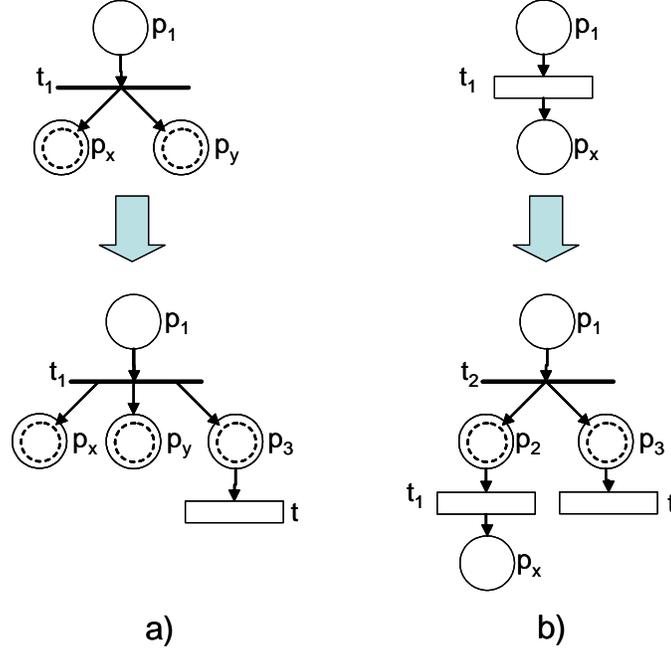


Figura 4.8: Proceso de para agregar transiciones tipo *copy*.

represente al evento de la regla. Si p_1 no es lugar de entrada de ninguna transición ($p_1^\bullet = \emptyset$), se crea un arco $a_1(p_1, t)$ y se agrega al conjunto de arcos A . En caso de que p_1 sea el lugar de entrada de una transición $t_1 \in T$ se debe verificar que tipo de transición es t_1 . Si $t_1 \in T_{copy}$ se crea un lugar $p_3 \in P_{copy}$, el cual se agrega al conjunto P ; se crean los arcos para conectar t_1 con p_3 ($a_1(t_1, p_3)$) y para conectar p_3 con t ($a_2(p_3, t)$) y se agregan al conjunto de arcos ($A \leftarrow A \cup \{a_1, a_2\}$). Este proceso se muestra en la figura 4.8 a). En caso contrario, $t_1 \notin T_{copy}$, entonces $t_1 \in T_{regla} \cup T_{comp}$, y se hace lo siguiente: Crear $t_2 \in T_{copy}$, agregar t_2 a T , crear lugares $p_2, p_3 \in P_{copy}$ y agregarlos al conjunto P , eliminar el arco que une a p_1 con t_1 , crear los arcos $a_1(p_1, t_2)$, $a_2(t_2, p_2)$, $a_3(t_2, p_3)$, $a_4(p_2, t_1)$, $a_5(p_3, t)$ y agregarlos al conjunto de arcos $A \leftarrow A \cup \{a_1, a_2, a_3, a_4, a_5\}$. Este proceso se muestra en la figura 4.8 b), donde t_1 puede ser de tipo *regla* o de tipo *compuesto*, en este caso se muestra para $t_1 \in T_{regla}$.

Después de conectar el evento de la regla (lugar) con la condición (almacenada en la transición) ahora se prosigue con la acción de la regla. Primero se verifica si ya existe un lugar $p_j \in P$ que represente a la acción de la regla. Si p_j no existe aún entonces se crea un $p_j \in P_{prim}$, se le asigna la

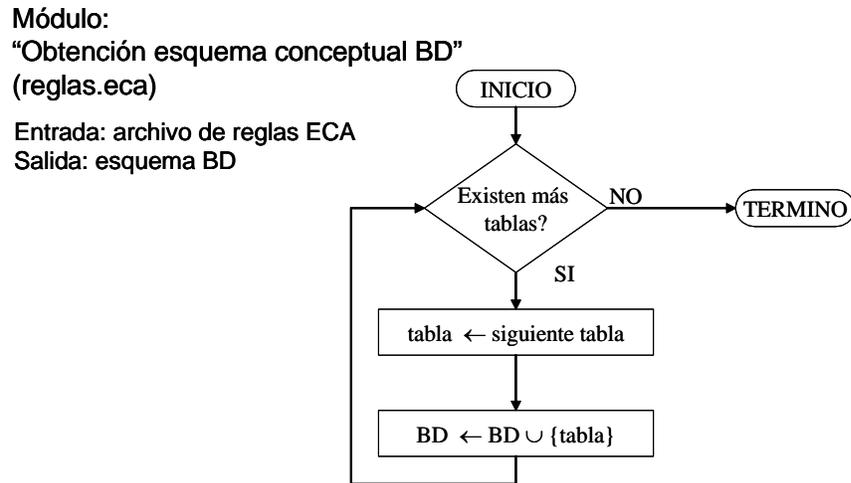


Figura 4.9: Diagrama de flujo del algoritmo para obtener el esquema conceptual de la BD descrita en el archivo reglas.ECA

acción de la regla y se agrega al conjunto de lugares. Posteriormente se crea un arco para conectar t (que tiene almacenada la condición de la regla) con el lugar p_j . Si ya existiese el lugar p_j en el conjunto P entonces simplemente se crea el arco $a_1(t, p_j)$ para conectarlos, y se agrega a_1 al conjunto de arcos.

El algoritmo para la obtención del esquema conceptual de la BD se muestra en la figura 4.9, en donde simplemente se muestra el recorrido sobre el archivo de entrada y se van recuperando las tablas definidas en él.

Aplicación

Para ejemplificar la aplicación de éste algoritmo, utilizaremos la siguiente base de reglas ECA, tomada de [50], y la descripción es la siguiente:

Regla 1: Cuando la cantidad de la prima de un empleado se modifica, si la cantidad es mayor que \$100.00, entonces el rango del empleado se incrementa en uno.

Regla 2: Cuando el rango de un empleado se modifica, si el rango ahora es mayor que el nivel 5, entonces la prima del empleado se incrementa en diez veces el nivel del rango.

Regla 3: Cuando se obtienen las ventas del mes y el número de éstas es superior a 50, entonces el rango del empleado se incrementa en un nivel.

Regla 4: Cuando el nivel del rango de un empleado se modifica y el rango alcanzó el nivel 15, entonces el salario del empleado se incrementa en un 10 %.

Las tablas que forman parte de la BD son:

EMPLEADO(emp_id, nombre, rango, salario)

PRIMA(emp_id, cantidad)

VENTAS(emp_id, mes, numero)

La traducción de ésta base de reglas ECA a un archivo *reglasTesisPhD1.eca* es:

EMPLEADO(emp_id integer, nombre CHAR(length), rango integer, salario float)

PRIMA(emp_id integer, cantidad float)

VENTAS(emp_id integer, mes integer, numero integer)

Regla 1,

ON update_PRIMA_cantidad,

IF update.cantidad > 100,

THEN update EMPLEADO set rango = empleado.rango+1 where emp_id = update.emp_id;

Regla 2,

ON update_EMPLEADO_rango,

IF update.rango > 5,

THEN update PRIMA set cantidad = prima.cantidad+rango*10 where emp_id = update.emp_id;

Regla 3,

ON insert_VENTAS,

IF insert.numero > 50,

THEN update EMPLEADO set rango = old.rango+1 where emp_id = insert.emp_id;

Regla 4,

ON update_EMPLEADO_rango,



Figura 4.10: Lugares para representar a los eventos de la base de reglas.

```
IF update.rango = 15,
THEN update EMPLEADO set salario = old.salario*1.1 where emp_id = insert.emp_id;
```

Siguiendo el procedimiento descrito en el diagrama de flujo de la figura 4.7, se lee el archivo de texto y se obtiene el esquema conceptual de la BD. Posteriormente creamos lugares para representar a los eventos, en este caso son tres eventos primitivos los que participan en esta base de reglas: `update_PRIMA_cantidad`, `update_EMPLEADO_rango` e `insert_VENTAS`, para los cuales se crean los lugares p_0 , p_1 y p_2 , respectivamente. Figura 4.10.

Ahora el algoritmo comienza a leer cada una de las reglas definidas. De acuerdo al algoritmo, para cada regla, crea una transición $t \in T_{regla}$, en este caso se crea una t_0 para la primer regla, y a la transición se le almacena la condición `update.cantidad > 100`, se busca el lugar que represente al evento de la regla 1 en P , en este caso es p_0 y como p_0 no es lugar de entrada de ninguna transición ($p_1^\bullet = \emptyset$ es verdadero) entonces se crea un arco $a_1(p_0, t_0)$ para conectarlos y se agrega el arco al conjunto de arcos A . Después, se verifica si ya existe un lugar que represente a la acción de la regla, que es la modificación del campo `rango` de la tabla `EMPLEADO` (`update_EMPLEADO_rango`), el cual ya existe (p_1) puesto que es un evento de una de las reglas. Por lo que se crea otro arco para conectar la transición t_0 con el lugar p_1 , (t_0, p_1) , y se agrega al conjunto de arcos. La CCPN obtenida hasta estos momentos se muestra en la figura 4.11.

Continuamos con la regla 2, en la cual su evento es `update_EMPLEADO_rango` correspondiente al lugar p_1 que ya existe, se crea otra transición, ahora con subíndice 1, y se le asigna la condición $t_1 \leftarrow \text{update.rango} > 5$. Como $p_1^\bullet = \emptyset$ se crea un arco para conectar p_1 con t_1 , (p_1, t_1) . Verificamos si existe un lugar que represente a la acción de la segunda regla, la cual modifica el campo `cantidad` de la tabla `PRIMA` (`update_PRIMA_cantidad`). El lugar p_0 representa precisamente a este evento, por lo que solamente creamos un arco para conectar t_1 con p_0 . La CCPN obtenida se muestra en la figura 4.12.

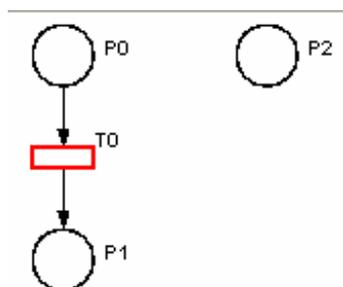


Figura 4.11: CCPN para la base de reglas considerando solamente la regla 1.

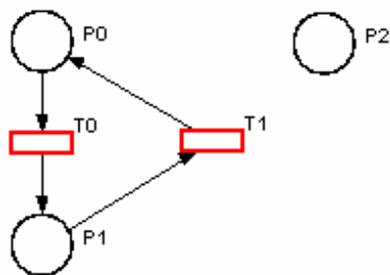


Figura 4.12: CCPN para la base de reglas considerando solamente las reglas 1 y 2.

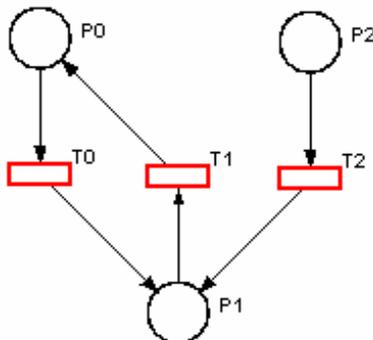


Figura 4.13: CCPN para la base de reglas considerando las reglas 1, 2 y 3.

Analizamos la tercer regla y creamos una $t_2 \in T_{regla}$ y la agregamos al conjunto de transiciones T . El evento de la tercer regla es `insert_VENTAS` denotado por el lugar p_2 , y como p_2 no es el lugar de entrada de ninguna transición, creamos un arco para conectar p_2 con t_2 . La acción de la regla está representada por el lugar p_1 utilizado para el evento `update_EMPLEADO_rango`, y como ya existe entonces creamos un arco para conectar la transición t_2 con el lugar p_1 (t_2, p_1). Figura 4.13.

La cuarta regla utiliza como evento de entrada a `update_EMPLEADO_rango` denotado por el lugar p_1 , sin embargo éste lugar ya está funcionando como lugar de entrada para t_1 , que representa a la regla 2. En esta situación es donde se utilizan los lugares y transiciones de tipo *copy*, para poder enviar la información del mismo evento a las reglas 2 y 4 representadas por las transiciones t_1 y t_3 respectivamente (figura 4.14). De acuerdo al algoritmo descrito, creamos una nueva $t_3 \in T_{regla}$, almacenamos en t_3 la condición de la cuarta regla ECA y localizamos el lugar que denota al evento de la regla 4 `update_EMPLEADO_rango`. Como se había mencionado antes, éste evento está representado por el lugar p_1 y por ende $p_1^\bullet \neq \emptyset$. La transición de salida de p_1 es t_1 y $t_1 \in T_{regla}$ (la condición $Type(t_1) = Copy$ es falsa) entonces procedemos a crear una $t_4 \in T_{copy}$ y la agregamos al conjunto T , creamos los lugares $p_3, p_4 \in P_{copy}$ y los agregamos al conjunto P , eliminamos el arco que conecta a p_1 con t_1 ($A \leftarrow A - \{(p_1, t_1)\}$), se crean los siguientes arcos $a_1(p_1, t_4), a_2(t_4, p_3), a_3(t_4, p_4), a_4(p_3, t_1), a_5(p_4, t_3)$, y se agregan al conjunto de arcos A .

Posteriormente verificamos si la acción de la regla ya se encuentra representada por medio de un lugar de la CCPN, en este caso la acción sería `update_EMPLEADO_salario`, la cual aún no se

encuentra denotada como un lugar de la CCPN, por lo cual se procede a crear un lugar para esta acción, el lugar $p_5 \in P_{prim}$ y se agrega al conjunto de lugares. Finalmente se crea un arco para conectar t_3 con el lugar p_5 .

Se preguntan si existen más reglas y como ya no se cuentan con más, entoces se termina la ejecución del algoritmo, obteniendo como resultado la CCPN mostrada en la figura 4.14.

4.4. Ejecución de la CCPN

La ejecución de la CCPN difiere un poco de la ejecución de la PN, está basada en la regla de disparo de teoría de PN, sin embargo, ésta es insuficiente para modelar el disparo de reglas ECA y la formación de eventos compuestos.

Primero se verifica si la transición $t \in T$ cumple con la regla de disparo de PN normal. Posteriormente se evalúa la transición, dependiendo el tipo de transición de que se trate. El disparo de las transiciones se maneja de diferente manera, la cual depende del tipo de transición.

Un elemento importante dentro de la ejecución de PN's son los tokens, los cuales rigen el comportamiento de la PN. En CCPN se almacenan datos dentro de los tokens, concepto tomado de la CPN. Además, para la conformación de los eventos compuestos, debemos conocer el tiempo específico en que los eventos ocurrieron, así como el lugar al que pertenecen en algún estado que guarde la BD en determinado momento. Por lo tanto, definimos a un elemento *token* de la CCPN así:

Definición 4.3 *En CCPN, un elemento token es una 3-tupla $(p, c, stamp)$ donde $p \in P$, $c \in C(p)$ y $stamp$ indica el punto en el tiempo en que ocurre el evento correspondiente y el token es depositado en p . El conjunto de todos los elementos token está denotado por TE . Una marca M es un multi-conjunto sobre TE . La marca inicial M_0 es obtenida al evaluar las expresiones de inicialización:*

$$\forall (p, c, stamp) \in TE : M_0(p, c, stamp) = I(p).$$

El conjunto de todas las marcas es expresado por $R(M)$.

Introducimos una notación nueva, $N_{color}(p)$, la cual expresa el número de diferentes tipos de colores que está presentes en un lugar p . Esta función tendrá valores mayores a 1 en los lugares

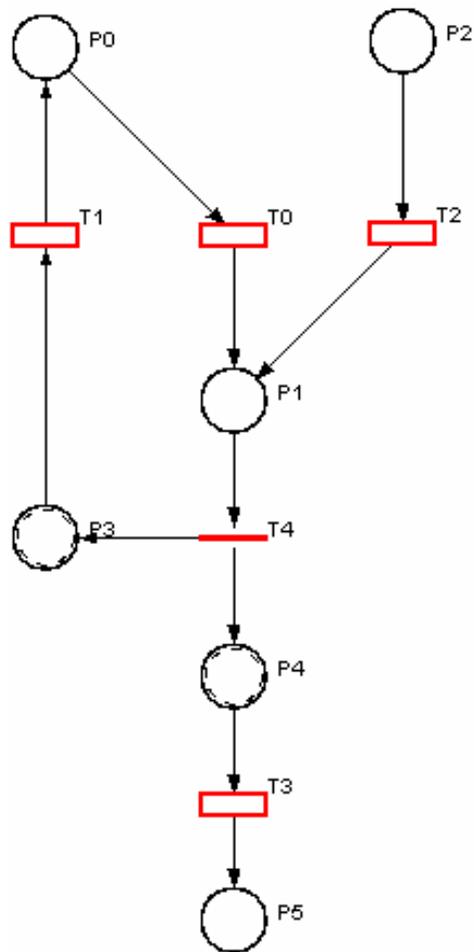


Figura 4.14: CCPN para la base de reglas del ejemplo.

$p \in P_{virtual}$, debido a los tokens que son suministrados por los lugares que le preceden. Si $p \in P_{prim} \cup P_{comp} \cup P_{copy}$ entonces los tokens alojados en p tendrán el mismo tipo de color.

Definición 4.4 Una transición $t \in T$ se habilita en una marca M ssi:

1) $\forall p \in \bullet t : |M(p)| < w(p, t)$, $t \in T_{comp}$, $type(t) = Negación$.

2) $\forall p \in \bullet t : |M(p)| \geq w(p, t)$, en cualquier otro caso.

donde $w(p, t)$ indica el peso del arco que conecta a p con t .

Definición 4.5 Cuando una transición $t \in T$ se habilita, la función de habilitación $C_{enabled}$ es definida desde $P \times T$ hacia expresiones tales que:

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

Cuando una transición t es habilitada, la función de habilitación $C_{enabled}$ es definida para especificar los tokens que provocaron la habilitación de t . En CCPN, una transición $t \in T_{copy}$ se dispara siempre y cuando esté habilitada. Pero, una transición $t \in T_{comp} \cup T_{regla}$ dispara de manera condicionada. Una transición $t_i \in T_{comp}$ dispara si está habilitada y si la condición del intervalo de tiempo se cumple. Y una transición $t_j \in T_{regla}$ dispara si está habilitada y si la evaluación de la condición de la regla ECA almacenada en t_j , contra el estado que guarde la BD, resulta verdadera.

Definición 4.6 Cuando una transición $t \in T_{comp}$ se habilita, se define una función de composición de eventos $C_{composition}$, que va de $T \times P$ hacia expresiones tales que:

$$Type(C_{composition}(t, p_0)) = Type(t)(C(p_k)_{MS})$$

donde $p_k \in \bullet t$ y $p_0 \in t^\bullet$

Dada la existencia de tres tipos de transiciones en la CCPN, existen tres casos para la ejecución de las reglas ECA.

Definición 4.7 Una transición $t \in T$ dispara ssi

(i) $\forall t \in T_{regla}$, t está habilitada y $Type(Con(t)) = verdadero$,

(ii) $\forall t \in T_{copy}$, t está habilitada, y

(iii) $\forall t \in T_{comp}$, t está habilitada y $\forall p \in \bullet t :$

$$D(t) = [d_1(t), d_2(t)] : [d_1(t) \leq \tau(M(p)) \leq d_2(t)], y$$

cumple la condición de cada evento compuesto

Definición 4.8 (*desplazamiento de tokens₁*) Cuando una transición $t \in T$ está habilitada en una marca M_1 , y es disparada, el estado de la marca M_1 cambia a una marca M_2 , definida por:

(i) Si $t \in T_{regla}$, $\forall p \in P$:

$$M_2(p) = M_1(p) - C_{enabled}(p, t) + Acción(t, p)$$

(ii) Si $t \in T_{copy}$, $\forall p_1 \in \bullet t, p_2 \in t \bullet$:

$$M_2(p_1) = M_1(p_1) - C_{enabled}(p_1, t)$$

$$M_2(p_2) = M_1(p_2) + C_{enabled}(p_1, t)$$

(iii) Si $t \in T_{comp}$, $\forall p_1 \in \bullet t, p_2 \in t \bullet$:

a) Si $Type(t) = Negación$,

$$M_2(p_1) = M_1(p_1)$$

$$M_2(p_2) = M_1(p_2) + C_{composition}(t, p_2)$$

b) En caso contrario,

$$M_2(p_1) = M_1(p_1) - C_{enabled}(p_1, t)$$

$$M_2(p_2) = M_1(p_2) + C_{composition}(t, p_2)$$

Cuando una transición se habilita, no significa que además dispare. El disparo de una transición depende de la evaluación adicional que se tenga que hacer. Si una transición t se habilita, pero no es disparada, entonces el token que la activó es eliminado de la CCPN, con el fin de desechar información no necesaria y evitar la acumulación masiva de datos. Las transiciones que pueden habilitarse y no ser disparadas son transiciones $t \in T_{regla} \cup T_{comp}$, porque la evaluación de la condición de la regla ECA almacenada en una $t \in T_{regla}$ puede resultar falsa, entonces la transición no se dispara. Por otro lado, una transición $t \in T_{comp}$ puede habilitarse ante la presencia de los eventos que conforman al evento compuesto, pero si no cumplen con el intervalo de tiempo D especificado y la condición adicional que estipula el propio evento compuesto, entonces t no se dispara. La única excepción de las transiciones $t \in T_{comp}$ es cuando $Type(t) = Negación$, porque cuando t se habilita significa que el intervalo D ha terminado y no ocurrió el evento representado

The figure shows three database window screenshots. The first window, 'VENTAS : Tabla', displays a table with columns 'emp_id', 'mes', and 'numero'. The second window, 'PRIMA : Tabla', displays a table with columns 'emp_id' and 'cantidad'. The third window, 'EMPLEADO : Tabla', displays a table with columns 'emp_id', 'nombre', 'rango', and 'salario'. Each window shows a grid of data and a status bar at the bottom indicating the current record and total records.

emp_id	mes	numero
001	12	36
002	12	25
003	12	40

emp_id	cantidad
001	\$80.00
002	\$60.00
003	\$89.00

emp_id	nombre	rango	salario
001	Juan Pérez	3	\$5,000.00
002	José Rodríguez	2	\$4,800.00
003	María Alvarado	4	\$8,500.00

Figura 4.15: Estado de la BD al inicio del procesamiento del disparo de reglas.

por p , $\{p\} = \bullet t$, entonces se procede a disparar a t . Por lo tanto, el desplazamiento de tokens en caso de que una transición $t \in T_{regla} \cup (T_{comp} - \{t \mid Type(t) = Negación\})$ no se dispare es el siguiente:

Definición 4.9 (*desplazamiento de tokens₂*) Cuando una transición $t \in T_{regla} \cup (T_{comp} - \{t \mid Type(t) = Negación\})$ está habilitada en una marca M_1 , pero NO es disparada, $Type(Con(t)) = falso$, y además, para el caso de que $t \in T_{comp}$, la estampa de tiempo del token cae fuera del intervalo D , el estado de la marca M_1 cambia a una marca M_2 , definida por:

$$\forall p \in P : M_2(p) = M_1(p) - C_{enabled}(p, t)$$

La expresión $M_1[t \succ M_2$ denota que M_2 puede alcanzarse directamente a partir de M_1 , disparando $t \in T$.

4.5. Ejemplo

Para mostrar la ejecución de una CCPN, se toma la CCPN que resultó en la base de reglas utilizada en la sección anterior (figura 4.14). Suponiendo que se tiene un estado de BD como se muestra en la figura 4.15.

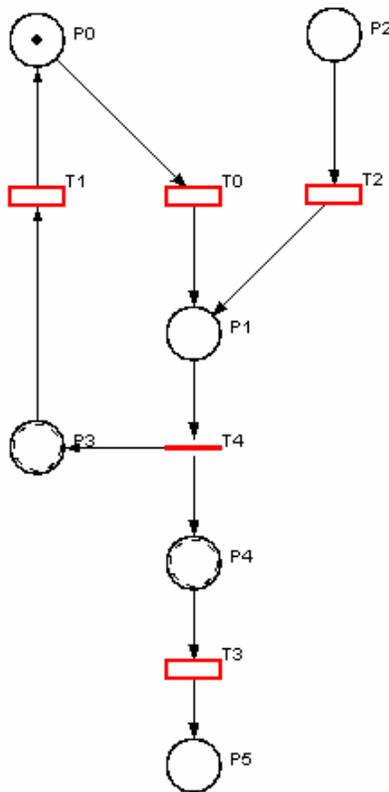


Figura 4.16: Se coloca un token en p_0 para expresar la ocurrencia del evento, con la información correspondiente al mismo.

Ocurre un cambio de estado en la BD a partir una modificación del campo *cantidad* de la tabla *PRIMA*, donde a este campo se le asigna 150 para el registro cuya clave (*emp_id*) es 001. Se genera un token con la información correspondiente al valor de la modificación y el valor de la clave. El token generado es depositado en el lugar p_0 , como se muestra en la figura 4.16.

De acuerdo a la definición 4.4 la transición t_0 es habilitada por el token alojado en p_0 , t_0 toma el token de p_0 y verifica su información para evaluar la condición que tiene almacenada. En este caso, la condición alojada en t_0 es la correspondiente a la regla 1 :

IF `update.cantidad` > 100,

donde se verifica si el nuevo valor para el campo `cantidad` es mayor que 100. De acuerdo a

la información del token, el nuevo valor del campo `cantidad` es 150, entonces la evaluación de la expresión lógica resulta verdadera, $Type(Con(t_0)) = verdadero$. Esto produce el disparo de t_0 , definición 4.7.

Cuando t_0 es disparada, se realiza el desplazamiento de tokens a través de la CCPN, eliminando el token de p_0 y enviando el token, correspondiente a la acción de la regla $C_{enabled}(p_1, t_0)$, hacia p_1 , definición 4.8. La información contenida en $C_{enabled}(p_1, t_0)$ corresponde a la acción de la regla `update EMPLEADO set rango = empleado.rango+1 where emp_id = update.emp_id`. Donde se expresa que el campo `rango` en la tabla `Empleado` se incrementa en uno para el empleado cuyo valor para `emp_id = 001`. De acuerdo al estado de la BD, el resultado para las operaciones `empleado.rango+1 = 3+1 = 4` y `update.emp_id = 001`. Por lo tanto, esta expresión puede reescribirse con los siguientes datos `update EMPLEADO set rango = 4 where emp_id = 001`.

El estado de la CCPN después del disparo de t_0 se muestra en la figura 4.17.

Cuando el token llega a p_1 , la transición $t_4 \in T_{copy}$ es habilitada, y de acuerdo a la definición 4.7 *ii*), t_4 es disparada, enviando una copia del token recibido hacia $p_3, p_4 \in P_{copy}$. Figura 4.18.

En este momento la secuencia de ejecución se realiza de manera paralela, por un lado se verifica la condición de t_1 y por el otro, la condición de t_3 . La condición de t_1 es

`IF update.rango > 5,`

sin embargo, $Type(Con(t_1)) = falso$, porque el valor del rango que tiene el token es 4, y por la definición 4.7 *i*) t_1 no es disparada.

En el otro hilo de ejecución, en la transición t_3 se tiene la condición

`IF update.rango = 15,`

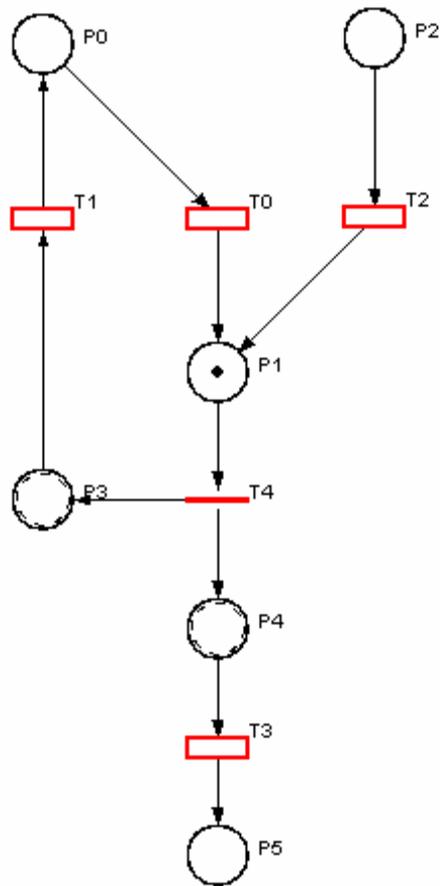
pero al igual que en t_1 , $Type(Con(t_3)) = falso$, y el valor del rango alojado en el token es 4.

De esta manera, se realiza el mismo procedimiento para cada evento que ocurre en la BD y se genera el token correspondiente, colocándolo en su lugar de la CCPN respectivo.

4.6. Comentarios

La red de Petri coloreada condicional (CCPN, Conditional Colored Petri Net) es una extensión de PN a la que se le han agregado los elementos necesarios para la modelación de reglas activas.

La CCPN es un modelo formal de representación de reglas ECA donde, a partir de la misma estructura de CCPN obtenida para un conjunto de reglas ECA, puede llevarse a cabo el análisis de

Figura 4.17: Estado de la CCPN después del disparo de t_0 .

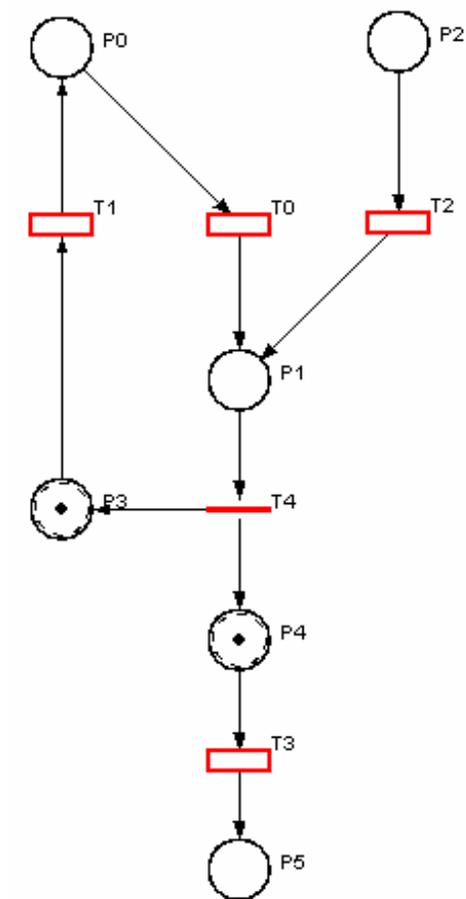


Figura 4.18: Estado de la CCPN después del disparo de t_4 .

las reglas, la simulación del comportamiento de las reglas sobre una BD y la ejecución de las reglas sobre un SBD real.

La CCPN ofrece una definición formal, donde se especifican los elementos de la regla ECA y sus elementos correspondientes en la CCPN. El evento de la regla es mapeado hacia un lugar de la CCPN (evento primitivo) ó a una estructura de CCPN (evento compuesto); la condición de la regla se almacena dentro de una transición; y la acción de la regla, debido a que modifica el estado de la BD se considera como evento, también se representa por un lugar de la CCPN.

La CCPN cuenta con un conjunto P de lugares, el cual se divide en cuatro subconjuntos: el conjunto P_{prim} , para representar eventos primitivos; P_{comp} , para representar eventos compuestos; P_{copy} , para representar eventos que son requeridos en al menos dos ocasiones como evento de regla o como elemento constituyente de un evento compuesto; y $P_{virtual}$, cuyos elementos son utilizados como almacenes de tokens.

Además cuenta con un conjunto de transiciones T , dividida en los conjuntos T_{regla} , donde se almacena la parte condicional de la regla, el evento que debe detectarse y la acción a ejecutarse; T_{comp} , cuyos elementos son utilizados para la especificación de los eventos compuestos; y T_{copy} , cuyos elementos se utilizan para replicar tokens con información sobre algún evento, hacia los lugares que pertenecen a P_{copy} .

La CCPN utiliza dos tipos de arcos; el arco normal que conecta a un lugar con una transición o viceversa, trasladando tokens entre estos elementos. Y el arco inhibidor, el cual envía tokens hacia una transición siempre y cuando el lugar de entrada contenga un número de tokens menor al peso del arco inhibidor. Este último arco es útil en la definición del evento compuesto *negación*.

La CCPN también cuenta con reglas de disparo, considerando que, además de la regla de disparo para una PN original, en las transiciones de tipo T_{regla} la evaluación de la condición debe resultar verdadera y en las transiciones de tipo T_{comp} deben evaluarse las estampas de tiempo correspondientes a la ocurrencia de eventos así como los intervalos de tiempo en que los eventos deben de ocurrir en la formación de eventos compuestos.

Capítulo 5

Modelación de Eventos Compuestos

La especificación de eventos compuestos en un SBDA lo convierte en un sistema robusto en la especificación de un sistema reactivo. Un evento compuesto se forma a partir de la combinación de eventos primitivos y/o compuestos, a través de un álgebra de eventos

En este capítulo de modelación de eventos compuestos se presentan los distintos trabajos realizados sobre el manejo y especificación de eventos compuestos en SBDA. Además, se describen los patrones utilizados en la representación de los eventos compuestos *conjunción*, *disyunción*, *secuencia*, *simultáneo*, *negación*, *primero*, *último*, *historia* y *alguno*, mediante el uso de la CCPN.

5.1. Estado del arte

Existen diferentes SBDA que soportan el manejo de eventos compuestos. Los SBDA's orientados a objetos son los que ofrecen una mayor versatilidad en el manejo de este tipo de eventos, sin embargo, los sistemas relacionales, aunque en menor capacidad, también ofrecen la detección de un número limitado de eventos compuestos.

A continuación se describen los SBDA's más relevantes en el manejo y detección de eventos compuestos.

5.1.1. SAMOS

SAMOS es un proyecto de investigación desarrollado en la Universidad de Zurich, en Suiza. SAMOS es un acrónimo de *Swiss Active Mechanism Based Object Oriented Database Systems*. También podría considerarse el nombre de la isla griega donde el filósofo y matemático Pitágoras vivió.

SAMOS agrega características de una BDA en un modelo de datos orientado a objetos, en la cual conceptos como clases, objetos y herencia son considerados para su diseño. Además del lenguaje de definición de datos que proporciona la BD sobre la que trabaje SAMOS, éste provee un lenguaje de definición de reglas para la especificación de reglas ECA. [38]

SAMOS soporta eventos *primitivos* y *compuestos*. Por el lado de los eventos primitivos, SAMOS detecta eventos generados a partir de *métodos* de las clases, los cuales pueden ser de dos tipos, los eventos que se dan antes de que el método actual sea invocado y, por otro lado, los eventos que se generan después de la invocación de un método.

Los eventos generados por una *transacción* son generados al inicio o al final de una transacción de usuario. Los eventos considerados en una transacción son los definidos como *inicio* de transacción, *final* de la transacción ó la *cancelación* (*abort*) de una transacción.

Además, los eventos primitivos de SAMOS soportan a los eventos que manejan tiempo, como eventos de tiempo *absolutos*, *relativos* y *periódicos*.

Finalmente, SAMOS maneja *eventos abstractos*, los cuales son eventos definidos por el usuario y son generados solamente mediante la intervención de un programa de aplicación utilizado por un usuario.

Por otro lado, SAMOS ofrece un lenguaje de expresiones de eventos para la definición de *eventos compuestos*. Los eventos compuestos que se definen son la *disyunción* ($E_1 \mid E_2$), el cual es alcanzado cuando ha ocurrido cualquiera de las expresiones que participan en el evento. La *conjunción* de eventos (E_1, E_2) es alcanzado cuando han ocurrido ambas expresiones que participan en la composición del evento compuesto. Y la *secuencia* de eventos ($E_1; E_2$), el cual es alcanzado si la primera expresión del evento ocurre antes que la segunda expresión.

Además, en SAMOS pueden especificarse eventos compuestos que dependen de un intervalo de tiempo. La ocurrencia del evento compuesto que utiliza el *operador* " * " es alcanzada después de la primera ocurrencia de la expresión denotada como argumento del evento, aún si existen mas de

una ocurrencia de la expresión dentro de un intervalo de tiempo definido. Por ejemplo, $(*E_1 \text{ en } I)$ expresa que este evento compuesto será alcanzado a partir de la primera ocurrencia de la expresión E_1 , dentro del intervalo de tiempo I .

El evento compuesto *historia* es alcanzado después de la ocurrencia de un n número de veces de una expresión de evento, dentro de un intervalo definido. Por ejemplo, $times(n, E_1)$ en I , es alcanzado después de que la expresión E_1 ha ocurrido n veces dentro del intervalo de tiempo I .

La *negación* de una expresión de eventos es alcanzada si la expresión que recibe como argumento no ocurre dentro de un intervalo de tiempo. En SAMOS, la especificación del intervalo de tiempo es obligatorio. Por ejemplo $not E_1$ en I es alcanzado al final del intervalo I , siempre y cuando no haya ocurrido el evento E_1 .

SAMOS utiliza los siguientes parámetros de ambiente:

$occ_tid(E)$ da el identificador de la transacción donde ocurrió E .

$occ_point(E)$ da el punto en el tiempo en el cual ocurrió el evento E .

$user_id(E)$ da el identificador del usuario que comenzó la transacción donde ocurrió E .

Para operar con estos parámetros SAMOS utiliza el comando *same* al definir el evento de una regla ECA.

Para la detección de éstos eventos compuestos, SAMOS utiliza una S-PetriNet, la cual está basada en una CPN [39]. Para cada evento compuesto definido en SAMOS se genera una estructura de S-PetriNet, y si en la composición de un evento compuesto tiene como parámetros a otros eventos compuestos, entonces la S-PetriNet que se genera se forma a partir de la combinación de todos los constructores para eventos primitivos y compuestos que participan en él. Cuando ocurre un evento primitivo que forma parte de un evento compuesto, se coloca un token en el lugar de la S-PetriNet correspondiente, para especificar su ocurrencia, y así sucesivamente hasta concluir con la formación del evento compuesto. Al ser formado el evento compuesto, se envía una señal al manejador de reglas de SAMOS para continuar con el proceso de disparo de la regla ECA a la que pertenece el evento detectado.

La diferencia que existe entre SAMOS y la propuesta de esta tesis doctoral radica en que en SAMOS solamente se utilizan PNs en la parte de detección de eventos y no en toda la modelación de reglas, la parte de evaluación de regla y de ejecución de acciones se realiza de manera separada, en un ambiente donde ya no se utilizan PNs; además de que los patrones para la detección de

eventos compuestos no manejan color en los tokens. En cambio, la CCPN modela patrones para la detección de eventos compuestos y los modelos de conocimiento y de ejecución como una PN, además de realizar el análisis de las reglas en el mismo modelo.

5.1.2. ODE

Los eventos compuestos en ODE son especificados como expresiones de eventos utilizando operadores para la especificación de eventos. En ODE se consideran atributos para los eventos primitivos que se manejan, tales como identificador de la transacción, parámetros de invocación a funciones, etc. Los atributos del evento también consideran el estado del minimundo en el momento en que éste ocurre. Los operadores básicos para formar expresiones de eventos son los siguientes:

- $a[h]$, donde a es un evento primitivo, es el subconjunto máximo de la historia h compuesto por las ocurrencias de eventos de la forma (a, eid) .
- $(E \wedge F) = h_1 \cap h_2$, donde $h_1 = E[h]$ y $h_2 = F[h]$.
- $(!E)[h] = (h - E[h])$.
- $relativo(E, F)[h]$ son las ocurrencias de eventos en h en el cual F se satisface, suponiendo que la historia comienza a formarse inmediatamente después de la ocurrencia de algún evento en h en la cual E toma lugar.

Ode cuenta además con operadores adicionales para la formación de eventos:

- $prior(E_1, E_2) = relativo(E_1, alguno) \wedge E_2$, E_2 toma lugar después de algún tiempo definido después de la ocurrencia de E_1 .
- $secuencia(E_1, E_2) = relativo(E_1, !(relativo(alguno, alguno))) \wedge E_2$, E_2 debe ocurrir después de la ocurrencia de E_1 .

5.1.3. Snoop

Dados E_1, E_2, E_3 algunos eventos primitivos o compuestos, los operadores que soporta Snoop en la composición de eventos son;

- *Disyunción*(∇), *Conjunción*(Δ), *Secuencia*($;$), *Negación*(\neg).
- *Alguno*(m, E_1, E_2, \dots, E_n), donde m eventos de un total de n distintos eventos deben ocurrir.
- *Eventos aperiódicos* ($A(E_1, E_2, E_3)$), A ha ocurrido cada vez que ocurre E_2 en el intervalo de tiempo definido por el tiempo de ocurrencia de E_1 y E_3 .
- *Eventos periódicos* ($P(E_1, TI, E_3), P * (E_1, TI[: \text{parametros}], E_3)$), P ocurre cada intervalo de tiempo TI durante el intervalo $[E_1, E_3]$, TI es una constante de tiempo. En el caso de los parámetros $P*$, éstos son coleccionados cada intervalo TI , pero son considerados al final del intervalo $[E_1, E_3]$.

5.1.4. EPL

A diferencia de los otros lenguajes de composición de eventos que están basados en un álgebra de eventos, EPL está basado en lógica.

Dados los eventos E_1, E_2, \dots, E_n , en EPL se formarían las siguientes expresiones de eventos:

- (E_1, E_2, \dots, E_n) : Una secuencia que consiste de una instancia de E_1 , seguida inmediatamente por una instancia de E_2, \dots , seguida inmediatamente por una instancia de E_n .
- $*$: E : Una secuencia de cero o mas instancias consecutivas de E .
- $(E_1 \& E_2 \& \dots \& E_n)$: Una conjunción de eventos. Este evento ocurre cuando todos los eventos ocurren simultáneamente.
- $\{E_1, E_2, \dots, E_n\}$: Una disyunción de eventos. Este evento ocurre cuando al menos uno de los eventos E_1, E_2, \dots, E_n ocurre.
- $!E$: Este evento ocurre cuando no existen instancias de ocurrencias de E .

5.1.5. CEDAR

CEDAR (Composite Event Definition for Active Rules) es un lenguaje de especificación de eventos compuestos, en el cual se definen un número de operadores para la composición de eventos compuestos. [73]

Los eventos primitivos en CEDAR son sus objetos básicos. Un evento primitivo es un par $(nombre_evento(atributos), instante_tiempo)$, donde $nombre_evento$ es un nombre simbólico del evento e $instante_tiempo$ es el instante de tiempo (en el reloj del sistema) cuando el evento ocurre.

Un evento compuesto es una secuencia de eventos primitivos en algún orden de tiempo. Los eventos compuestos se especifican utilizando *expresiones de eventos*, los cuales se forman utilizando eventos primitivos, intervalos y operadores del lenguaje

Dadas las expresiones de eventos E, E_1, E_2 , donde E, E_1, E_2 pueden ser eventos primitivos o compuestos, se tienen los siguientes operadores:

- $E = E_1 \ominus E_2$, tal que $E_1(t) = E_1(t) \text{ OR } E_2(t)$. El operador \ominus define el evento que ocurre cuando alguno de E_1 o E_2 ocurre.
- $E = E_1 \text{ fby } E_2$, tal que $E(t) = \exists t_1((E_1(t_1) \text{ AND } E_2(t)) \text{ AND } t_1 < t)$. El evento E sucede cuando E_2 ocurre después de la ocurrencia de E_1 .
- $E = E_1 \text{ conc } E_2$, tal que $E(t) = (E_1(t_1) \text{ AND } E_2(t))$. Ambos eventos E_1 y E_2 suceden concurrentemente.
- $E = E_1 \text{ in } [I]$, donde I es un intervalo de tiempo delimitado por el inicio del evento I^i y el final del evento I^f . E sucede cuando E_1 ocurre dentro del intervalo I .
- $E = E_1 \text{ not_in } [I]$, E ocurre si E_1 no sucede dentro del intervalo I .
- $E = first(E_1) \text{ in_end } [I]$, E ocurre solamente con la primera ocurrencia de E_1 dentro del intervalo I , ocurrencias posteriores de E_1 son omitidas.
- $E = last(E_1) \text{ in_end } [I]$, E ocurre solamente con la última ocurrencia de E_1 dentro del intervalo I .

CEDAR utiliza la CPN para la implementación de éstos eventos compuestos. Para cada uno de los operadores del álgebra de eventos de CEDAR se genera su estructura correspondiente en CPN, creando constructores para cada caso.

5.1.6. Sistemas comerciales

Los sistemas comerciales soportan el comportamiento activo por medio de la definición de disparadores, sin embargo, en la definición de éstos solamente se permite su definición sobre una tabla. El único evento compuesto que manejan es la *disyunción*, restringida además a operaciones sobre una sola tabla.

5.2. Eventos Compuestos en CCPN

La detección de eventos compuestos es una parte importante dentro del modelo de conocimiento de un SBDA. Es importante dar soporte a esta parte para que en el desarrollo de una base de reglas ECA se cubran los requerimientos que un desarrollador de reglas necesite. En el modelo CCPN se ofrecen nueve constructores de los eventos compuestos más utilizados, tales como la conjunción, disyunción, negación, secuencia, simultáneo, primero, último, historia y alguno; descritos todos ellos en el capítulo 2. Los constructores de los eventos compuestos corresponden a estructuras en términos de la CCPN para denotar a cada uno de los eventos que lo, ya sean éstos primitivos o compuestos.

5.2.1. Conjunción

Para que el evento compuesto *conjunción* tome lugar, deben haber ocurrido cada uno de los eventos que lo constituyen, sin importar su orden de ocurrencia. Para modelar a éste evento utilizando la teoría de CCPN, tomamos la regla de disparo de una transición $t \in T_{comp}$ que tiene al menos dos lugares de entrada, $|\bullet t| \geq 2$. La transición t debe de ser de tipo compuesta, porque este tipo de transiciones son las que se utilizan para la creación de los constructores de eventos compuestos, a excepción del evento compuesto *disyunción*.

La transición t será disparada siempre y cuando el número de tokens para cada lugar de entrada sea igual o mayor al peso de los arcos, $\forall p_i \in \bullet t [M(p_i) \geq w(p_i, t)]$. Si cada uno de los lugares de entrada p_i representa a un evento constituyente del evento compuesto de la conjunción, entonces la transición será disparada hasta que ocurran los eventos constituyentes y sean colocados los token correspondientes en cada p_i , denotando así su ocurrencia. Después del disparo de t , se envía un token al lugar de salida $p_j \in P_{virtual}$, el cual representará la ocurrencia del evento compuesto

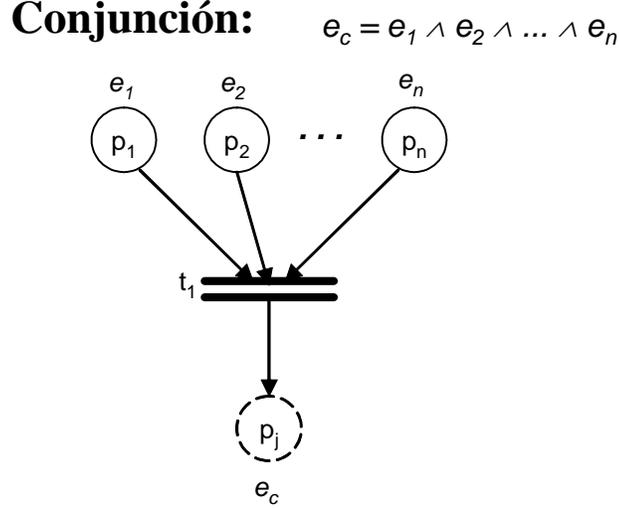


Figura 5.1: Estructura de CCPN para modelar al evento compuesto *conjunción*.

y contendrá información sobre los eventos constituyentes de la *conjunción*, es decir $M_2(p_j) = M_1(p_j) + C_{composition}(t, p_j)$. El lugar de salida p_j es de tipo virtual porque en éste lugar solo se está almacenando la información concerniente a los eventos que componen el evento compuesto, $Type(M_2(p_j)) = Type(C_{composition}(t, p_j))$.

En la figura 5.1 se muestra la estructura de CCPN para construir al evento compuesto *conjunción*. En esta figura se observa que a partir de la expresión $e_c = e_1 \wedge e_2 \wedge \dots \wedge e_n$ se forma la estructura de CCPN, donde cada evento constituyente $e_i, i = 1, 2, \dots, n$, son representados por los lugares de entrada p_i de la transición t_1 , $\bullet t = \{p_1, p_2, \dots, p_n\}$. Cuando ocurran los eventos e_1, e_2, \dots, e_n se asignarán los tokens correspondientes a los lugares p_1, p_2, \dots, p_n , respectivamente. Posteriormente, y de acuerdo a la regla de disparo de una transición, t_1 se disparará y se generará un nuevo token, que será la unión de los tokens que dispararon la transición, enviándolo hacia p_j , lugar de salida de t_1 . En otras palabras $C_{composition}(t_1, p_j) = C(p_1) \cup C(p_2) \cup \dots \cup C(p_n)$.

5.2.2. Disyunción

La *disyunción* es el evento compuesto donde solamente necesita que ocurra uno de sus eventos constituyentes para que éste ocurra. Dado que solamente necesitamos de la ocurrencia de tan solo un evento, el evento compuesto disyunción contendrá la información del evento que ocurra. Por lo tanto, al detectar a uno de los eventos constituyentes, éste pasa a ser el evento compuesto. Utilizando la CCPN, tomamos un lugar p_j de tipo virtual, $p_j \in P_{virtual}$, para alojar la ocurrencia de uno de los eventos que conforman a la disyunción, en el cual se alojará el token correspondiente del evento que ocurra. Los tokens que sean alojados en los lugares $p_i, i = 1, 2, \dots, n$, correspondientes a los eventos que conforman a la disyunción, deben ser enviados a p_j sin modificación alguna. En este caso, el tipo de transición que se utiliza para el paso de tokens es la transición de tipo *copy*, que toma el token de los lugares p_i y los envía al lugar p_j , utilizando por cada p_i una transición t_i , es decir, $\bullet t_i = \{p_i\}$ y $\bullet p_j = \{t_1, t_2, \dots, t_n\}$

Cuando la transición t_i es disparada, la marca del lugar p_i que representa al evento que ocurrió se modifica, eliminándole el token del evento, y se modifica la marca de p_j , agregándole el mismo token que se eliminó de p_i , tal y como se definió en el disparo de transiciones de tipo *copy*, descrito en el capítulo 3.

En la figura 5.2 se muestra la estructura de la CCPN que representa al constructor para el evento compuesto de la disyunción. El evento compuesto $e_c = e_1 \vee e_2 \vee \dots \vee e_n$ ocurrirá cuando ocurra un evento $e_i, i = 1, 2, \dots, n$; cada uno de los eventos e_i está representado por un lugar p_i , respectivamente. Cada p_i es lugar de entrada de una $t_i \in T_{copy}$, y todas éstas t_i tienen como lugar de salida al lugar p_j , el cual representa al evento e_c , es decir, $\bullet p_j = \{t_1, t_2, \dots, t_n\}$.

Cuando ocurra uno de los eventos e_1, e_2, \dots, e_n se asignará el token correspondiente en el lugar p_i . Posteriormente se disparará la transición t_i la cual enviará el mismo token a p_j . En otras palabras $C(p_j) \leftarrow C(p_i)$.

5.2.3. Negación

El evento compuesto *negación* es la no ocurrencia de un evento dentro de un cierto intervalo de tiempo. El evento que recibe como parámetro éste evento es monitoreado para determinar si no ocurre en el espacio de tiempo que nos interesa. Dado que en esencia éste evento compuesto significa la no existencia del evento e_1 dentro del lapso estipulado para que, al término de este

Disyunción: $e_c = e_1 \vee e_2 \vee \dots \vee e_n$

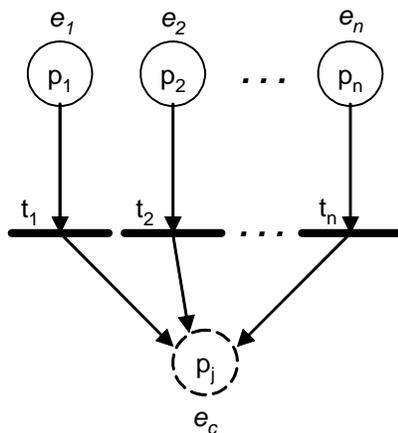


Figura 5.2: Estructura de CCPN para modelar al evento compuesto *disyunción*.

lapso, el evento compuesto e_c ocurra; en la CCPN utilizamos el tipo de arco *inhibidor*, tomado de la teoría de PN's [32], para ofrecer este comportamiento y soportar el evento compuesto de la *negación*.

Entonces, para modelar a éste evento compuesto mediante la CCPN se utiliza un arco inhibitorio $a \in A_{inh}$ que conecta al lugar de entrada p_1 con una transición t_1 , donde p_1 representa al evento e_1 y la transición t_1 es de tipo *compuesta*, $t_1 \in T_{comp}$, y el lugar de salida de t_1 es un lugar $p_2 \in P_{comp}$. Además, dentro de t_1 se almacena el intervalo de tiempo, denotado por D , especificado en la regla ECA. El intervalo D nos dice el periodo en el cual se estará monitoreando la no ocurrencia de e_1 . Si al término del intervalo D no se deposita ningún token en p_1 , es decir e_1 no ocurre, entonces t_1 es disparada y se envía un token a su lugar de salida p_2 , solamente para indicar que el evento compuesto e_c , representado por p_2 , ha ocurrido. La única información que contendrá el token enviado a p_2 será la estampa de tiempo, la cual será exactamente la misma que el punto final del intervalo D .

En la figura 5.3 se observa que para el evento compuesto $e_c = \sim e_1$ en $Int(sp, ep)$, el cual indica que el evento compuesto ocurrirá si no sucede el evento e_1 entre el intervalo de tiempo Int formado por un tiempo inicial sp y un tiempo final ep , el intervalo Int en la CCPN es denotado por D . En la figura aparece el arco inhibitorio que conecta a p_1 con t_1 , donde t_1 será disparada cuando se

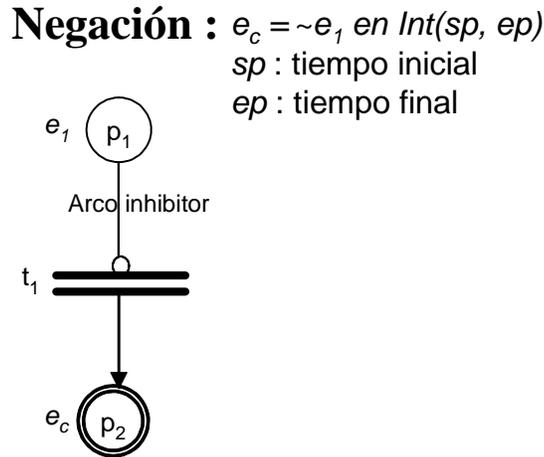


Figura 5.3: Estructura de CCPN para modelar al evento compuesto *negación*.

llegue al tiempo ep y $|M(p_1)| = 0$. En ese momento, t_1 envía un token hacia p_2 para señalar que el evento e_c ha ocurrido. La estampa de tiempo para el token enviado a p_2 será $\tau(M(p_2)) = ep$.

Si llegase a ocurrir el evento durante el intervalo de tiempo, entonces t_1 no se dispara y el evento compuesto no ocurre. Sin embargo, al final del intervalo D el token alojado en p_1 se elimina para posteriores formaciones del evento compuesto. De la misma manera, todos los tokens cuya estampa de tiempo está fuera del intervalo D son desechados.

5.2.4. Secuencia

El evento compuesto *secuencia* es semejante al evento compuesto *conjunción*, dado que deben de ocurrir todos los eventos que lo constituyen, sin embargo, en este caso el orden de ocurrencia sí es importante, y el evento compuesto *secuencia* ocurrirá siempre y cuando sus eventos constituyentes ocurran en el orden en que son especificados.

La modelación de éste evento compuesto mediante la CCPN se hace con la utilización de una transición t_1 de tipo *compuesta*, $t_1 \in T_{comp}$, porque este tipo de transiciones proporciona la capacidad de evaluar valores de tiempo. Los valores de tiempo son considerados para verificar el orden de ocurrencia de los eventos constituyentes y validar si efectivamente fue en la secuencia especificada. Los lugares de entrada de t_1 es el conjunto de lugares $p_i, i = 1, 2, \dots, n$; que representan a cada uno

de los eventos constituyentes e_i del evento compuesto *secuencia*, $\bullet t_1 = \{p_1, p_2, \dots, p_n\}$.

Para que la transición t_1 sea disparada el número de tokens de cada p_i debe ser mayor o igual que $w(p_i, t_1)$, además, de que los tokens considerados cumplan la restricción de orden de ocurrencia $\forall p_i[\tau(M(p_i)) < \tau(M(p_{i+1}))]$ para $i = 1, 2, \dots, n - 1$.

El valor de tiempo se toma de los tokens que son depositados en los lugares p_i , el cual indica el punto en el tiempo en que ocurrió el evento e_i representado por p_i .

Una vez la transición t_1 es disparada, se genera un token que contendrá la información de todos los tokens que participaron en el disparo de t_1 , y la estampa de tiempo que se asignará a éste nuevo token será el instante de tiempo en que ocurrió el evento representado por p_n . El nuevo token generado será enviado al lugar $p_j \in P_{virtual}$, el cual es el lugar de salida de t_1 , $\bullet t_1 = \{p_j\}$, expresando de esta manera la ocurrencia del evento compuesto *secuencia*. El lugar p_j es de tipo *virtual* porque en este lugar se alojará un token compuesto por diferentes tipos de tokens, existiendo una gran variedad de combinaciones de eventos constituyentes para formar al evento compuesto *secuencia*.

Después del disparo de la transición t_1 la marca de p_j cambiará al recibir al nuevo token generado, es decir, $M_2(p_j) = M_1(p_j) + C_{composition}(t_1, p_j)$. Los tokens utilizados en la composición del nuevo token son removidos de los lugares de entrada que se tomaron, $M_2(p_i) = M_1(p_i) - C_{enabled}(p_i, t_1)$. El tipo de dato de p_j será de acuerdo al tipo resultante de la composición de los tokens de cada p_i , $Type(M_2(p_j)) = Type(C_{composition}(t, p_j))$

La estructura de CCPN utilizada en la representación del evento compuesto *secuencia* se muestra en la figura 5.4, en la cual el evento compuesto *secuencia* $e_c = sec(e_1, e_2, \dots, e_n)$, está constituido por los eventos e_1, e_2, \dots, e_n . Los eventos constituyentes $e_i, i = 1, 2, \dots, n$, son representados mediante los lugares p_i , lugares de entrada de la transición $t_1 \in T_{comp}$, $\bullet t_1 = \{p_1, p_2, \dots, p_n\}$. Dada la ocurrencia de los eventos e_i , se procede a verificar su orden de ocurrencia mediante la expresión lógica $\tau(M(p_1)) < \tau(M(p_2)) < \dots < \tau(M(p_n))$. Si ésta expresión lógica se cumple, entonces se forma el token que será enviado hacia p_j , utilizando la información de todos los tokens evaluados, es decir $C_{composition}(t_1, p_j) = C(p_1) \cup C(p_2) \cup \dots \cup C(p_n)$. La estampa de tiempo del nuevo token será la estampa de tiempo del token tomado del lugar p_n , que en orden de ocurrencia es el que sucede al final, $\tau(M(p_j)) = \tau(M(p_n))$.

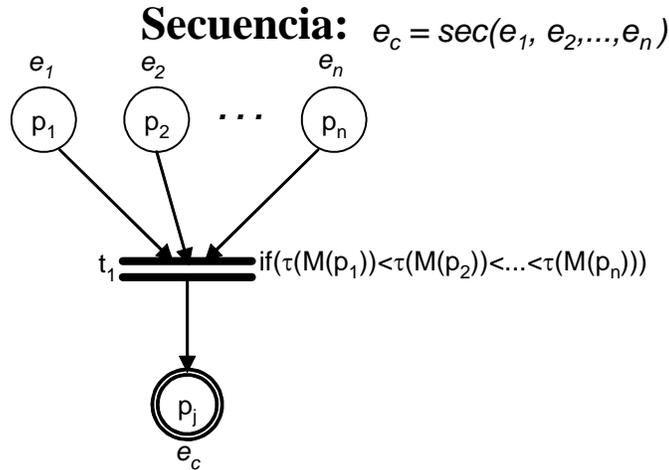


Figura 5.4: Estructura de CCPN para modelar al evento compuesto *secuencia*.

5.2.5. Simultáneo

El evento compuesto *simultáneo* es muy parecido al evento compuesto de secuencia, pero en este caso, la verificación del orden de ocurrencia de los eventos constituyentes es más fuerte, debido a que las estampas de tiempo de los eventos que lo constituyen debe de ser la misma, es decir, los eventos que conforman al evento compuesto deben de ocurrir al mismo tiempo.

La estructura de CCPN para modelar al evento compuesto *simultáneo* toma en cuenta como lugares de entrada $p_i, i = 1, 2, \dots, n$, a aquellos que representarán a los n eventos e_i que conforman al evento compuesto. Los lugares p_i son los lugares de entrada de una transición $t_1 \in T_{comp}$, la cual se encarga de verificar que las estampas de tiempo de los tokens, que son enviados por los lugares de entrada, tengan el mismo valor, es decir que $\forall p_i [\tau(M(p_i)) = \tau(M(p_{i+1}))], i = 1, 2, \dots, n - 1$. Cuando ocurren los eventos e_i , se depositan los tokens correspondientes en sus respectivos lugares p_i , entonces la transición t_1 evalúa la estampa de los tokens

En este caso, la política de consumo que se consideraría sería *acumulado*, con la diferencia de que se tomarían todos los tokens de cada p_i que habilitaron a la transición, si existe al menos un token en cada lugar cuya estampa de tiempo sea la misma en todos los casos entonces se dispara la transición t_1 generando un token a partir de la unión de todos los tokens que tienen la misma

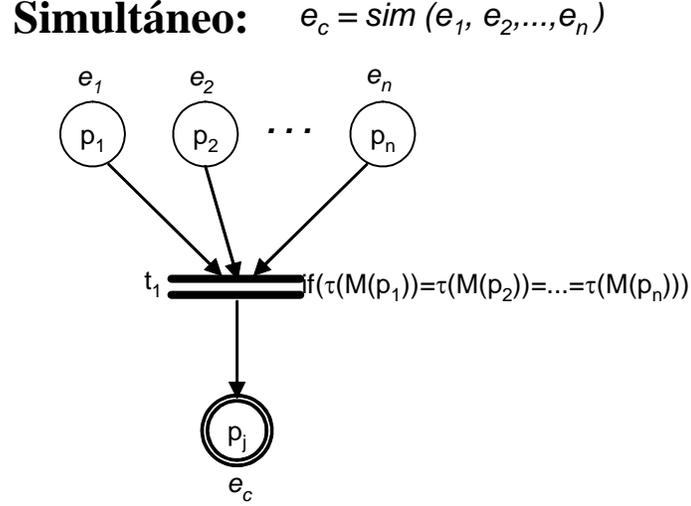


Figura 5.5: Estructura de CCPN para modelar al evento compuesto *conjunción*.

estampa de tiempo, desechando el resto de ellos. El token generado se envía al lugar de salida $p_j \in P_{comp}$, y la estampa de tiempo para el nuevo token será el de cualesquiera de los tokens evaluados, es decir, $\tau(M(p_j)) \leftarrow \tau(M(p_i))$.

Gráficamente, el evento compuesto *simultáneo* se representa con la CCPN como se muestra en la figura 5.5. En esta imagen, el evento compuesto $e_c = \text{sim}(e_1, e_2, \dots, e_n)$ contiene como eventos constituyentes a e_1, e_2, \dots, e_n , los cuales son representados mediante los lugares p_1, p_2, \dots, p_n , respectivamente. A partir de la ocurrencia de los eventos $e_i, i = 1, 2, \dots, n$, los tokens correspondientes a cada evento serán depositados en los lugares p_i respectivos. Si todos los lugares p_i tienen un número de tokens mayor o igual que el peso de su arco, $|M(p_i)| \geq w(p_i, t_1)$, entonces la transición $t_1 \in T_{comp}$ se habilita. Posteriormente t_1 verifica la estampa de tiempo de los tokens que la habilitaron y si existen tokens, de cada lugar, que tengan la misma estampa de tiempo, $\tau(M(p_1)) = \tau(M(p_2)) = \dots = \tau(M(p_n))$, entonces t_1 se dispara y manda el token generado $C_{composition}(t_1, p_j) = C(p_1) \cup C(p_2) \cup \dots \cup C(p_n)$ hacia el lugar de salida $p_j \in P_{comp}$.

5.2.6. Historia

Este evento compuesto es alcanzado cuando el evento e_1 que recibe como argumento ha ocurrido n veces, dentro de un periodo especificado. En la creación de este evento se consideran las n ocurrencias del evento e_1 para la evaluación de la regla ECA. Por lo tanto es necesario tomar en cuenta los parámetros con que se genera cada una de las ocurrencias de e_1 en el proceso de disparo de la regla ECA.

La modelación de éste evento compuesto utilizando la CCPN considera el uso de un lugar de entrada p_1 para representar al evento e_1 , y de una transición $t_1 \in T_{comp}$, en la cual se evaluará que la estampa de tiempo, de cada uno de los tokens que sean enviados desde p_1 a t_1 , estén dentro del intervalo D especificado. Sin embargo, en el caso del evento compuesto *historia* necesitamos considerar que el evento e_1 ocurra n veces, por lo tanto, el peso del arco que está dirigido de p_1 a t_1 tendrá un valor de n , es decir, $w(p_1, t_1) = n$. Una vez la transición t_1 sea disparada, se genera un token con información de los n tokens provenientes de p_1 que provocaron su disparo, $C_{composition}(t_1, p_2) = \cup_{i=1}^n M(p_1)_i$, donde i indica la posición en el orden de ocurrencia del evento e_1 , representado por $C(p_1)$. Este token es enviado a p_2 , lugar de salida de t_1 , $\bullet t_1 = \{p_2\}$.

La política de consumo que se considera para la conformación de éste evento compuesto es de modo *reciente*, porque basta con que se cumpla la condición de n ocurrencias de e_1 para que el evento compuesto suceda, sin necesidad de esperar al final del intervalo de tiempo.

En la figura 5.6 se observa que el evento compuesto *historia*, denotado por $e_c = times(n, e_1)$ en $Int(sp, ep)$, especifica la ocurrencia del evento e_1 en n ocasiones dentro del intervalo de tiempo Int , el cual inicia en el tiempo sp y termina en el tiempo ep . El evento e_1 es mapeado hacia el lugar p_1 y la transición $t_1 \in T_{comp}$ es la que verificará que las n ocurrencias de e_1 , representadas por tokens alojados en p_1 , entren dentro del intervalo de tiempo D , $\tau(M(p_1)_{i=1}^n) \in D$. La formación del nuevo token contendrá la información de los n tokens tomados de p_1 , y la estampa de token será la del n -ésimo token alojado en p_1 , $C_{composition}(t_1, p_2) = \cup_{i=1}^n M(p_1)_i$, $\tau(C_{composition}(t_1, p_2)) \leftarrow M(p_1)_n$.

Si el total de ocurrencias del evento e_1 dentro del intervalo D fue mayor que n , entonces los tokens $M(p_1)_j, j = n + 1, n + 2, \dots, |M(p_1)|$, serán eliminados de p_1 para futuras evaluaciones del evento compuesto, $|M(p_1)| = 0$.

Para eliminar los tokens correspondientes a los eventos que no ocurren dentro del intervalo de tiempo D proponemos que el peso del arco que conecta a p_1 con $t_1 \in T_{comp}$, dentro de la estructura

Historia: $e_c = \text{times}(n, e_1)$ en $\text{Int}(sp, ep)$

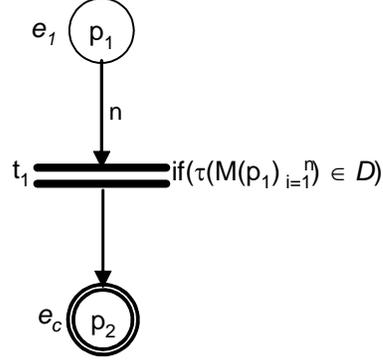


Figura 5.6: Estructura de CCPN para modelar al evento compuesto *historia*.

para formar eventos compuestos de tipo *historia*, sea adaptable, es decir, que fuera del intervalo D el peso del arco sea 1, mientras que dentro del intervalo de tiempo D el peso del arco sea n .

$$w(p_1, t_1) = \begin{cases} n & \text{Dentro del intervalo de tiempo } D. \\ 1 & \text{Fuera del intervalo de tiempo } D. \end{cases}$$

5.2.7. Primero

El evento compuesto *primero* sucede a partir de la primera ocurrencia de un evento e_1 que se está monitoreando dentro de un intervalo de tiempo especificado. Cabe mencionar que una vez tomada la primera ocurrencia de e_1 , las futuras ocurrencias del mismo evento dentro del intervalo de tiempo no serán consideradas para efectos de formar otra vez al evento compuesto.

La modelación de este evento compuesto, utilizando la CCPN, se logra con el uso de un lugar p_1 para representar al evento e_1 , y una transición $t_1 \in T_{comp}$ para evaluar la estampa de tiempo de los tokens enviados de p_1 a t_1 así como el orden de ocurrencia de los tokens. Dado que para que este evento suceda, es suficiente solamente una ocurrencia de e_1 dentro del intervalo D , entonces al entrar al intervalo de tiempo y detectar una ocurrencia de e_1 , entonces el lugar p_1 alojará el token correspondiente y será enviado a t_1 , provocando su disparo y, en consecuencia, la ocurrencia

Primero: $e_c = first(e_1)$ en $Int(sp, ep)$

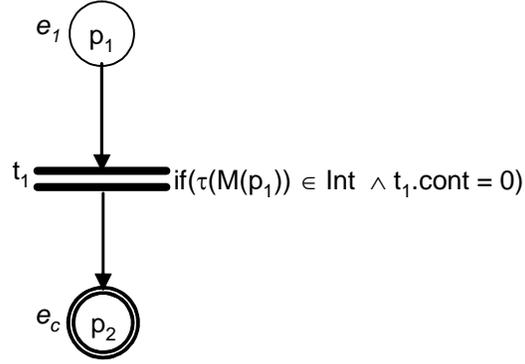


Figura 5.7: Estructura de CCPN para modelar al evento compuesto *primero*.

del evento compuesto *primero*. Posteriores ocurrencias de e_1 serán descartadas, aún dentro del intervalo D . Para eliminar las ocurrencias posteriores a la primera, dentro de t_1 se almacena una variable para contar el número de tokens evaluados dentro de cada periodo, reseteando su valor al inicio del intervalo D . Entonces, si el valor de esta variable es mayor que 1, significa que la primera ocurrencia de e_1 ya se dio.

En la figura 5.7 puede observarse que el evento compuesto $e_c = first(e_1)$ en $Int(sp, ep)$ es mapeado a una estructura de CCPN. El evento e_1 se representa mediante un lugar p_1 , el cual es el lugar de entrada para una transición $t_1 \in T_{comp}$. La transición t_1 verificará que la estampa de tiempo del token enviado por p_1 , correspondiente al evento e_1 , ocurra dentro del intervalo de tiempo D , $\tau(M(p_1)) \in D$. Además de verificar si el token es el primero en ocurrir desde el inicio del intervalo de tiempo D .

Si $M(p_1)$ es la primera ocurrencia dentro de D , entonces éste token, se envía a hacia el lugar $p_2 \in P_{comp}$, $t^\bullet = \{p_2\}$. Como es el mismo token el que se desplaza, entonces $C_{composition}(t_1, p_2) = M(p_1)_1$.

5.2.8. Último

El evento compuesto *último* sucede cuando, al terminar el intervalo de tiempo especificado, ha ocurrido al menos una vez el evento e_1 que recibe como argumento, tomando para formar el evento compuesto a la última ocurrencia de e_1 .

Para modelar a este evento compuesto, utilizando la CCPN, se utiliza una estructura semejante a la del evento compuesto *primero*, sin embargo, en este caso no se utiliza un contador de tokens y la transición $t_1 \in T_{comp}$ se dispara hasta el final del intervalo de tiempo D . Para efectos de formar al evento compuesto, se toma la información del token correspondiente a la última ocurrencia de e_1 , es decir, si hubo n ocurrencias del evento e_1 dentro del intervalo de tiempo D , entonces se toma a $M(p_1)_n$ para formar el token que será enviado hacia el lugar p_2 , lugar de salida de t_1 , $t_1^\bullet = \{p_2\}$. Por lo tanto, la composición del nuevo token será $C_{composition}(t_1, p_2) = M(p_1)_n$. La estampa de tiempo que se asignará al nuevo token es el valor del punto final del intervalo D . La ocurrencia del evento *último* se expresa con la colocación del nuevo token generado dentro del lugar p_2 .

La figura 5.8 muestra la representación gráfica para el evento compuesto *último*, en la cual, la expresión $e_c = last(e_1)$ en $Int(sp, ep)$ denota que el evento compuesto e_c ocurrirá si al final del intervalo $Int(sp, ep)$ ha ocurrido al menos una vez el evento e_1 , tomando los datos de la última ocurrencia de e_1 como los datos de e_c . En la CCPN obtenida, se tiene que $\bullet t_1 = \{p_1\}$ y $t_1^\bullet = \{p_2\}$, donde p_1 representa al evento e_1 y p_2 representa la ocurrencia del evento compuesto e_c . Además, $t_1 \in T_{comp}$ realiza la verificación del intervalo de tiempo D , y toma a la i -ésima ocurrencia de e_1 , $M(p_1)_n$.

Los tokens que son enviados desde p_1 hacia t_1 y que no cumplen con la condición del intervalo, así como aquellos que sí cumplen la condición del intervalo pero que no fueron el último en ocurrir, son eliminados del lugar p_1 .

5.2.9. Alguno

El evento compuesto *alguno* sucede cuando, dentro de un conjunto de n eventos posibles, si ocurren al menos m eventos, $m \leq n$, entonces el evento compuesto *alguno* toma lugar.

Para modelar a éste evento compuesto utilizando la CCPN es necesario utilizar transiciones de tipo *copy* y un lugar de tipo *virtual*, además de los lugares necesarios para representar a los eventos e_i , que participan en la composición del evento compuesto, y e_c que denota al evento compuesto y

Ultimo: $e_c = \text{last}(e_1)$ en $\text{Int}(sp, ep)$

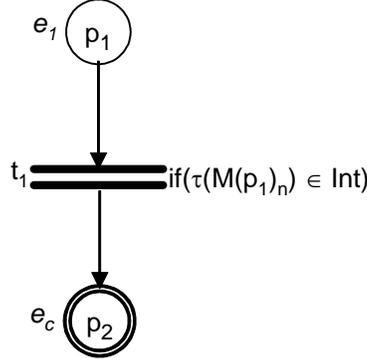


Figura 5.8: Estructura de CCPN para modelar al evento compuesto *último*.

de la transición t_{comp} para proporcionar la capacidad de detección del evento compuesto.

Cada evento $e_i, i = 1, 2, \dots, n$, se representa con un lugar p_i , los cuales, cada uno de ellos, es un lugar de entrada para una transición $t_i \in T_{copy}$, respectivamente. Todas las transiciones t_i envían el token recibido hacia un mismo lugar $p_{n+1} \in T_{virtual}$, el cual los alojará hasta completar la cuota de m tokens de diferente tipo de color.

La transición t_{n+1} verificará que efectivamente, los tipos de color, de los tokens que la habilitaron, sean diferentes, ya que no es válido que haya m ocurrencias pero del mismo evento e_i , sino que deben de ser m ocurrencias de m diferentes eventos e_i . Si t_{n+1} se dispara, entonces se forma un nuevo token a partir de la unión de los tokens que dispararon a t_1 , el cual es enviado hacia el lugar p_j , lugar de salida de t_{n+1} , $t_{n+1}^\bullet = \{p_j\}$. La formación del nuevo token se define por $C_{composition}(t_{n+1}, p_j) = \cup_{i=1}^n M(p_i)$, para valores de i que sí participaron en la ocurrencia del evento compuesto.

Gráficamente, la estructura utilizada en la formación del evento compuesto *alguno* se muestra en la figura 5.9. En la figura se observa que a cada evento e_1, e_2, \dots, e_n le corresponde un lugar p_1, p_2, \dots, p_n , respectivamente. Además, a cada lugar p_1, p_2, \dots, p_n , le corresponde una transición $t_1, t_2, \dots, t_n \in T_{copy}$, respectivamente. Estas transiciones tomarán el token correspondiente a la ocurrencia de un evento e_i y lo mandarán hacia un mismo lugar $p_{n+1} \in P_{virtual}$, el cual los alojará

Alguno: $e_c = any(m, e_1, e_2, \dots, e_n)$

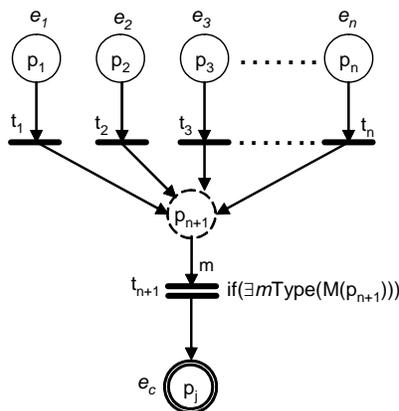


Figura 5.9: Estructura de CCPN para modelar al evento compuesto *alguno*.

mientras se acumulan m tokens, de diferente tipo de color, para formar el evento compuesto *alguno*. La validación de que existan m tokens de distinto tipo de color se realiza en la transición $t_{n+1} \in T_{comp}$. La transición t_{n+1} se dispara si se cumple que $\exists mType(M(p_{n+1}))$. Donde se indica que exista al menos m diferentes tipos de colores dentro de la marca M del lugar p_{n+1} .

5.3. Algoritmo de conversión de un evento compuesto a una CCPN

Para llevar a cabo la conversión de eventos, tanto primitivos como compuestos, se utiliza una estructura para almacenarlos, la cual contiene los atributos *tipo*, para almacenar el tipo de evento ya sea primitivo o compuesto; *nombre*, para almacenar el nombre del evento de acuerdo a la nomenclatura descrita anteriormente; y *comp*, para almacenar los nombres de los eventos que lo conforman en el caso de ser un evento compuesto:

```
Estructura EV {
    tipo,
    nombre,
    comp,
};
```

La creación de la CCPN para los eventos que forman parte de la base de reglas ECA se abordó en el capítulo 4, donde se describe la CCPN, sin embargo en este punto de la tesis aún no se hacía mención de los eventos compuestos y se habló solamente de los eventos primitivos, por lo que se pospuso la descripción del algoritmo hasta este capítulo, donde se describen, precisamente, los eventos compuestos.

El algoritmo que se utiliza para convertir los eventos de la base de reglas ECA en una CCPN se describe en el diagrama de flujo de la figura 5.10, el cual recibe como entrada el archivo de extensión *.eca* y produce como salida la CCPN de los eventos que participan en la base de reglas ECA.

Se define una variable del tipo de la estructura *EV* denominada *evStr* y el conjunto *E* que contendrá a los eventos que participan en la base de reglas. Del archivo con extensión *.eca* se toman los eventos que participan en el disparo de las reglas ECA y comenzamos con la parte iterativa del algoritmo mientras existan más eventos que leer.

El evento que se lee se le asigna a la variable *evento*, se verifica si el evento es primitivo o se trata de un evento compuesto. Si es un evento primitivo, al campo *tipo* de la variable *evStr* se le asigna “primitivo”, y al campo *nombre* se le asigna el nombre del evento que se está leyendo y se agrega *evStr* al conjunto de eventos *E*.

Si el tipo de evento no es primitivo (se trata de un evento compuesto), entonces al campo *tipo* de la variable *evStr* se le asigna la palabra “compuesto”, al campo *nombre* se le asigna el nombre del evento en cuestión, y al campo *comp* se le asignan los eventos que conforman al evento compuesto.

Teniendo representado al evento (primitivo o compuesto), éste es agregado al conjunto de eventos *E* ($E \leftarrow E \cup \{evStr\}$).

Se verifican si hay más eventos en el archivo *.eca*, a los cuales se les aplica el mismo procedimiento descrito.

Posteriormente se recorren cada uno de los elementos del conjunto *E* para crear su estructura de CCPN respectiva, por medio del módulo “**Evento a CCPN**”.

El diagrama de flujo para el módulo “**Evento a CCPN**” se muestra en la figura 5.11.

Este algoritmo recibe como entrada un elemento del conjunto *E* que es del tipo de la estructura *EV*. Primero se verifica el tipo de evento, si es de tipo primitivo se crea un lugar $p_1 \in P_{prim}$, se le asigna al nombre del lugar el mismo que tiene el evento (*evento.nombre*), y si no existe un elemento con las características de p_1 dentro del conjunto *P* entonces se agrega ($P \leftarrow P \cup \{p_1\}$). Si ya hubiera

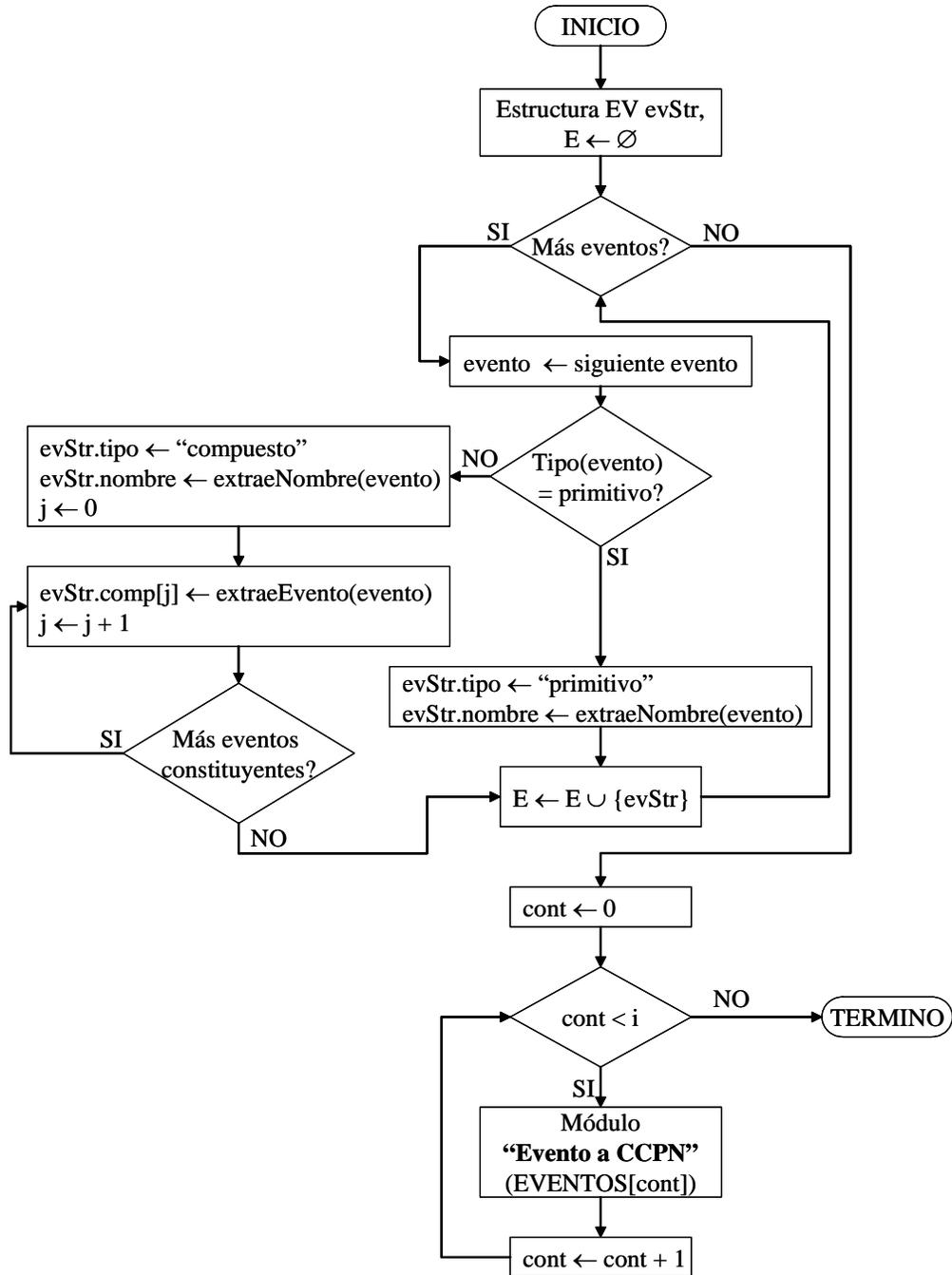


Figura 5.10: Diagrama de flujo del algoritmo que genera la CCPN para los eventos de las reglas ECA.

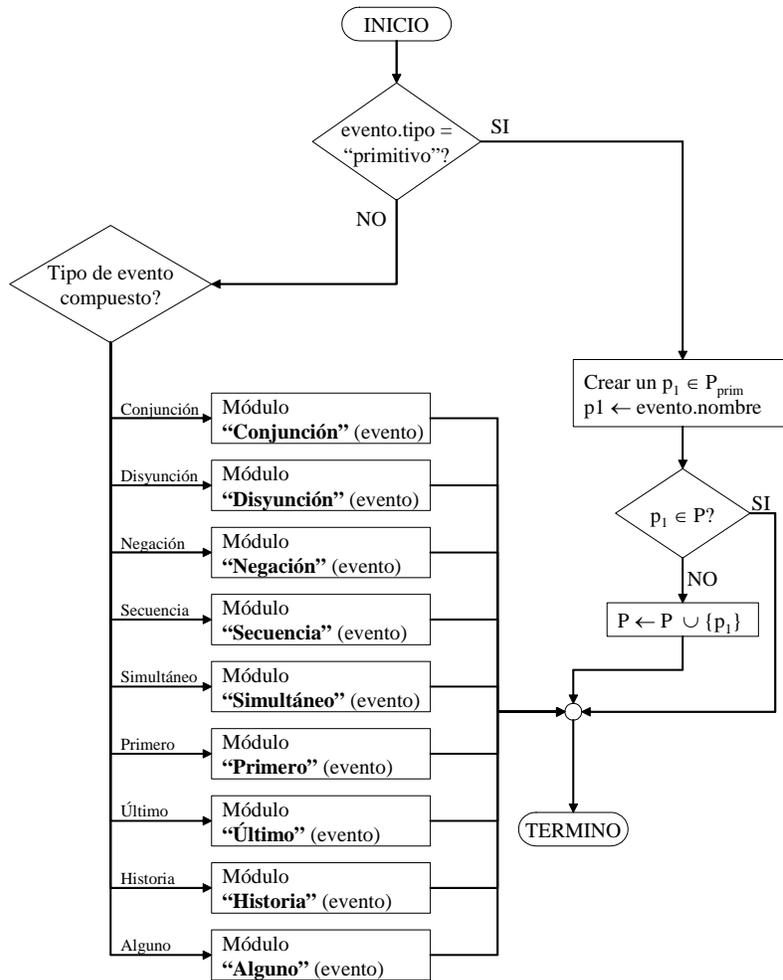


Figura 5.11: Diagrama de flujo del módulo “Evento a CCPN” (Estructura EV evento).

un elemento para denotar a éste evento entonces no se grega p_1 a P y se termina la ejecución del algoritmo.

Si se tratara de un evento compuesto, se verifica qué tipo de evento compuesto es. Si es el evento compuesto *conjunción* se llama al módulo “**Conjunción**”, para el evento compuesto *disyunción* se llama al módulo “**Disyunción**”, y así sucesivamente hasta el evento compuesto *alguno* que utiliza el módulo “**Alguno**”.

Para cada uno de los módulos de generación de los eventos compuestos, se manda como parámetro el elemento $evento \in E$ que se recibe como parámetro.

El algoritmo del módulo “**Conjunción**” se muestra en la figura 5.12.

En este algoritmo primero creamos la transición $t_1 \in T_{comp}$, a la cual se conectarán todos los eventos que forman parte de la conjunción. Posteriormente, para cada evento constituyente se verifica si ya existe un lugar $p \in P$ que represente a dicho evento. Si aún no se encuentra representado por un lugar p , entonces se llama al módulo “**Evento a CCPN**”, y el lugar obtenido con este módulo es asignado a p y se crea un arco para conectar a p con t_1 . En caso de que ya exista un lugar $p \in P$ que represente al evento constituyente entonces verificamos si p no es lugar de entrada de alguna transición ($p^\bullet = \emptyset$). Si $p^\bullet = \emptyset$, es decir p no es el lugar de entrada de ninguna transición, entonces se crea un arco para conectar a p con la transición que creamos para denotar al evento compuesto t_1 . Si $p^\bullet \neq \emptyset$ significa que p es lugar de entrada de una transición, digamos t_2 . Si $t_2 \in T_{copy}$ significa que ya existen al menos dos transiciones que están utilizando a p como su lugar de entrada por medio de copias de p , podemos agregar otra copia de p para utilizarla como lugar de entrada a t_1 ; en otras palabras, creamos un $p_1 \in P_{copy}$, creamos un arco desde $t_2 \in T_{copy}$ hacia $p_1 \in P_{copy}$, y creamos otro arco para conectar p_1 con $t_1 \in T_{comp}$. Por el contrario, si $Type(t_2) \neq Copy$, necesitamos agregar una transición copy para enviar los tokens que lleguen a p tanto a t_1 como a t_2 ; primero creamos una $t_3 \in T_{copy}$, creamos los lugares $p_1, p_2 \in P_{copy}$, eliminamos el arco que conecta a p con t_2 , y creamos los siguientes arcos (p, t_3) , (t_3, p_1) , (t_3, p_2) , (p_1, t_1) , (p_2, t_1) .

Cuando ya no existan más eventos constituyentes, se procede a crear un lugar $p_3 \in P_{virtual}$, el cual representará al evento compuesto *conjunción*, y se conecta la transición $t_1 \in T_{comp}$ con el lugar p_3 mediante el arco (t_1, p_3) . Quedando como lugar del evento compuesto el lugar p_3 .

El algoritmo del módulo “**Disyunción**” se muestra en la figura 5.13.

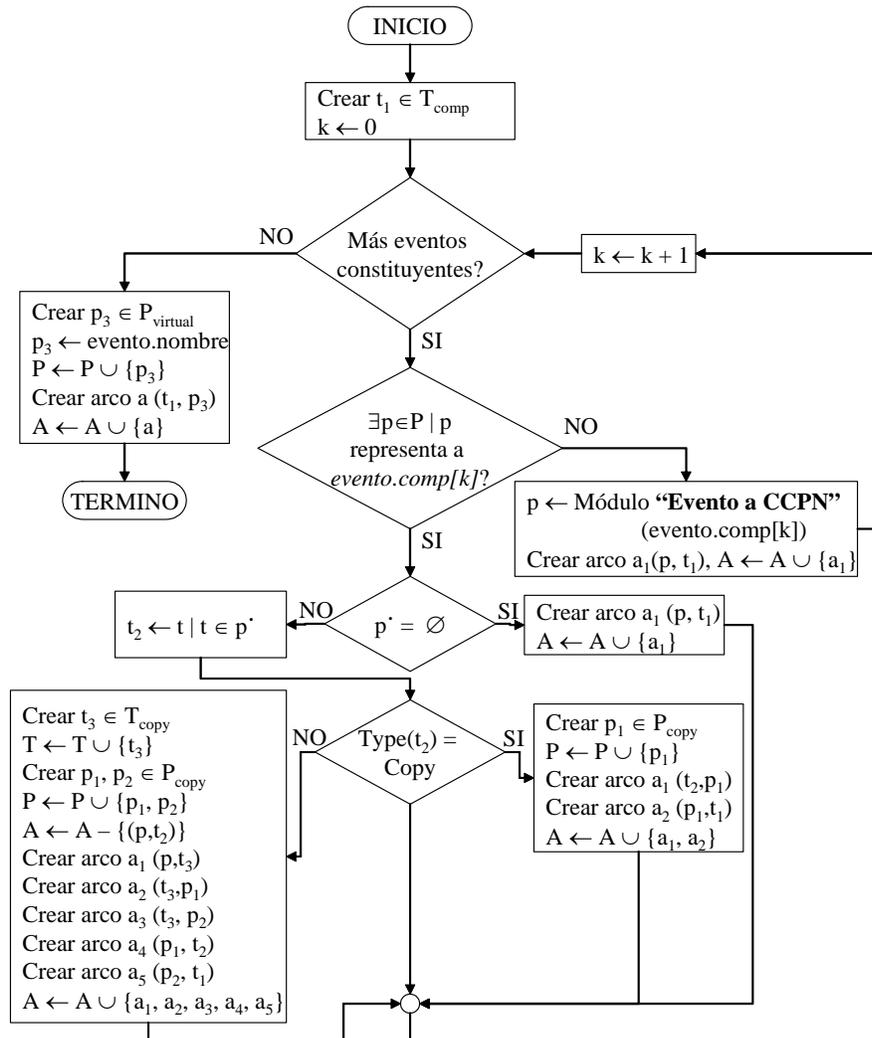


Figura 5.12: Diagrama de flujo del módulo que genera la CCPN para el evento compuesto *conjunción*.

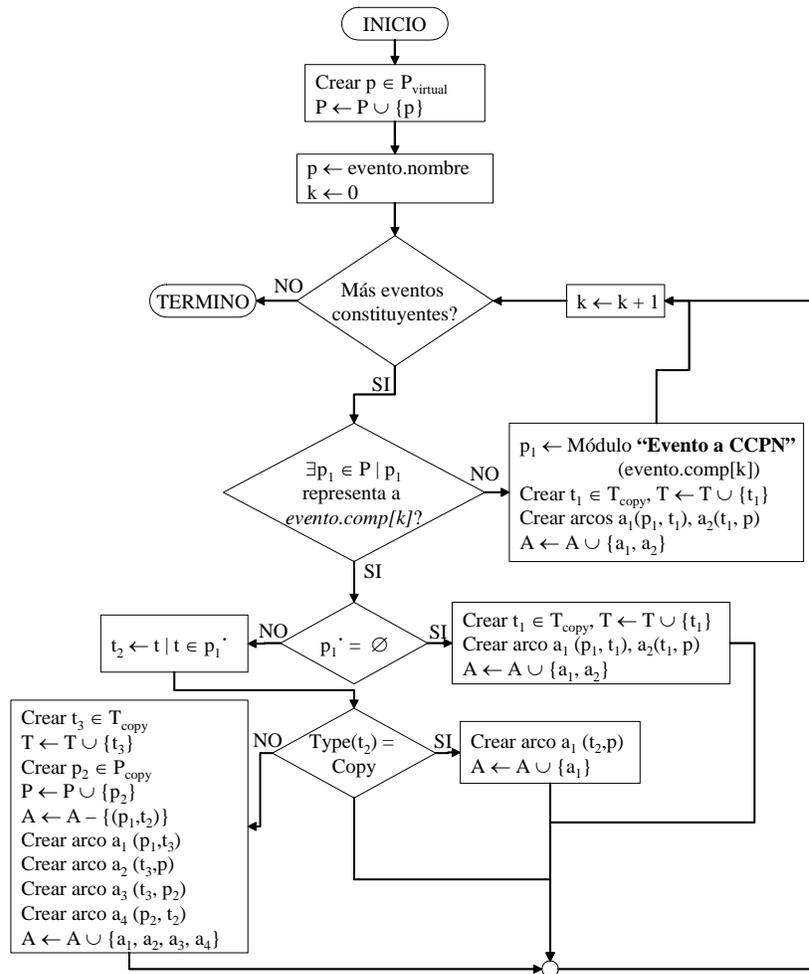


Figura 5.13: Diagrama de flujo del módulo que genera la CCPN para el evento compuesto *disyunción*.

En este algoritmo, primero se crea un lugar virtual $p \in P_{virtual}$ donde confluirán los eventos que conforman a la disyunción de eventos, se le asigna a p el nombre del evento y se inicializa la variable k a cero (k se utiliza para contabilizar los eventos que participan en la disyunción).

Para cada uno de los eventos constituyentes se pregunta si ya existe un lugar p_1 que represente al evento en cuestión ($evento.comp[k]$). Si no existe tal lugar, entonces se llama al módulo **“Evento a CCPN”** para obtener la estructura correspondiente a $evento.comp[k]$, y asignarle el lugar del evento a p_1 . Se crea una transición $t_1 \in T_{copy}$ y se conectan por medio de arcos el lugar p_1 con la transición t_1 , y a la vez se conecta a la transición t_1 con p .

Si ya existe un lugar p_1 que denote a $evento.comp[k]$, entonces verificamos si p_1 está siendo utilizado como lugar de entrada. Si p_1 no es lugar de entrada de alguna transición ($p_1^\bullet = \emptyset$) entonces se crea una transición $t_1 \in T_{copy}$ y se conecta p_1 con la transición recién creada ($a_1(p_1, t_1)$), además, se conecta a t_1 con el lugar virtual p ($a_2(t_1, p)$).

Por otro lado, si p_1 ya es el lugar de entrada de una transición ($p_1^\bullet \neq \emptyset, p_1^\bullet = \{t_2\}$), se verifica que tipo de transición es t_2 . Si $t_2 \in T_{copy}$ simplemente se conecta esta transición con el lugar virtual ($a(t_2, p)$); sin embargo, si $t_2 \notin T_{copy}$ ($t_2 \in T_{regla} \cup T_{comp}$), entonces se crea una transición $t_3 \in T_{copy}$ y se crea un lugar $p_2 \in P_{copy}$, se elimina el arco que conecta al lugar p_1 con la transición t_2 ($A \leftarrow A - \{a(p_1, t_2)\}$), y finalmente se crean los arcos para conectar p_1 con t_3 ($a_1(p_1, t_3)$), t_3 con p ($a_2(t_3, p)$), t_3 con p_2 ($a_3(t_3, p_2)$), y p_2 con t_2 ($a_4(p_2, t_2)$).

El algoritmo del módulo **“Negación”** se muestra en la figura 5.14.

En el diagrama de flujo puede observarse que primero se crea una transición $t_1 \in T_{comp}$, posteriormente verificamos si ya existe un lugar p dentro de la CCPN que represente al evento $evento.comp[0]$ (el subíndice es 0 porque la estructura para la negación solo necesita un lugar de entrada). Si no existe, se llama al módulo **“Evento a CCPN”** para crear la estructura correspondiente a $evento.comp[0]$, asignándola al lugar p , y se conecta el lugar p con la transición t_1 por medio de un arco inhibidor ($\hat{a}_1(p, t_1)$).

Si ya existe un lugar p que represente al evento $evento.comp[0]$, entonces verificamos si p está siendo utilizado como lugar de entrada. En caso negativo ($p = \emptyset$), existe un lugar p pero éste no está conectado con ninguna transición, entonces se crea un arco inhibidor para conectar a p con t_1 ($\hat{a}(p, t_1)$). Por otro lado, si p está siendo utilizado como lugar de entrada para una transición ($p \neq \emptyset$), se verifica que tipo de transición es a la que está conectado p . Sea t_2 la transición a la que se

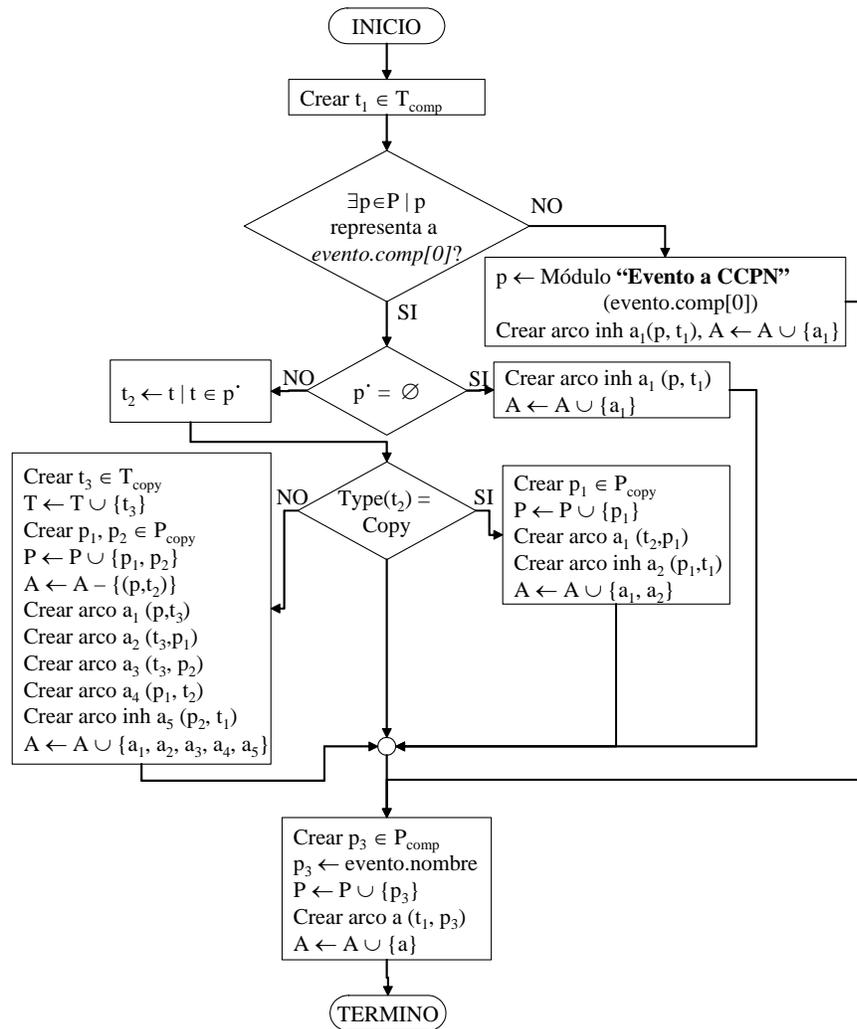


Figura 5.14: Diagrama de flujo del módulo que genera la CCPN para el evento compuesto *negación*.

encuentra conectada p ($p^\bullet = \{t_2\}$), si $t_2 \in T_{copy}$ entonces creamos un lugar $p_1 \in P_{copy}$ y conectamos t_2 con este nuevo lugar p_1 , y creamos un arco inhibidor para conectar p_1 con t_1 ($\hat{a}(p_1, t_1)$). Ahora, si $t_2 \notin T_{copy}$, significa que $t_2 \in T_{regla} \cup T_{comp}$, creamos entonces la estructura para generar las copias de p ; para esto creamos una $t_3 \in T_{copy}$, creamos dos lugares de tipo copy $p_1, p_2 \in P_{copy}$, eliminamos el arco que conecta a p con t_2 ($A \leftarrow A - \{(p, t_2)\}$), creamos arcos normales para conectar p con t_3 ($a_1(p, t_3)$), t_3 con p_1 ($a_2(t_3, p_1)$), t_3 con p_2 ($a_3(t_3, p_2)$), p_1 con t_2 ($a_4(p_1, t_2)$), y un arco inhibidor para conectar p_2 con t_1 ($\hat{a}_5(p_2, t_1)$).

Después que se conecta el lugar p (que representa al evento que forma parte del evento compuesto) con la transición compuesta t_1 , se crea un lugar $p_3 \in P_{comp}$, que denotará al evento compuesto en cuestión, a p_3 se le asigna el nombre del evento y se crea un arco para conectar a t_1 con p_3 .

El algoritmo del módulo “**Secuencia**” se muestra en la figura 5.15.

Este algoritmo es similar al del evento *conjunción*, con la diferencia de que al terminar de conectar a todos los eventos constituyentes, a la transición $t_1 \in T_{comp}$ se le asigna la condición $if(\forall token_i(p_i, c_i, stamp_i)[p_i \in \bullet t_1, stamp_i < stamp_{i+1}])$, es decir, que para que la transición t_1 se dispare se debe de cumplir, además de que $\forall p \in \bullet t_1[M(p) \geq w(p, t_1)]$, que la estampa de tiempo de los tokens esté en orden cronológico de acuerdo al lugar de entrada. Otra de las diferencias con el algoritmo del módulo “**Conjunción**” es que en este caso se crea un lugar $p_3 \in P_{comp}$ y no un $p_3 \in P_{virtual}$.

El algoritmo del módulo “**Simultáneo**” es mostrado en la figura 5.16. La única diferencia que existe con el algoritmo anterior es la condición que se asigna a t_3 , ya que en este caso se debe cumplir de que las estampas de tiempo tengan el mismo valor, es decir, que la ocurrencia de los eventos representados en los tokens haya sido al mismo tiempo.

Para el caso de los eventos compuestos *primero* y *último*, el algoritmo es exactamente el mismo, la diferencia es al momento de la ejecución de la CCPN, cuando en el evento compuesto *primero* toma la primera ocurrencia del evento y en el caso del evento compuesto *último* toma la última ocurrencia del evento. En la figura 5.17 se tiene el diagrama de flujo del algoritmo para crear la estructura de CCPN correspondiente. La lógica de éste algoritmo es similar a la del algoritmo para el módulo “**Negación**”, con la notable diferencia que en lugar de utilizar arcos inhibidores (como el caso de la estructura para el evento compuesto *negación*) se utilizan arcos normales.

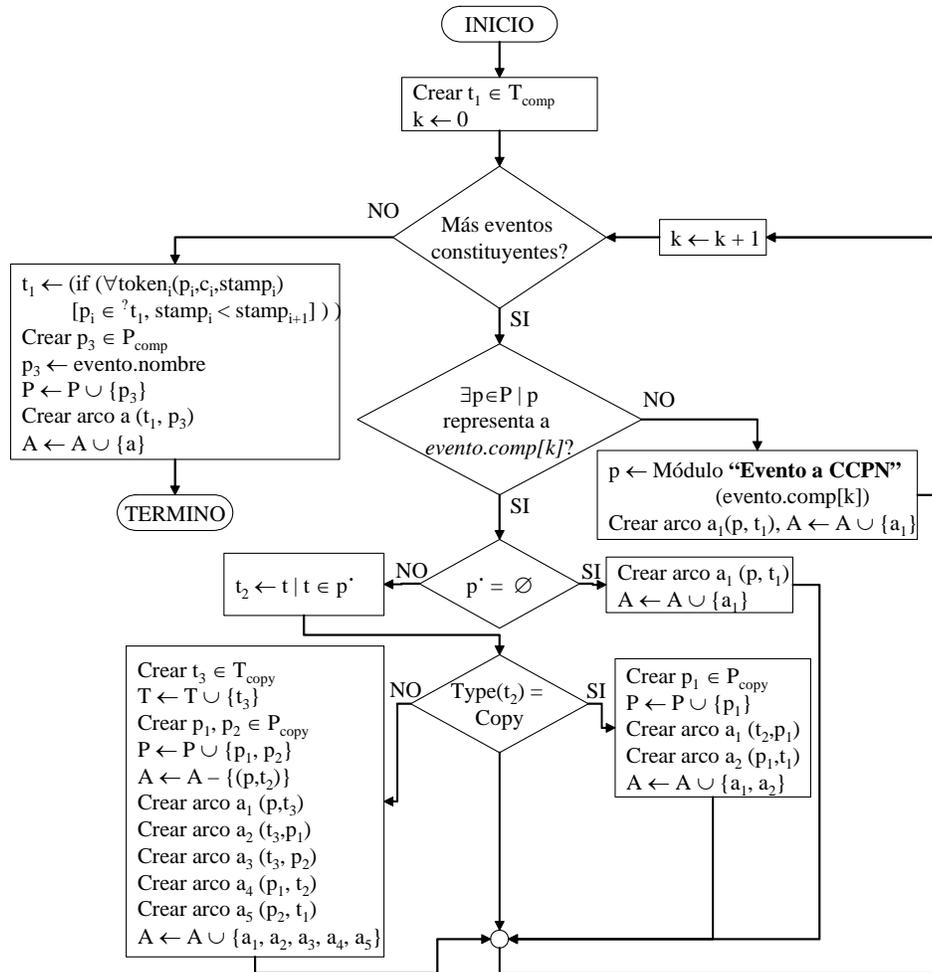


Figura 5.15: Diagrama de flujo del módulo que genera la CCPN para el evento compuesto *secuencia*.

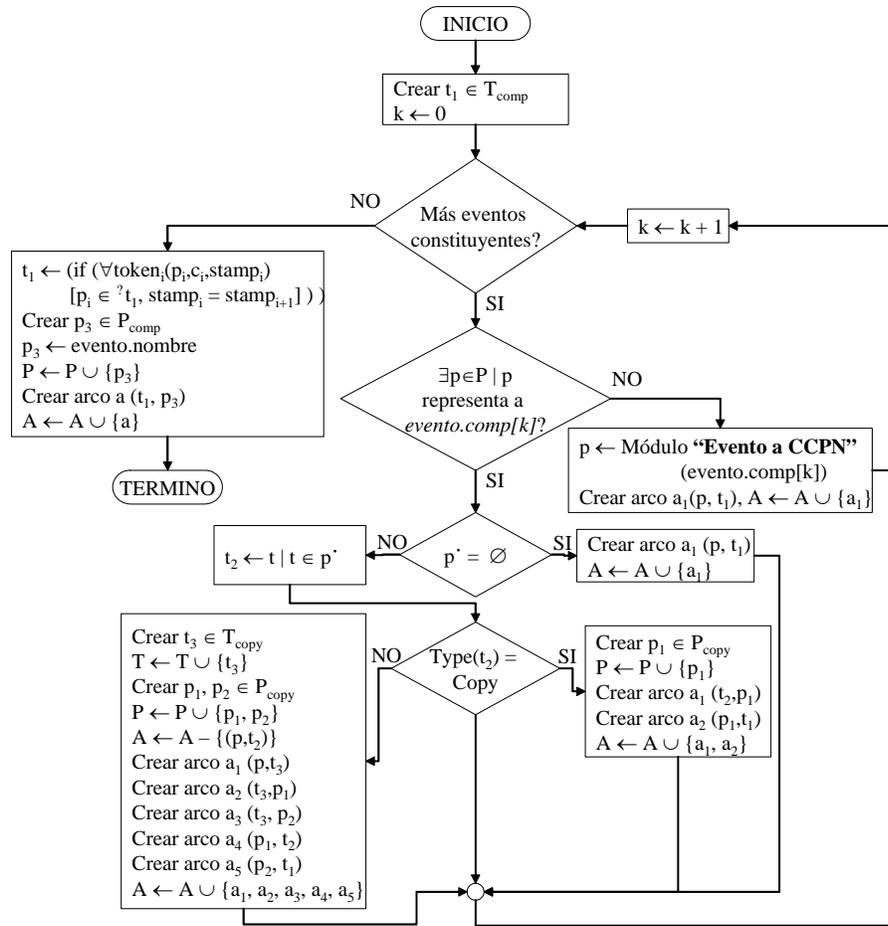


Figura 5.16: Diagrama de flujo del módulo que genera la CCPN para el evento compuesto *simultáneo*.

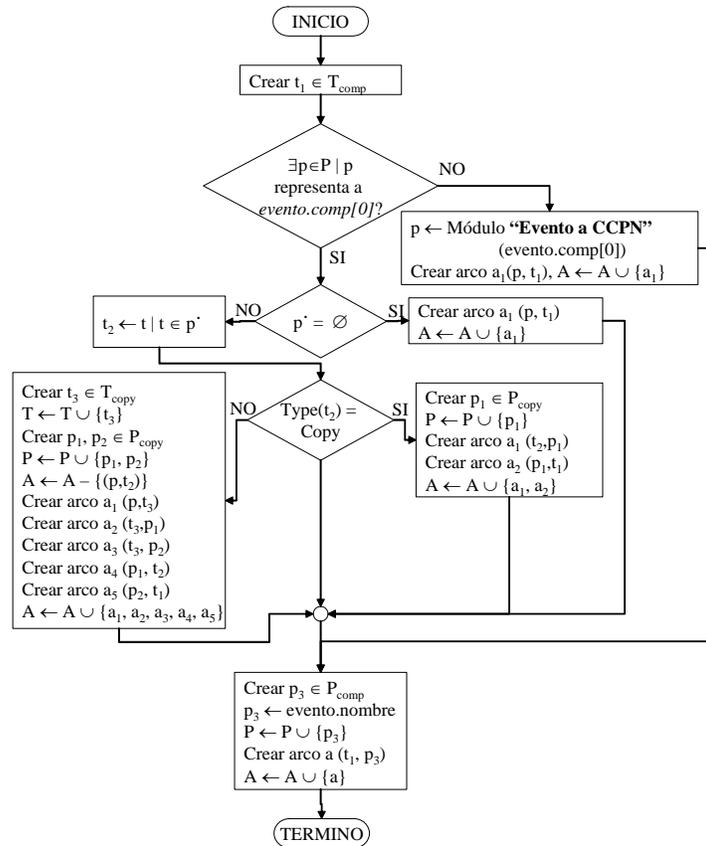


Figura 5.17: Diagrama de flujo del módulo que genera la CCPN para los eventos compuestos *primero* y *último*.

Para el evento compuesto *historia* es necesario conocer la cantidad n de veces que debe ocurrir el evento constituyente para que sea considerado como evento compuesto. La estructura de éste evento compuesto se deriva de la estructura descrita en el párrafo anterior, sin embargo como es necesario que el evento constituyente ocurra n veces entonces se agrega esta restricción. A diferencia del algoritmo anterior, en el algoritmo mostrado en la figura 5.18 se asigna el valor n al peso del arco que conecta a t_1 desde su lugar de entrada, el valor para n se lee utilizando la función `extraerN()`; sea $\bullet t_1 = \{p\}$, $w(p, t_1) = n$.

El algoritmo para la generación de la estructura de la CCPN para el evento compuesto *alguno* está basado en el algoritmo del módulo “**Disyunción**”, hasta el punto en que se agregan todos los eventos constituyentes al lugar virtual p . Cuando ya no se tienen más eventos constituyentes que agregar se crea una transición $t_4 \in T_{comp}$ y se crea un lugar $p_3 \in P_{comp}$; a p_3 se le asigna el nombre del evento compuesto y se crean los arcos para conectar a p con t_4 ($a_1(p, t_4)$) y a t_4 con p_3 ($a_2(t_4, p_3)$). A t_4 se le asigna la condición $if(\exists mType(M(p_{n+1})))$, $\bullet t_4 = \{p_{n+1}\}$, y al arco que conecta a p con p_4 se le asigna el valor de m que contiene la definición del evento compuesto *alguno*; en otras palabras, t_4 se disparará si en el lugar p_{n+1} existen m tipos diferentes de marcas (tokens) provenientes de los lugares que representan a los eventos constituyentes del evento compuesto *alguno*.

5.4. Ejemplo

Para mostrar un ejemplo de representación de eventos compuestos a partir de la CCPN se utilizan las siguientes tres reglas ECA:

Regla 1: Cuando el registro de un empleado es agregado a la BD o cuando el salario de un empleado es modificado, si el salario del empleado es mayor que el salario del gerente, entonces el salario del empleado es modificado asignándole el 20% del salario del gerente.

Regla 2: Cuando un empleado es nuevo en la empresa y en su primer día obtiene altas ventas, entonces su salario es incrementado en un 10%.

Regla 3: Cuando no se agregaron registros en la tabla de ventas en un día de trabajo normal, entonces se debe notificar al gerente sobre el estado de las ventas.

e_0 , e_1 y e_2 son utilizados para representar las instrucciones *insert empleado*, *update empleado.salario* e *insert ventas*, respectivamente. Las reglas anteriores son escritas en la sintaxis utilizada para lograr la conversión a CCPN, quedando de la siguiente manera:

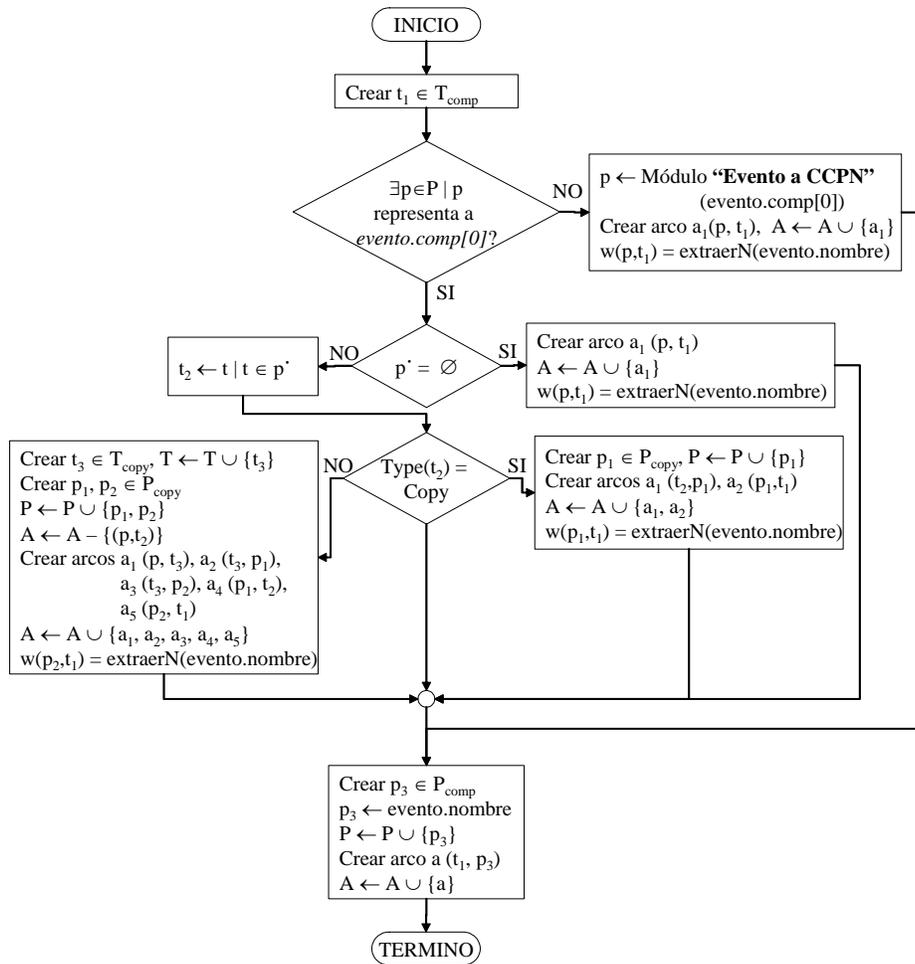


Figura 5.18: Diagrama de flujo del módulo que genera la CCPN para el evento compuesto *historia*.

Rule 1:

```

on  $or(e_0, e_1)$ 
if empleado.salario > gerente.salario
then empleado.salario = gerente.salario * 0.20;

```

Rule 2:

```

on  $and(e_1, e_2)$ 
if true
then empleado.salario = empleado.salario * 1.10;

```

Rule 3:

```

on  $not(e_2)$ 
if true
then insert into MSG ('Gerente', 'No se registraron ventas');

```

En este ejemplo se tienen tres eventos compuestos, $or(e_0, e_1)$, $and(e_1, e_2)$, y $not(e_2)$. ECAPNSim convertirá estas tres reglas en su CCPN correspondiente, la cual se muestra en la figura 5.19. En esta figura los nombres de los lugares que inician con E denotan eventos primitivos, mientras que los eventos que inician con EC denotan a los eventos compuestos. Las transiciones $T2$ y $T3$ son transiciones compuestas correspondientes a la parte del evento de las reglas 2 y 3, respectivamente. Las transiciones $T5$, $T6$ y $T7$ son transiciones tipo T_{regla} correspondientes a las reglas 1, 2, y 3 respectivamente. Los lugares $E9$ y $E10$ representan las acciones de las reglas.

En este modelo de CCPN no es difícil observar que el número de lugares no es tan grande como la suma del número de eventos y acciones. Esto es porque muchos eventos y acciones son la misma operación en la BD, es decir, el evento de una regla puede a la vez ser la acción de alguna otra regla.

5.5. Comentarios

El proceso de detección de eventos compuestos que más se asemeja al proceso que utiliza la CCPN es el detector de eventos de SAMOS [38] [39], en el cual se utiliza una variación de una CPN, denominada S-PetriNet, la cual se utiliza para la especificación de los constructores de eventos compuestos. Sin embargo, la S-PetriNet solamente se utiliza para la detección de los eventos

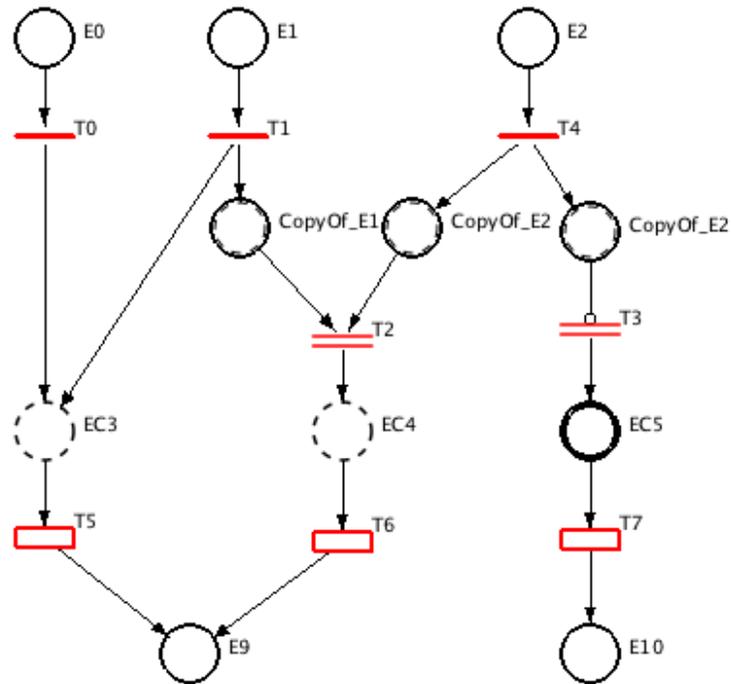


Figura 5.19: Modelo de la CCPN que contiene a los eventos compuestos *conjunción*, *disyunción* y *negación*.

compuestos, dejando el resto del procesamiento de la regla ECA al manejador de reglas de SAMOS. Otro trabajo donde se utiliza la teoría de PN en la detección de eventos compuestos es en CEDAR [73], donde se utilizan CPN's para la formación de estructuras en la implementación de eventos compuestos. Sin embargo, en estas propuestas solamente en la detección de eventos es donde se aplican PN's, y en el resto del modelo de conocimiento de la BD activa no se maneja la teoría de PN.

En éste trabajo se están proponiendo patrones para la definición de nueve elementos compuestos: *conjunción, disyunción, secuencia, simultáneo, negación, primero, último, historia, y alguno*. Estos patrones están basados en el mismo modelo de definición de reglas ECA: la CCPN. De esta manera se mantiene la misma estructura en la definición de eventos compuestos y en la definición de la regla ECA.

Para la generación de las estructuras, correspondientes a los patrones de eventos compuestos, dentro de una base de reglas ECA, se desarrollaron algoritmos para producir la CCPN para cada uno de los eventos compuestos participantes.

La ejemplificación de los eventos compuestos propuestos en la literatura se realiza mediante el uso de variables, siendo escasos los documentos donde consideran ejemplos prácticos del uso de eventos compuestos.

Capítulo 6

Análisis de Terminación y Confluencia

Para asegurar el buen funcionamiento de la base de reglas ECA, antes de su implementación en un SBDA, éstas deben analizarse para determinar si esta base de reglas no conducirá a un estado inconsistente de la BD.

Las relaciones que se dan entre diferentes reglas ECA forman secuencias de reglas, produciendo resultados inesperados durante el procesamiento de las reglas. Por lo tanto, es importante que el desarrollador de reglas conozca, con anticipación, el comportamiento que mantendrá la base de reglas ECA desarrollada durante tiempo de ejecución, y evitar posibles inconsistencias del SBDA durante el procesamiento y ejecución de reglas.

El comportamiento de la base de reglas puede conocerse a partir de un análisis del comportamiento de las reglas ECA, el cual puede realizarse en tiempo de compilación (análisis estático) y en tiempo de ejecución (análisis dinámico).

El *análisis estático* se realiza durante el proceso de desarrollo de la base de reglas ECA, antes de ser implementadas sobre un SBDA. Este análisis le da al desarrollador de reglas la pauta para determinar si las reglas desarrolladas no producirán un estado inconsistente del SBD durante su ejecución. En caso de que el disparo de reglas ECA provoque un estado inconsistente del SBD, entonces el desarrollador de reglas decide si modifica la base de reglas, con el fin de evitar el estado inconsistente.

Por otro lado, el *análisis dinámico* es aquel que se realiza ya cuando la base de reglas están ejecutándose en un SBDA. En este tipo de análisis, el SBDA monitorea el procesamiento en el disparo de las reglas ECA, para detectar si la acción de alguna de ellas producirá un estado inconsistente.

6.1. Problemas de análisis estático

El estado inconsistente, dentro de un SBDA, puede producirse, principalmente, a partir de dos problemas conocidos: el problema de *no terminación* y el de *no confluencia*.

6.1.1. No terminación

El problema de *no terminación* es cuando no se garantiza la propiedad de *terminación*, definida en el capítulo 2, en la cual se especifica que una base de reglas tiene la propiedad de terminación si el disparo en cadena, de una secuencia de reglas, siempre termina. Por lo tanto, el problema de *no terminación* es cuando el disparo de reglas se hace de manera infinita si el disparo de reglas cae en un ciclo, en la relación entre reglas, y no termina de dispararse.

Durante el desarrollo de una base de reglas ECA, es muy importante asegurar si el disparo de las reglas no producirá el disparo infinito de las mismas, garantizando así la propiedad de terminación. Los análisis que se han desarrollado para determinar la propiedad de terminación en una base de reglas ECA, están basados en análisis estático y análisis dinámico.

6.1.2. Confluencia

El problema de *no confluencia* es cuando no se tiene garantizado que el procesamiento de un conjunto de reglas produzca siempre el mismo resultado después del disparo de las reglas, aún cuando el orden de las reglas disparadas sea diferente; es decir, si cada vez que sea procesado un mismo conjunto de reglas en un orden diferente, el estado final de la BD siempre es el mismo, entonces se dice que el SBDA es confluente.

Es importante también detectar este problema cuando se hace el desarrollo de una base de reglas ECA, para proponer soluciones alternativas a las reglas que provocan este conflicto.

6.2. Estado del arte

Se han desarrollado diferentes enfoques para realizar el análisis de bases de reglas ECA. En el artículo [41], Alexander Aiken, Joseph M. Hellerstein y Jennifer Widom presentan métodos para realizar el análisis estático de conjuntos de reglas activas y determinar si las reglas terminan, producen un estado final único, así como un flujo único de acciones. Los autores presentan sus métodos de análisis en el contexto del sistema de reglas *Starburst*.

En [42] se presenta un análisis de bases de reglas activas basado en un algoritmo de "propagación", el cual utiliza un álgebra relacional extendida para determinar cuando la acción de una regla afecta a la condición de otra y además, determinar cuando las acciones de las reglas llegan a conmutar en un mismo punto.

En [43], se discute el problema de terminación en un conjunto de reglas, donde se analizan las *actualizaciones con conflictos*—determinando cuando una regla puede deshacer los cambios realizados previamente por otra regla. Aunque se proponen modelos y una "arquitectura para resolver problemas", no se proporcionan los algoritmos respectivos. Posteriormente, el mismo autor, presenta una técnica de análisis de reglas basada en los resultados obtenidos de sistemas de reescritura [44]. Esta técnica se aplica a un modelo de regla con restricciones y no puede extenderse fácilmente a sistemas de BDA.

El análisis de terminación también se ha explorado desde el ámbito de las bases de datos deductivas [45], [46], desafortunadamente el modelo de procesamiento de reglas es muy diferente del modelo para BDA.

Fraternali, Paraboschi y Tanca en [47], construyen una "Hipergráfica de disparo". En esta gráfica, los nodos son restricciones y los arcos son reglas, que viajan desde una restricción hacia un conjunto de restricciones. Los ciclos en la gráfica se interpretan como secuencias infinitas de reglas. Debido a que la hipergráfica es muy compleja, una herramienta automática convierte una hipergráfica de disparo, que contiene ciclos, a una hipergráfica acíclica dirigida. Las reglas que son parte de los ciclos se eliminan de la gráfica, pero no se eliminan en el sistema, solamente se marcan para un monitoreo especial en tiempo de ejecución y prevenir ciclos infinitos.

El artículo [48] describe la implementación del método de la Gráfica de Disparo Refinado (RTG) para el análisis de terminación de reglas activas. El método RTG está definido dentro del lenguaje de base de datos orientadas a objetos activa-deductivas conocido como CDOL (Comprehensive,

Declarative, Object Language). El método RTG estudia el contenido de pares de reglas y ciclos de reglas en una gráfica de disparo y evalúa la unificación exitosa de la acción de una regla con el evento de otra, además de que si se satisfacen las condiciones de la regla. Si el análisis demuestra que una regla no puede disparar a otra, el arco que conecta a estas dos reglas en la gráfica de disparo se elimina, hasta obtener una gráfica de disparo refinada. Sin embargo, no existe una definición formal de éste método, solamente se presentan los resultados sobre un sistema implementado.

En el estado del arte existen trabajos donde se intenta incorporar comportamiento activo a las bases de datos por medio de la teoría de redes de Petri. La mayoría de estas investigaciones aprovechan las propiedades básicas que ofrecen las redes de Petri, al igual que la simulación gráfica de las reglas activas. A continuación se presenta una descripción de éstos trabajos.

Se propuso un modelo de abstracción múltiple denominado Red de Petri con Parámetros (Parameterized Petri Net, PPN). Este modelo es utilizado como una herramienta para modelar y analizar el comportamiento activo de una base de datos. Se presentan procedimientos para indicar si una base de datos activa presenta el problema de No-terminación. Para detectar un ciclo en una PPN, se obtiene su árbol de alcanzabilidad (reachability tree), método de análisis en la teoría de redes de Petri. Aunque con PPN puede probarse la ausencia de ciclos en una base de reglas, no puede probarse su presencia; si un ciclo es detectado en la PPN, no implica que existe un ciclo en el disparo de las reglas. [49]

Otro trabajo interesante es un método, basado en redes de Petri, para analizar las reglas de bases de datos activas y determinar si las reglas presentan ciclos, si son inconsistentes y si entran en contradicciones. Los autores proponen una Red de Petri con Restricciones (Constraint Petri Net, CPN), la cual es una sistema de redes Predicado/Transición y almacena conjuntos de restricciones en cada transición. Sin embargo, éste método es estático. El algoritmo para detectar contradicciones en un conjunto de reglas solo realiza la búsqueda en un nivel [50], es decir, solamente se basan en la detección de ciclos en la CPN para determinar que puede existir la presencia de un disparo infinito de reglas ECA.

La investigación descrita en [51] presenta la modelación y simulación de reglas en el sistema de base de datos activa ALFRED (Active Layer For Rule Execution in Database Systems), donde las reglas y los comandos de usuario se representan como redes de Petri Coloreadas llamadas "Redes de Petri para el Flujo de Reglas Activas". ALFRED es un proyecto del Instituto de Sistemas de

Información de la Universidad de Bern.

Se desarrolló un análisis estático de terminación de reglas en [52], basado en el metamodelo *Vampire*. El metamodelo Vampire utiliza redes de Petri como un método formal para una especificación, precisa y exhaustiva, de muchos de los aspectos semánticos elementales en los modelos de reglas. Su método de análisis de terminación de reglas se basa en el "coverability tree", el cual es una gráfica donde se detecta la presencia de un ciclo infinito en la PN. Sin embargo, al igual que en [49], la presencia de un ciclo en una PN no implica el disparo infinito de reglas en la base de datos activa. Posteriormente proponen un método, para el análisis de terminación de disparo de reglas, también basado en redes de Petri [53]. Este método parte del supuesto de que cada regla, individualmente, está correcta y se deriva automáticamente su correspondiente PN. El algoritmo presentado en este artículo es una extensión del *método Trac*. El método Trac está basado en el análisis de dos gráficas dirigidas: la gráfica de disparo (Triggering Graph, TG) y la gráfica de activación (Activation Graph, AG). En ambos tipos de gráficas, las reglas se almacenan como nodos. La desventaja principal de este método es la complejidad, que depende de la cantidad e interdependencia de las reglas.

Finalmente, en [54], Detlef Zimmer, Rainer Unland y Axel Meckenstock, proponen a las redes de Petri como la base para un análisis de reglas en tiempo de compilación. Argumentan que el análisis basado en redes de Petri es un enfoque que puede utilizarse para una especificación detallada del comportamiento de las reglas.

En este trabajo de tesis doctoral se utiliza la matriz de incidencia de la teoría de PN, como herramienta auxiliar en la detección del problema de No terminación.

6.3. Matriz de Incidencia de CCPN

El comportamiento dinámico de muchos sistemas estudiados en ingeniería pueden describirse mediante ecuaciones diferenciales o ecuaciones algebraicas. El comportamiento dinámico de las PN también se describen por medio de ecuaciones. Las ecuaciones de estado utilizadas en teoría de PNs toman a la *matriz de incidencia* como una estructura de representación[32], en la cual se almacena información relevante sobre la relación existente entre los lugares (representados por las columnas de la matriz, y las transiciones (representados por los renglones), además de que denota las reglas de activación y disparo de las transiciones. La definición formal de la matriz de incidencia es la siguiente.

Definición 6.1 Para una red de Petri N , con n transiciones y m lugares, la matriz de incidencia $A = [a_{ij}]$ es una matriz de números enteros de $n \times m$. El valor para cada elemento de la matriz está dado por:

$$a_{ij} = a_{ij}^+ - a_{ij}^-$$

donde $a_{ij}^+ = w(i, j)$ es el peso del arco que conecta una transición $t_i \in T$ con su lugar de salida $p_j \in P$ y $a_{ij}^- = w(j, i)$ es el peso del arco que conecta una transición $t_i \in T$ con su lugar de entrada $p_j \in P$.

En la matriz de incidencia se almacena suficiente información sobre la estructura completa de una PN, y es posible reconstruir la PN original a partir de los valores almacenados en la matriz de incidencia.

En la matriz de incidencia que se genera para una CCPN, se observa que en las conexiones que existen entre elementos que forman a una regla ECA, los arcos solamente tomarán valores de $-w(p, t)$ y 1 para los lugares de entrada y salida, respectivamente. En el caso de las transiciones $t \in T_{copy}$ los arcos, tanto de entrada como de salida, que están conectados con t tienen como valor del peso del arco $w(p, t) = w(t, p) = 1$. Pero, en la representación de eventos compuestos como *historia* y *alguno*, se utilizan arcos de entrada a la transición $t \in T_{comp}$ con valor para $w(p, t) \ll 1$, como se muestra en el capítulo de descripción de Eventos Compuestos. En el caso del evento compuesto *negación*, se utiliza un arco inhibitorio para formar su estructura de CCPN correspondiente, para efectos de representación en la matriz de incidencia se representa como un arco normal, pero para efecto de análisis de terminación, en tiempo de ejecución se verifican si se trata de un arco normal o un arco inhibitorio.

El valor a_{ij} de A , que representa la relación entre una transición $t_i \in T$ y su lugar de entrada $p_j \in P$ es -1 cuando $t_i \in T_{regla} \cup T_{copy}$, porque $a_{ij}^+ = 0$ y $a_{ij}^- = 1$, por lo tanto $a_{ij} = 0 - 1 = -1$. De manera similar, el valor a_{ij} de A que representa la relación entre una transición $t_i \in T$ y su lugar de salida $p_j \in P$ es 1 cuando $t_i \in T$, porque $a_{ij}^+ = 1$ y $a_{ij}^- = 0$, por lo tanto $a_{ij} = 1 - 0 = 1$.

Sin embargo, cuando $t_i \in T_{comp}$, en el caso de que $Type(t_i) = Historia$ ó $Type(t_i) = alguno$, entonces el peso del arco de entrada a t_i toma como valor el especificado en el evento compuesto, por lo que en este caso el valor a_{ij} de A será $-w(p_i, t_j)$.

El valor de a_{ij} es 0 cuando t_i y p_j no tienen relación alguna, es decir, que p_j no es ni lugar de entrada ni de salida para t_i . Lo anterior puede resumirse de la siguiente manera:

$$a_{ij} = \begin{cases} -1 & \text{El lugar } p_j \in P \text{ es un lugar de entrada a la transición} \\ & t_i \in T_{regla}UT_{copy}. \\ -w(p_j, t_i) & \text{El lugar } p_j \in P \text{ es un lugar de entrada} \\ & \text{a la transición } t_i \in T_{comp}. \\ 0 & \text{No existe un arco que conecta al lugar } p_j \in P \\ & \text{con la transición } t_i \in T \text{ y viceversa.} \\ 1 & \text{El lugar } p_j \in P \text{ es un lugar de salida de la} \\ & \text{transición } t_i \in T. \end{cases}$$

En cada una de las columnas de A aparecerá, a lo más, un valor negativo $-n$, donde $n = w(p_j, t_i)$, $a_{ij} = -w(p_j, t_i)$. Si el lugar $p_j \in P$ es un lugar de entrada, entonces en su respectiva columna j de A aparecerá un valor de $-n$. En cambio, si el lugar p_j solamente es lugar de salida, entonces no existirán valores negativos en la columna j de la matriz de incidencia A .

Si en A todas las columnas tienen valores positivos y uno negativo, entonces todos los lugares son lugares de entrada y salida a la vez.

En base a las propiedades particulares de la matriz de incidencia para una CCPN, se presentan las siguientes definiciones.

Definición 6.2 *Un lugar p_j es un nodo inicial NI si la columna j de la matriz de incidencia A presenta solamente valores 0's y un solo valor de $-n$, donde $n = w(p_j, t_i)$.*

Definición 6.3 *Un nodo terminal NT es un lugar p_j que representa la acción, mas no el evento, de una o varias reglas. En la matriz de incidencia, este lugar se caracteriza por almacenar solamente valores 0's y 1's en la columna correspondiente. La ausencia de valores negativos $-n$ es debido a que el lugar no representa el lugar de entrada para alguna transición.*

En la figura 6.1 se muestra una CCPN en la que el lugar p_0 es un nodo inicial, porque no es el lugar de salida de alguna transición. Además, ésta CCPN tiene dos lugares que son nodos finales, los lugares p_1 y p_4 , los cuales son solamente lugares de salida de las transiciones t_0 y t_1 , respectivamente.

En la figura 6.2 se muestra la matriz de incidencia para esta CCPN, donde puede observarse que la columna que representa al lugar p_0 solamente contiene un valor negativo de -1 y el resto de

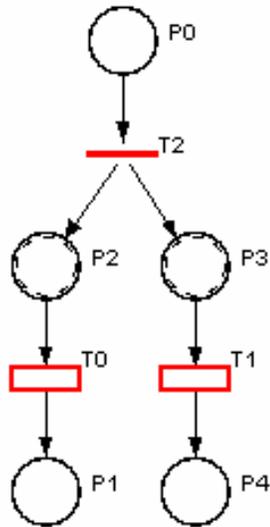


Figura 6.1: CCPN que contiene un nodo inicial (p_0) y dos nodos terminales (p_1 y p_4).

la columna tiene valores 0 's. Por otro lado, los lugares p_1 y p_4 , que son nodos terminales, tienen en sus respectivas columnas solamente valores 0 's y enteros positivos (1 's).

6.4. Análisis de Terminación y Confluencia con CCPN

Durante el desarrollo de una base de reglas ECA, llegan a existir relaciones entre las reglas que conforman a la base de reglas. Estas relaciones provocan que se susciten problemas en tiempo de ejecución durante el procesamiento de las reglas.

El desarrollo de bases de reglas ECA se vuelve una actividad complicada cuando el número de reglas se incrementa. Los problemas con que se encuentran los desarrolladores de reglas son conocidos como el problema de *no terminación* y el problema de *no confluencia*. En el problema de terminación, la interrelación existente entre las reglas puede originar un disparo en cadena de la base de reglas y si en algún momento esta cadena regresa a la que comenzó con el disparo de reglas, entonces se provoca un disparo infinito que no termina de dispararse. Por otro lado, una base de reglas ECA es confluente siempre y cuando el disparo de un conjunto de reglas siempre obtenga un mismo estado final de la BD, sin importar el orden en que estas reglas sean disparadas .

		P L A C E S				
		0	1	2	3	4
T R A N S I T I O N S	0	0	1	-1	0	0
	1	0	0	0	-1	1
	2	-1	0	1	1	0

Figura 6.2: Matriz de incidencia para la CCPN de la figura 6.1.

El análisis de una base de reglas ECA modelada con la CCPN puede realizarse utilizando la matriz de incidencia, definida anteriormente, a partir de la cual pueden encontrarse las secuencias o rutas de procesamiento de reglas, útiles para realizar el análisis de *terminación y confluencia*.

Una ruta en una CCPN se da por la interrelación existente entre los componentes de dos o más reglas ECA. Por lo tanto, tomando en consideración el uso de la matriz de incidencia de la CCPN, una ruta la podemos definir de la siguiente manera.

6.4.1. Rutas

Definición 6.4 Una ruta R es una secuencia de pares ordenados (i, j) , obtenidos a partir de la matriz de incidencia de la CCPN, los cuales están colocados de la siguiente manera:

$$(a, b), (a, c), (d, c), \dots, (w, x), (y, x), (y, z)$$

Los valores de i y j son los índices en la matriz de incidencia. La secuencia de pares (i, j) es el movimiento sobre la matriz de incidencia A que describe la conexión entre lugares y transiciones.

El orden que siguen estas parejas de valores se rigen por el criterio de que el valor para la pareja (i, j) , donde $p_j \in P$ es el lugar de entrada de la transición $t_i \in T$, es un valor negativo correspondiente al valor negativo del peso del arco $w(p_j, t_i)$; y el valor para la pareja (i, j) , donde $p_j \in P$ es el lugar de salida de la transición $t_i \in T$, es 1; entonces la primer pareja de valores (i, j) deben ser las coordenadas para un valor de $-w(p_j, t_i)$, dado que primero buscamos un lugar de entrada para iniciar una ruta. La siguiente pareja de valores debe ser una coordenada del mismo renglón pero diferente columna y con un valor de $+1$, para buscar un lugar de salida de la misma transición, y así sucesivamente.

En la definición, $(a, b), (a, c), (d, c), \dots, (w, x), (y, x), (y, z)$, el primer par ordenado está compuesto por la transición $t_a \in T$ y su lugar de entrada $p_b \in P$, continuando con la ruta, ahora formamos el segundo par ordenado con la misma transición t_a y su lugar de salida $p_c \in P$, siguiendo con estos pasos se forman las rutas de la CCPN.

El orden de los valores de cada par ordenado debe ser $-w_1, 1, -w_2, \dots, -w_3, 1, -w_4$, como se muestra en la siguiente tabla:

$(a,b),$	$(a,c),$	$(d,c),$	$\dots,$	$(w,x),$	$(y,x),$	(y,z)
$-w_1$	1	$-w_2$	\dots	$-w_3$	1	$-w_4$

El algoritmo para encontrar las parejas ordenadas que forman una ruta es el siguiente:

```
siguienteNodo(int valor, int indice) {
    if(valor < 0) {
        boolean flag ← false;
        for(int i=0; i<n; i++) {
            if(A[i][indice] < 0) {
                flag ← true;
                if(noExisteNodo(i,indice)) {
                    agregaNodo(new Coordenada(i,indice));
                    siguienteNodo(1,i);
                    eliminaUltimoNodo();
                }
            }
            else {
                agregaNodo(new Coordenada(i,indice));
                print("Ruta cíclica: ");
            }
        }
    }
}
```

```

    imprimeRuta();
    eliminaUltimoNodo();
  }
}
}
if(!flag) {
  imprimeRuta();
}
else {
  for(int j=0; j<m; j++) {
    if(A[indice][j] > 0) {
      agregaNodo(new Coordinada(indice,j));
      siguienteNodo(-1,j);
      eliminaUltimoNodo();
    }
  }
}
}
}

```

Definición 6.5 Una ruta cíclica RC es una ruta R donde el último par ordenado (y, z) ya se encuentra listado en R , es decir, $y = q_k$ y $z = r_k$, $k \in \{1, 2, \dots, |R| - 1\}$.

Definición 6.6 Una ruta acíclica RA es aquella donde la última pareja ordenada (i, j) de la ruta es diferente de sus antecesoras y le corresponde un valor de 1 en su matriz de incidencia respectiva, es decir, que RA tiene un nodo terminal.

De acuerdo a [32], una *PN pura* es aquella PN que no tiene ciclos. Como la CCPN es una extensión de la PN, entonces una CCPN que no tiene ciclos la denominaremos CCPN pura.

6.4.2. Análisis de Terminación

Dadas las definiciones anteriores, podemos presentar los siguientes teoremas para el análisis de terminación:

Teorema 6.1 *Si todas las rutas R de una CCPN C son acíclicas, entonces C es una CCPN pura, por lo tanto el disparo de las reglas termina.*

Demostración. Dado que cada ruta R_i en C es acíclica entonces, por definición de una ruta acíclica, existen tanto nodos iniciales (r_{inic}, c_{inic}) como nodos terminales (r_{term}, c_{term}) para cada ruta R_i . Por lo tanto, si el disparo de reglas comienza en algún nodo (r, c) de R_i y, en el peor de los casos, todas las condiciones de las transiciones $t \in T_{regla}$ son evaluadas a verdadero, el disparo de reglas termina cuando se llega al nodo terminal (r_{term}, c_{term}) cuando se ejecuta la acción representada por el lugar $p_{c_{term}}$. ■

Sin embargo, si existe al menos una ruta cíclica RC en la CCPN, el disparo de reglas puede caer en un ciclo infinito de disparo de reglas. Pero, la existencia de una RC dentro de la CCPN es condición necesaria pero no suficiente para que el disparo de reglas no termine, existen otros factores que deben tomarse en cuenta para asegurar si el disparo infinito de reglas no ocurrirá.

Por ejemplo, en el modelo de la CCPN se consideran los eventos compuestos que están presentes en el desarrollo de reglas ECA, los cuales favorecen en gran medida el análisis de terminación de reglas, mientras que en [42] argumentan que si en el enfoque que ellos proponen se consideran a los eventos compuestos, esto aumentaría la complejidad de su propuesta, por lo que solo manejan eventos primitivos.

Los eventos compuestos se clasifican de acuerdo a la influencia que tengan sobre el análisis de terminación. Por un lado se tiene al evento compuesto *negación*, considerado como un evento que evita un disparo infinito de reglas cuando éste se encuentra dentro de una ruta cíclica RC , debido a su estructura en $CCPN$, la cual contiene a un arco inhibidor.

Teorema 6.2 *Si una ruta cíclica RC contiene un arco inhibidor, debido a que el evento compuesto negación está incluido dentro de RC , entonces RC termina el disparo de reglas en el punto donde se encuentra el arco inhibidor.*

Demostración. Por definición del evento compuesto *negación* y del propio arco inhibidor, éste dispara solamente si el lugar de entrada p_i no contiene ningún token al final del intervalo de tiempo D . Por lo tanto, si un token es depositado en un lugar p_j , el cual está incluido dentro de una ruta cíclica RC , a la cual también pertenece p_i , entonces, por efecto del disparo de las reglas

y el desplazamiento de tokens sobre RC , se depositará un token en p_i , evitando con esto el paso posterior de tokens sobre el arco inhibitor, y en consecuencia, el disparo de reglas se termina. ■

Por otro lado, los eventos compuestos conjunción, secuencia y simultáneo, están formados por dos o más eventos constituyentes, lo que significa que para poder ser creados necesitan de la presencia de todos estos eventos constituyentes. Tomando esto en consideración, es posible definir el siguiente teorema.

Teorema 6.3 *Para los eventos compuestos conjunción, secuencia y simultáneo, si alguno de éstos pertenece a una ruta cíclica RC y al menos uno de los eventos que los conforman no se genera por la acción de una de las reglas que están presentes dentro RC , entonces el disparo de reglas dentro de RC termina.*

Demostración. Los eventos que conforman a un evento compuesto, son representados por lugares de entrada $p_i \in P$ a transiciones $t_j \in T_{comp}$, y si t_j representa a alguno de los eventos compuestos *conjunción, secuencia o simultáneo*, entonces t_i necesita la ocurrencia de todos sus lugares de entrada $\bullet t_i$ para poder ser disparada y formar al evento compuesto que está representando. Sin embargo, si $t_i \in \{t \mid (p, t) \in RC\}$ y alguno de sus eventos constituyentes $p_i \in \{p \mid p \in \bullet t_i, (p, t_i) \notin RC\}$ entonces el disparo de reglas termina. Supongamos que $p_1 \in \{p \mid p \in \bullet t_i, (p, t_i) \in RC\}$ y $p_2 \in \{p \mid p \in \bullet t_i, (p, t_i) \notin RC\}$, por lo tanto, cuando un token es depositado en p_1 durante el procesamiento de reglas en RC , t_i necesita esperar a que sea depositado en p_2 el token correspondiente al evento que está representando. ■

El evento compuesto *alguno* necesita un tratamiento distinto porque, de acuerdo con su definición, no es necesario que todos sus eventos constituyentes ocurran, pero sí un número determinado de eventos.

Teorema 6.4 *Si el evento compuesto $alguno(m, e_1, e_2, \dots, e_n)$ forma parte de una ruta cíclica RC , y si k de sus eventos que lo forman no son generados por la acción de una regla representada por una transición t_i que se encuentra dentro de RC y $n - k < m$, entonces el disparo de reglas que están en RC termina.*

Demostración. Por definición, el evento compuesto $alguno(m, e_1, e_2, \dots, e_n)$ necesita al menos de la ocurrencia de m eventos de un total de n posibles eventos para que éste ocurra; por lo tanto,

la estructura en CCPN para el evento compuesto *alguno* necesita de n lugares de entrada p_i , donde $i = 1, 2, \dots, n$, y de un lugar $p_j \in P_{virtual}$ utilizado para almacenar las ocurrencias de cada evento constituyente. Supongamos que $p_j \in \{p \mid (p, t) \in RC\}$ y que k lugares de entrada p_i no pertenecen al conjunto $\{p \mid (p, t) \in RC\}$ y $n - k < m$, es decir, que el número total de eventos posibles n para formar al evento compuesto *alguno*, menos un número k es menor que el total de eventos generados por la acción de una regla dentro de RC , entonces el disparo de reglas termina, porque el evento compuesto *alguno* necesita esperar por más eventos generados fuera de RC y completar el total de m eventos requeridos. ■

Finalmente, los eventos compuestos con intervalos de tiempo *último*, *primero* e *historia* no son considerados, porque si alguno de éstos pertenece a una ruta cíclica RC y su intervalo de tiempo D no es muy corto o la periodicidad del intervalo no es alta, entonces el disparo infinito de reglas no termina, pero este disparo infinito no puede considerarse como un problema de no terminación, puesto que no conllevaría a un estado inestable del SBDA. Sin embargo, dependiendo del criterio del desarrollador de reglas ECA, las rutas cíclicas RC que contengan a éstos eventos compuestos, serán monitoreados en tiempo de ejecución.

El evento compuesto *disyunción* no afecta para nada el disparo infinito de reglas ECA, como el caso de los otros eventos compuestos. Debido a que los eventos que conforman al evento compuesto *disyunción* son considerados por separado, es decir, con uno de ellos que ocurra se considera la ocurrencia del evento compuesto, entonces, no necesita esperar por la ocurrencia de todos ellos. Si el evento compuesto *disyunción* forma parte de una RC , y al menos uno de los eventos constituyentes de la *disyunción* también forma parte de RC , es condición suficiente para que el evento compuesto ocurra y se continúe con el procesamiento de RC .

Por otro lado, si alguna ruta cíclica RC cubre con todos los teoremas descritos anteriormente, aún existe una posibilidad para determinar si el disparo infinito de reglas en RC termina.

Teorema 6.5 *Si la condición de una regla ECA almacenada en una transición $t_i \in \{t \mid t \in T_{regla}, (t, p) \in RC\}$ siempre se obtiene un valor de falso de acuerdo a la información contenida dentro de la regla que le precede, entonces el disparo de reglas termina.*

Demostración. En la CCPN se almacena información sobre la base de reglas ECA que está representando, por lo que puede deducirse el estado de la BD al que se llegará después de la ejecución de la acción de la regla que precede a t_i , y en algunos casos, es posible realizar la evaluación

de la condición almacenada en t_i contra la información de la acción de la regla que le precede, almacenada en $t_j \in T_{regla}$. Supongamos que una transición $t_i \in T_{regla}$ se encuentra dentro de una ruta cíclica RC , y la acción de la regla ECA que le precede dentro de RC , representada por $t_j \in T_{regla}$, genera un token $C(p_j)$, donde $p_j \in t_j^\bullet$ y $p_j \in \bullet t_i$. Si $C(p_j)$ contiene información suficiente para determinar que t_i no será disparada, si la evaluación de la condición que tiene almacenada resulta falsa $Type(Con(t_i)) = falso$, entonces se determina que el disparo de reglas termina. Este procedimiento se aplica a cada $t \in T_{regla}$ y que se encuentra dentro de una ruta cíclica. Si existe al menos una donde pueda determinarse que por la acción de la regla, la condición resulte falsa, entonces RC termina en la ejecución del disparo de reglas ECA. ■

Si en la estructura de CCPN, correspondiente a una base de reglas ECA, existen rutas cíclicas que no satisfacen los teoremas listados en los párrafos anteriores, éstas son monitoreadas en tiempo de ejecución. Estas rutas cíclicas RC presentan las siguientes características:

- Los nodos de las rutas cíclicas $(r, c) \in \{(a, b) \mid p \in P, p_b \in P_{prim}\}$, es decir, los lugares que se encuentran dentro de la ruta cíclica son lugares pimitivos $p_b \in P_{prim}$.
- Los nodos de las rutas cíclicas $(r, c) \in \{(a, b) \mid p \in P, p_b \text{ representa a uno de los eventos compuestos conjunción, secuencia o simultáneo}\}$ y cada uno de los eventos que conforman a éstos eventos compuestos son generados a partir de la acción de alguna regla que pertenece a la ruta cíclica RC , es decir, $t_c \in T_{comp}$, $t_c^\bullet = \{p_b\}$, $\forall p \in \bullet t[p \in t_i^\bullet, t_i \in RC]$.
- Las rutas cíclicas RC no contienen arcos inhibidores.
- El evento compuesto $any(m, e_1, e_2, \dots, e_n)$, al menos m eventos son generados por la acción de alguna regla que pertenece a RC .
- El intervalo de tiempo D en los eventos compuestos *último*, *primero* e *historia* es muy corto y su periodicidad es altamente frecuente.
- El valor de verdad en la evaluación de las $t \in T_{regla}$, $t \in RC$ depende del estado de la BD, el cual solo puede conocerse mediante una consulta a la BD.

En tiempo de ejecución, solamente las rutas cíclicas que presentan éstas propiedades son monitoreadas, y el estado de la BD es verificado si alguna de éstas rutas comienza con el disparo de las reglas que la conforman.

6.4.3. Ejemplo

Para ejemplificar la aplicación de la CCPN en el análisis estático de reglas, se tiene la siguiente base de reglas:

Regla 1: Cuando un empleado es agregado en la BD de una empresa y la producción del departamento al que pertenece el empleado es modificada, si el valor de la producción es mayor de \$900.00, entonces la prima del empleado se actualiza a \$100.00.

Regla 2: Cuando el salario o la prima de un empleado se modifica, si el salario se incrementa en mas de \$200.00 ó la prima se incrementa en más de \$50.00, entonces el rango del empleado también se incrementa.

Regla 3: Cuando el rango del empleado es modificado, si el rango es mayor de 15, entonces al presupuesto del departamento donde labora el empleado se le agregan \$1,000.00.

Regla 4: Cuando se modifica el presupuesto de un departamento, si el presupuesto es mayor de \$20,000.00, entonces la producción del departamento se incrementa en un 3 %.

El esquema para las tablas o relaciones de la BD donde se aplican éstas reglas es el siguiente:

Departamento(dep_id, producción, presupuesto)

Empleado(emp_id, dep_id, salario, prima, rango)

La CCPN obtenida para ésta base de reglas ECA se muestra en la figura 6.3.

Gráficamente podemos determinar que en esta base de regla se encuentra presente un ciclo, y posiblemente un problema de no terminación del disparo de reglas. A partir de esta CCPN obtenemos su matriz de incidencia para encontrar este ciclo. En la figura 6.4 se muestra la matriz de incidencia correspondiente a la CCPN del ejemplo. Aplicando el algoritmo de búsqueda de rutas, encontramos la ruta cíclica (que se encuentra delineada) que se había detectado visualmente en la gráfica de la CCPN.

Sin embargo, el hecho de que exista una ruta cíclica en la CCPN no es condición suficiente para que exista el problema de no terminación. En este sentido, se verifican los teoremas presentados para determinar si efectivamente esta ruta cíclica conduce a un estado inconsistente al SBDA.

La ruta cíclica encontrada por el algoritmo está representada como el conjunto de pares ordenados (3,2), (3,4), (2,4), (2,5), (4,5), (4,6), (5,6), (5,7), (6,7), (6,1), (0,1), (0,2), (3,2), donde el primer nodo es el mismo que el último. Además, de acuerdo al teorema 6.3, debido a la presencia del evento compuesto *conjunción* no se da el disparo infinito de reglas porque éste evento compuesto tiene

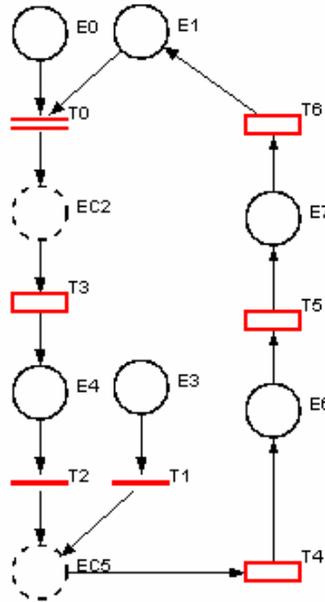


Figura 6.3: CCPN obtenida a partir de una base de cuatro reglas ECA.

		P L A C E S								
		0	1	2	3	4	5	6	7	
T R A N S I T I O N S	0	-1	-1	1	0	0	0	0	0	0
	1	0	0	0	-1	0	1	0	0	0
	2	0	0	0	0	-1	1	0	0	0
	3	0	0	-1	0	1	0	0	0	0
	4	0	0	0	0	0	-1	1	0	0
	5	0	0	0	0	0	0	-1	1	0
	6	0	1	0	0	0	0	0	0	-1

Figura 6.4: Matriz de incidencia obtenida de la CCPN de la figura 6.3.

que esperar a la ocurrencia del evento representado por E_3 , el cual ocurre fuera de la ruta cíclica, con lo que podemos determinar que en ésta base de reglas no existe el problema de no terminación.

6.4.4. Análisis de Confluencia

Como se menciona al inicio de este capítulo, el análisis de confluencia debe verificar si el disparo de un conjunto de reglas ECA es confluyente, es decir, si a partir de un conjunto de reglas activadas, independientemente del orden en que éstas sean disparadas, el estado de la final de la BD sea siempre el mismo.

Si el estado de la BD difiere entre distintas secuencias de disparo para el mismo conjunto de reglas activadas, entonces se dice que la base de reglas no es confluyente.

Para determinar el problema de confluencia también se utiliza la matriz de incidencia, a partir de la cual encontramos las rutas de ejecución que se presentan.

Si existe un lugar $p \in P$ que sea lugar de entrada para una transición $t \in T_{copy}$, entonces este lugar estará participando en diferentes rutas de ejecución de reglas. Si en alguna de estas rutas de ejecución existen al menos dos de ellas que pasen por el mismo lugar $p_j \in P$, $p_j \neq p$, entonces el disparo de reglas que pasó por el lugar p confluye nuevamente en el lugar p_j . Con lo que se puede concluir que en la base de reglas ECA existen dos rutas que confluyen.

Sin embargo, esto no es condición suficiente para decir que una base de reglas presenta el problema de no confluencia. Puede darse el caso que dos reglas, o más, sean disparadas al mismo tiempo y no pertenecer a la misma ruta. Si no existe una relación entre estas reglas entonces puede decirse que la base de reglas es confluyente. Sin embargo, si hay una relación donde lo que ocurra con una de ellas afecte a la otra, entonces si es importante el orden en que sean disparadas, y por la tanto existirá el problema de confluencia en esta base de reglas ECA.

6.5. Comentarios

La CCPN puede ser utilizada para la definición de reglas ECA, y en la misma representación de las reglas activas, el análisis de las reglas puede ser llevado a cabo. No es necesario realizar representaciones fuera del modelo de CCPN. La capacidad de la CCPN para soportar eventos

compuestos facilita la labor de la detección del problema de no terminación al considerar a todos los elementos constituyentes de un evento compuesto dentro de una ruta cíclica.

Los SBDA que manejan “triggers” evitan el disparo infinito de reglas al establecer como una restricción que la acción de un “trigger” no dispare a otro “trigger”; sin embargo, esta restricción limita la potencia que una regla ECA puede ofrecer.

Existen trabajos que abordan los problemas de no terminación y no confluencia, sin embargo, el análisis que proponen se lleva a cabo en un contexto separado a la definición de la base de reglas activas. [42] [43] [45] [46]

Por otro lado, existen sistemas que implementan el análisis estático, mediante métodos que verifican las reglas para determinar si existen los problemas de no terminación y no confluencia. [41] [48]

Además, dentro de la literatura hay trabajos donde se proponen métodos que utilizan la teoría de PN en la detección de éstos problemas presentes en el desarrollo de reglas activas; sin embargo, éstos métodos solamente encuentran ciclos en las relaciones existentes entre las reglas, y la presencia de un ciclo no es condición suficiente para asegurar el disparo infinito de reglas. [52] [53] [70] [50] [49]

En este trabajo, además de verificar la presencia de un ciclo, se determina si un ciclo no cae en un disparo infinito de reglas. A diferencia de los trabajos presentados, donde se utilizan PN's en la detección de ciclos, en este caso se utiliza la matriz de incidencia de la CCPN para encontrar todas las rutas en el disparo de transiciones, y en consecuencia las rutas cíclicas que existan en la base de reglas denotada como una CCPN.

El conjunto de rutas cíclicas encontradas en la CCPN es analizado, y si existen algunas rutas que pueda determinarse que no se dispararán infinitamente (como el caso de la negación), se eliminan del conjunto de rutas cíclicas.

Cuando no puede determinarse si el disparo de las reglas que forman un ciclo termina, entonces las rutas que permanecen en el conjunto de rutas cíclicas deben ser monitoreadas en tiempo de ejecución. El monitoreo de reglas lo lleva a cabo ECAPNSim, interfaz gráfica desarrollada como parte de éste trabajo de investigación y que toma los conceptos de la CCPN.

Capítulo 7

ECAPNSIM

El desarrollo de bases de reglas activas es una actividad que necesita realizarse con mucho cuidado. Actualmente existen muy pocos sistemas, como [51], que realizan la depuración y análisis de la base de reglas. La mayoría de los sistemas comerciales de BDA ([10], [14], [16], entre otros) ofrecen una sintaxis para la definición de reglas ECA, pero no es posible llevar a cabo un análisis de éstas para detectar problemas de inconsistencia.

ECAPNSim ofrece un medio gráfico y visual para representar bases de reglas ECA, auxiliándose de la CCPN. Como cualquier editor de PN, es posible realizar una simulación del comportamiento del sistema, en este caso, la simulación de la base de reglas ECA. Durante la ejecución de la simulación pueden observarse problemas de BDA, como la no-terminación y la confluencia, y por lo tanto, el desarrollador de la base de reglas puede modificar la base de reglas y pensar en otra alternativa de solución para evitar éstos problemas que conducen a estados inconsistentes del SBD.

A diferencia de otros sistemas que soportan la definición de reglas ECA, ECAPNSim ofrece dos modalidades de uso. En la modalidad "Simulación", el usuario ejecuta la simulación del comportamiento de la base de reglas. En la modalidad "Real", la base de reglas analizadas previamente, dará comportamiento activo a una BD pasiva, utilizando la misma CCPN con que se ejecutó la simulación.

7.1. Ambiente de desarrollo

La interfaz gráfica ECAPNSim se desarrolló con el lenguaje de Programación Orientado a Objetos (POO) Java. Se optó por un lenguaje orientado a objetos dado que cada elemento de la CCPN puede manejarse como objeto. Los lugares, transiciones y arcos poseen funciones y procedimientos implementados como métodos para cada elemento. Los tokens también pueden definirse como otra clase, debido a que un token, dentro de una CCPN, posee información e interactúa con los lugares y transiciones para modelar las reglas ECA. En particular, se utilizó Java por la portabilidad que ofrece para ejecutar los programas en diferentes plataformas. ECAPNSim proporciona la facilidad necesaria para el diseño y edición de PN. Además, de mecanismos de desplazamiento y de acercamiento sobre la estructura CCPN obtenida, dado que las bases de reglas activas podrían generar estructuras CCPN demasiado grandes. La velocidad de la simulación es controlable. La interfaz gráfica ofrece barras de herramientas e íconos de edición, simulación y operaciones sobre archivos.

ECAPNSim cuenta con un procedimiento para generar automáticamente una estructura CCPN a partir de la definición, en un archivo de texto, de una base de reglas activas. Las reglas leídas se convierten en objetos y se almacenan en un arreglo, el cual es el dato de entrada al procedimiento de conversión a la CCPN.

El desarrollo de ECAPNSim se hizo, originalmente, bajo la plataforma MAC OS X Server y se tomó como BD pasiva a Postgres, sin embargo, aprovechando la portabilidad de Java para diferentes plataformas, ECAPNSim puede ser ejecutado en diferentes sistemas operativos. Además de Postgres, se han hecho conexiones con sistemas gestores de bases de datos como MS Access, Oracle y Visual FoxPro.

7.2. Arquitectura de ECAPNSim

ECAPNSim es un sistema de BDA multi-capas, el detector de eventos se encuentra entre la BD y el usuario. Cuando se detecta un evento, éste se envía a la base de reglas ECA para que active aquellas que lo tengan como evento activador. ECAPNSim funciona de manera similar a estos sistemas multi-capas, existe una consola de usuario que se conecta con ECAPNSim, a través de sockets, la cual envía la instrucción SQL que se ejecutará en la BD. Dentro de ECAPNSim el detector de eventos recibe el mensaje enviado desde la consola hacia la BD, lo clasifica según

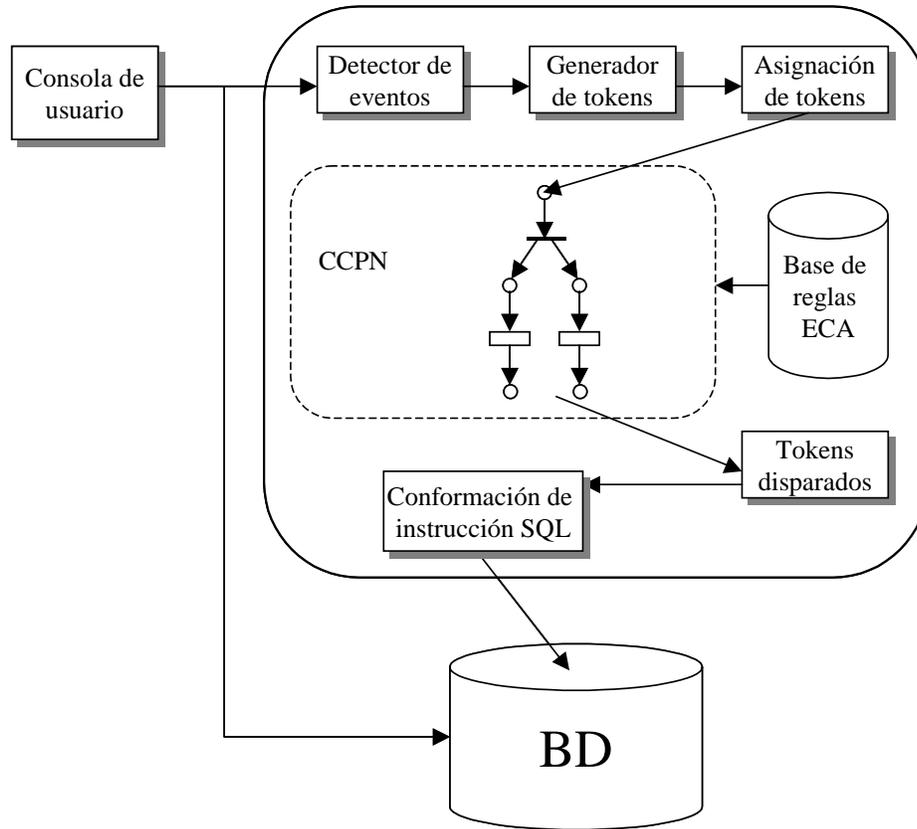


Figura 7.1: Arquitectura del ECAPNSim.

el comando SQL (insert, delete, update ó select) y lo envía al generador de tokens. El generador de tokens crea estructuras de tokens de acuerdo al comando de SQL y les agrega la información contenida en la instrucción. La asignación de tokens se refiere a la colocación de tokens en los lugares de la CCPN correspondiente. La CCPN siempre se encuentra lista para recibir tokens. Cuando un token es recibido, ECAPNSim verifica si la transición correspondiente se dispara y, en caso afirmativo, genera el token que se coloca en el lugar de salida. Los tokens generados como resultado del disparo de una transición se analizan para preparar la instrucción SQL que se ejecutará en la BD. La descripción anterior se muestra en la figura 7.1.

ECAPNSim está compuesto por dos bloques, por un lado está el *Kernel de ECAPNSim*, y por

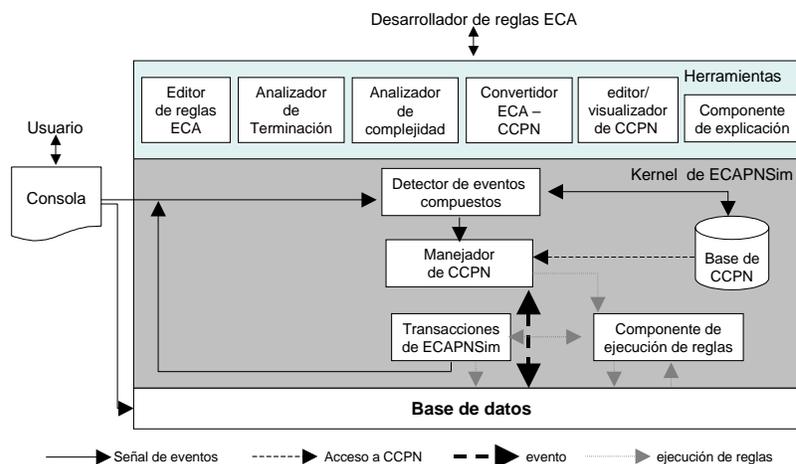


Figura 7.2: Estructura de ECAPNSim dividida en dos bloques, Kernel de ECAPNSim y herramientas de diseño en ECAPNSim.

el otro se encuentra el conjunto de *herramientas de ECAPNSim*. Figura 7.2.

7.2.1. Kernel de ECAPNSim

El kernel de ECAPNSim proporciona la capacidad de reacción, ante la ocurrencia de eventos, a una BD pasiva. El kernel de ECAPNSim está constituido por un *manejador de CCPN*, una *base de CCPN*, un *detector de eventos compuestos* y el *componente de ejecución de reglas*.

Manejador de CCPN. Cuando un evento es detectado por el *detector de eventos compuestos*, el *manejador de CCPN* verifica si este evento pertenece al conjunto de eventos, formado por los eventos considerados en la definición de las reglas ECA. Cuando un evento que pertenece a éste conjunto de eventos es detectado, y el token correspondiente es depositado en el lugar que representa al evento, el *manejador de CCPN* desplaza el token hacia la transición t correspondiente, realizando la activación y el disparo dependiendo si $t \in T_{copy} \cup T_{comp} \cup T_{regla}$. Si $t \in T_{regla}$, entonces el *manejador de CCPN* evalúa la condición de la regla almacenada en t . Cuando la información del token no es suficiente para evaluar a la condición almacenada en t , entonces el *manejador de CCPN* accesa a la BD para conocer su estado y tomar la información necesaria para la evaluación de la condición. Si la evaluación de la condición da como resultado *verdadero*, entonces el *manejador de CCPN* toma

la información correspondiente a la acción a ejecutar y crea un nuevo token con esta información, enviándolo hacia el lugar de la acción (o en su caso lugares, cuando se trate de muchas acciones de una sola regla). Cuando el token llega al lugar de la acción, el *manejador de CCPN* interpreta el token para generar el comando y enviarlo al *componente de ejecución de reglas* para su ejecución dentro de la BD. El procesamiento que realiza el manejador de CCPN termina cuando ya no existen transiciones habilitadas.

Detector de eventos compuestos. Los eventos primitivos y compuestos son monitoreados por este componente. Primero, los eventos primitivos y compuestos son convertidos en una estructura de CCPN, posteriormente son almacenados en la *base de CCPN*. Los eventos compuestos que son reconocidos por el detector de eventos son *conjunción, disyunción, negación, secuencia, simultáneo, primero, último, historia y alguno*. En tiempo de ejecución, el *detector de eventos compuestos* “escucha” todos los movimientos en la BD que modifican su estado, y toma cada uno de los eventos primitivos para mapearlos a la CCPN y a la vez conformar a los eventos compuestos. Cuando un evento relevante toma lugar (primitivo o compuesto), el *detector de eventos compuestos* envía la señal, de que fue detectado, al *manejador de la CCPN* para su procesamiento.

Componente de ejecución de reglas. Cuando una transición $t \in T_{regla}$ es disparada, el *manejador de CCPN* envía la instrucción, al *componente de ejecución de reglas*, para que sea ejecutada la acción de la regla. Esta instrucción está codificada en el lenguaje de consultas SQL, la cual es enviada desde ECAPNSim hacia el SBD a través del controlador JDBC-ODBC, para realizar la acción sobre la BD. La modalidad de ejecución de reglas ofrece ECAPNSim es el “*modo inmediato*”, porque ECAPNSim no utiliza transacciones en el manejo de las operaciones de la BD.

Base de CCPN. En ECAPNSim, las reglas ECA son definidas a través del *editor de reglas ECA*. La base de reglas ECA es convertida en una estructura de CCPN, a la que se ha denominado *base de CCPN*, la cual contiene al conjunto de lugares P , al conjunto de transiciones T y al conjunto de arcos A , que en conjunto forman a la base de reglas ECA definida. La base de CCPN solamente es utilizada por el *manejador de CCPN*, el cual es el que asigna tokens, habilita y dispara transiciones, y toma la información contenida en t sobre la acción de la regla para enviarla al *componente de ejecución de reglas*.

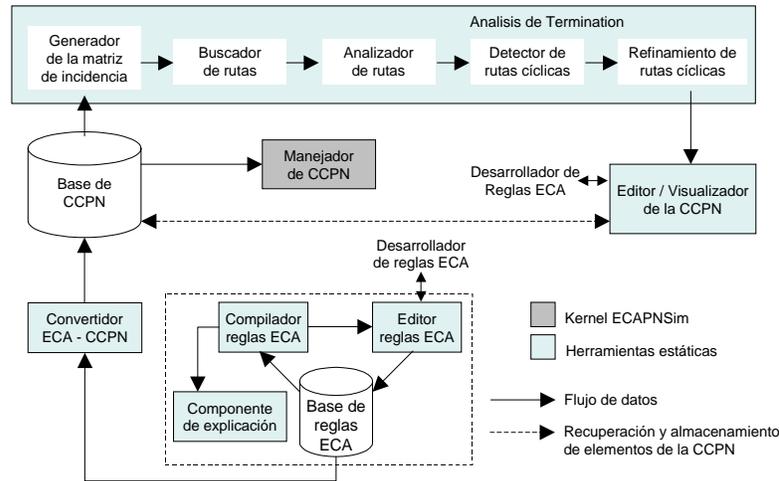


Figura 7.3: Conjunto de herramientas estáticas que ofrece ECAPNSim para el diseño de una base de reglas ECA en CCPN.

7.2.2. Herramientas de ECAPNSim

ECAPNSim ofrece un conjunto de herramientas para ser utilizadas por un desarrollador de reglas ECA. Estas herramientas comprenden un *Editor de reglas ECA*, un *analizador de reglas*, el *analizador de complejidad de la base de reglas ECA*, el *convertidor de reglas ECA a CCPN*, un *editor/visualizador de la base de CCPN*, así como de *componentes de explicación* sobre la ejecución y procesamiento de las reglas en CCPN. Las herramientas utilizadas durante la etapa de construcción de la base de reglas se muestran en la figura 7.3.

Editor de reglas ECA

En el editor de reglas se escriben en modo texto, y bajo una sintaxis establecida, la base el conjunto de reglas que proporcionará comportamiento activo a un SBD pasivo. El editor de reglas ECA se auxilia de un compilador de reglas, para verificar la sintaxis y correcta descripción de reglas ECA para la generación de la CCPN.

Analizador reglas

El analizador de reglas realiza el procedimiento descrito en el capítulo 6, donde se describen los pasos que se siguen en la generación de la matriz de incidencia para la CCPN, la manera en que se encuentran todas las rutas posibles dentro de la CCPN, se realiza un análisis de las rutas encontradas para determinar la presencia de ciclos y, finalmente, se desechan aquellas rutas cíclicas en las que puede determinarse que el disparo infinito de reglas no se alcanzará, de acuerdo a los teoremas presentados en el capítulo 6.

Analizador de complejidad

El analizador de complejidad fue desarrollado como un submódulo de ECAPNSim en [55] [56]. Este analizador lleva a cabo un análisis para determinar la complejidad de una base de reglas ECA, tomando como base de reglas ECA a la estructura de CCPN correspondiente.

Convertidor de reglas ECA a CCPN

Este módulo es el encargado de interpretar el archivo de reglas ECA y generar los objetos para cada uno de los elementos de las reglas. Posteriormente, toma los objetos generados para hacer el mapeo de elemento de reglas ECA a elementos de CCPN, estableciendo, dentro de la CCPN, las relaciones que pudiesen existir entre reglas ECA. El algoritmo utilizado por el convertidor de reglas está descrito en la figura 4.7.

Editor/visualizador de la base de CCPN

ECAPNSim cuenta con una pantalla de visualización para observar la CCPN obtenida de la base de reglas ECA. Cuando una CCPN es creada, el *editor/visualizador* distribuye automáticamente la CCPN obtenida, sin embargo, si el usuario desea una distribución diferente puede modificarla utilizando los botones del ratón para “arrastrar” los elementos de la CCPN a una posición diferente. Además, el usuario puede eliminar algún elemento de la CCPN si así lo desea.

Componentes de explicación.

Este módulo se utiliza para describir los procesos de compilación, de análisis de reglas y el de procesamiento y ejecución de la CCPN. Durante el proceso de compilación, el componente de explicación marca los errores en el que ha incurrido un usuario durante el desarrollo de una base de reglas ECA. Cuando se realiza el proceso de análisis de reglas, el componente de explicación es útil para mostrar las rutas cíclicas que deben monitorearse en tiempo de ejecución. Y en el proceso de disparo de la CCPN, se muestra un resumen sobre las transiciones que cumplen con las condiciones de habilitación y disparo.

7.3. Diagrama de clases

En POO se utilizan los conceptos de herencia y composición [74]. En el diseño del diagrama de clases, la herencia se representa con una flecha que conecta a la clase hija con la superclase, la clase hija hereda atributos y métodos de la superclase. Gráficamente la clase hija se coloca por debajo de la superclase y la flecha llega a la parte inferior de la superclase. Una clase es una composición de otra cuando forma parte de sus atributos. La composición se representa con una flecha que parte de la clase atributo hacia el lado izquierdo de la clase que la contiene. El diagrama de las clases que se utilizaron en la implementación de ECAPNSim se presentan en la figura 7.4.

La clase principal es *ECAPNSim*, la cual es una extensión de la clase *JFrame* de Java y es una composición de las clases *AreaDibujo*, *Menús*, *BarraHerram*, *Propiedades*, *Velocidad*, *FiltroArch* y *Conexión*. La clase *AreaDibujo* es una extensión de la clase de Java *JPanel* y es una composición de las clases *ControlAnimación*, *ECA*, *Transición*, *Lugar*, *Hoja*, *Arbol*, *ArcoTP* y *ArcoPT*. La clase *ECA* es una composición de la clase *Condición*. La clase *Lugar* es una composición de las clases *Tabla* y *Token*, hereda los atributos de la clase *Círculo*, que a la vez hereda de la clase *Punto*. La clase *Rectángulo* es una composición de la clase *Punto*. La clase *Transición* hereda los atributos de la clase *Rectángulo*. La clase *Arco* hereda los atributos de la clase *Línea*, que a la vez hereda de la clase *Punto*. Las clases *ArcoPT* y *ArcoTP* heredan de la clase *Arco*, sobre escribiendo los métodos donde es necesario diferenciar entre un arco de lugar de entrada y uno de lugar de salida.

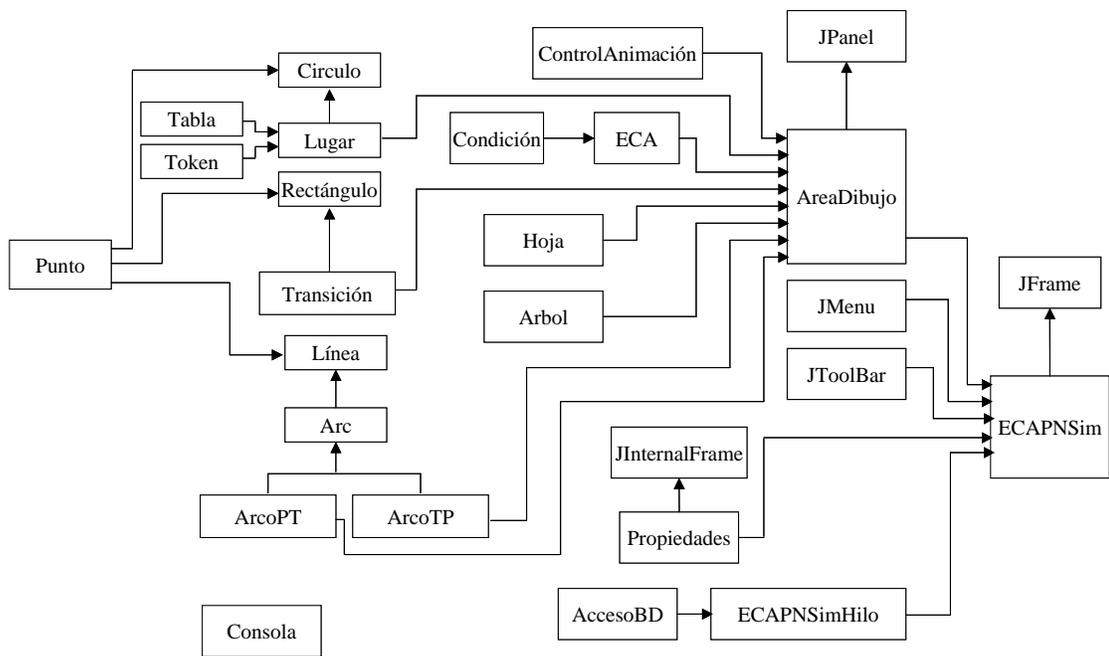


Figura 7.4: Diagrama de clases del ECAPNSim.

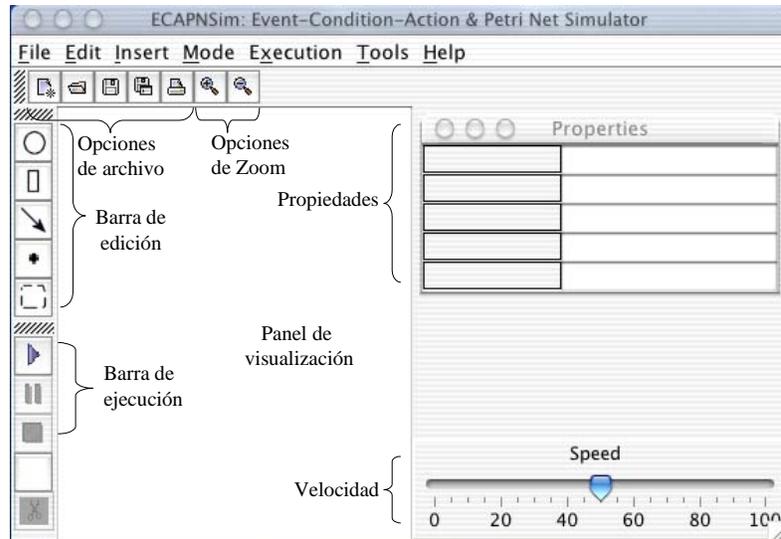


Figura 7.5: Pantalla de visualización de ECAPNSim.

7.4. Diseño e Implementación

ECAPNSim ofrece una interfaz gráfica con menús y barras de herramientas. El área de diseño o de dibujo proporciona elementos para la edición y diseño de CCPN. Las opciones de edición y diseño son accedidas a través de las barras de herramientas adyacentes al área de dibujo, o bien, a través de los menús desplegables. La barra de herramientas proporciona opciones para salvar y guardar archivos. Además, dentro del menú "Archivo" cuenta con una opción para importar reglas ECA desde archivo y generar la CCPN correspondiente. Las características de la interfaz gráfica se muestran en la figura 7.5.

Los campos de los elementos pueden modificarse por medio de la ventana de propiedades. Características como el número de tokens de los lugares, la información del token ó el peso de los arcos pueden especificarse en esta ventana. Además, el usuario puede desplazar elementos de la CCPN hacia otro lugar.

Dado que el ECAPNSim funciona como un servidor, se implementó una clase que proporciona una consola de edición al usuario. En la consola el usuario escribe las instrucciones SQL que se envían al ECAPNSim. Estas instrucciones se convierten en tokens para su distribución dentro de la

estructura CCPN. La consola tiene la capacidad de comunicarse con el ECAPNSim para funcionar como cliente, dentro del esquema Cliente-Servidor.

Se implementó una clase para realizar la conexión de ECAPNSim con la BD cuando se encuentre en la modalidad "Real". De esta manera, ECAPNSim ejecuta instrucciones SQL dentro de la BD correspondientes a la acción de las reglas disparadas.

7.5. Plataforma independiente

Para mostrar la independencia de plataforma que ofrece ECAPNSim, se ha desarrollado una base de reglas ECA en CCPN, haciendo la conexión de ECAPNSim con diferentes manejadores de bases de datos relacionales. Entre los sistemas manejadores de bases de datos con que se trabajó están MS Access, MS Visual Fox Pro, Oracle y SQL Server que corren bajo la plataforma del sistema operativo Windows; y Postgresql, corriendo en la plataforma de Machintosh.

7.5.1. Base de reglas

El esquema conceptual de la base de datos donde se aplicarán las reglas ECA está conformado por tres tablas:

```
EMPLEADO(emp_id, rango, nombre, rango, salario)
```

```
PRIMA(emp_id, cantidad)
```

```
VENTAS(emp_id, mes, numero)
```

La base de reglas ECA está conformada por las siguientes seis:

Regla 1.- Cuando la prima de un empleado se incrementa por más de \$1000, el rango del empleado se incrementa en 1.

Regla 2.- Cuando el rango de un empleado es modificado (supongamos incrementado), la prima para ese empleado se incrementa en 10 veces el nuevo rango.

Regla 3.- Cuando las ventas de un empleado para un mes son mayores a 50, se incrementan \$100 al salario de ese empleado y la cantidad de la prima se incrementa en \$1000.

Regla 4.- Cuando las ventas del mes de un empleado son mayores a 100, entonces el rango del empleado se incrementa en 1.

Regla 5.- Cuando el rango de un empleado llega a 15, el salario del empleado se incrementa en un 10 %.

Regla 6.- Cuando las ventas del mes de un empleado son mayores a 50 y su rango está por debajo de 15, entonces su prima se decrementa en \$1000.

Las reglas anteriores convertidas en la sintáxis reconocida por ECAPNSim quedan de la siguiente manera:

```
//Definicion de tablas
EMPLEADO(emp_id integer, nombre CHAR( length), rango integer , salario float);
PRIMA(emp_id integer, cantidad float);
VENTAS(emp_id integer, mes integer, numero integer);

//Definicion de eventos
e0 : update_PRIMA_cantidad;
e1 : update_EMPLEADO_rango;
e2 : insert_VENTAS;

//Definicion de reglas
Rule 001,
on e0,
if (update.cantidad - old.cantidad) > 1000,
Then update EMPLEADO set rango = old.rango + 1 where emp_id = update.emp_id;

Rule 002,
on e1,
if update.rango > old.rango,
Then update PRIMA set cantidad = old.cantidad + update.rango * 10 where emp_id = update.emp_id;

Rule 003,
on e2,
if insert.numero > 50,
```

Then update EMPLEADO set salario = old.salario+100 where emp_id = insert.emp_id &
update PRIMA set cantidad = old.cantidad+1000 where emp_id = insert.emp_id;

Rule 004,

on e2,

if insert.numero > 100,

Then update EMPLEADO set rango = old.rango+1 where emp_id = insert.emp_id;

Rule 005,

on e1,

if update.rango = 15,

Then update EMPLEADO set salario = old.salario*1.1 where emp_id = update.emp_id;

Rule 006,

on e2,

if insert.numero > 50 & EMPLEADO.rango < 15,

Then update PRIMA set cantidad = old.cantidad - 1000 where emp_id = insert.emp_id;

Supongamos que el estado de la BD es tal y como se muestra en la figura 7.6, sobre el que se observarán los cambios que sean generados automáticamente por la base de reglas.

La CCPN que ECAPNSim genera para esta base de reglas se muestra en la figura 7.7, donde las reglas 1, 2, 3, 4, 5 y 6, son representadas por las transiciones t_0, t_1, t_2, t_3, t_5 y t_7 , respectivamente. Los eventos e_0, e_1 y e_2 son denotados por los lugares p_0, p_1 y p_2 , respectivamente. El lugar p_3 representa a la acción de las reglas 3 y 5.

Ahora se describirá la conexión de ECAPNSim con diferentes sistemas manejadores de bases de datos utilizando la CCPN de la figura 7.7.

7.5.2. MS Access

Microsoft[®] Access[®] es un sistema manejador de bases de datos que viene incluido en la suite de MS Office. Access es un sistema manejador de bases de datos relacionales que corre bajo el ambiente de Windows[®]. La definición de las tablas en MS Access se muestran en la figura 7.8.

Tabla EMPLEADO

emp_id	nombre	rango	salario
1	Jose Romero	2	7,500.00
2	Juan Mendoza	3	8,000.00
3	Martha Villanueva	1	6,800.00
4	Marcos Andrade	4	10,000.00

Tabla PRIMA

emp_id	cantidad
1	860.00
2	950.00
3	1,500.00
4	2,500.00

Tabla VENTAS

emp_id	mes	número
Tabla vacía		

Figura 7.6: Estado de la Base de Datos al iniciar el procesamiento de reglas.

El estado de la BD descrito en la figura 7.6, ahora se muestra dentro del ambiente de Access. Figura 7.9.

La interfaz de ECAPNSim ejecutándose en Windows se muestra en la figura 7.10.

Para llevar a cabo la conexión con MS Access se debe especificar la fuente de datos ODBC en el sistema operativo, en este caso, a ésta base de datos la hemos denominado TesisPhD2.mdb. Se establece el puente ODBC con esta base de datos y se pueda acceder desde el ECAPNSim.

Si enviamos la instrucción “update PRIMA set cantidad = 1900 where emp_id = 1;” a ECAPNSim por medio de la consola auxiliar (figura 7.11), se genera un token correspondiente a la modificación del campo cantidad de la tabla PRIMA, y se coloca en el lugar p_0 correspondiente, figura 7.12. Posteriormente se habilita la transición t_0 (que representa a la regla 1) y se evalúa la condición (if (update.cantidad - old.cantidad) > 1000). Si evaluamos esta condición contra los valores almacenados en el token creado, podemos ver que la condición resulta verdadera (update.cantidad = 1900, old.cantidad = 850, $1900 - 850 = 1050$, $1050 > 1000$). Por lo tanto se genera un nuevo token, ahora con información respecto a la acción de la regla y se envía p_1 , lugar de salida de t_0 (figura 7.13).

El estado de la BD, después del disparo de la regla 1 (t_0), se muestra en la figura 7.14. En esta

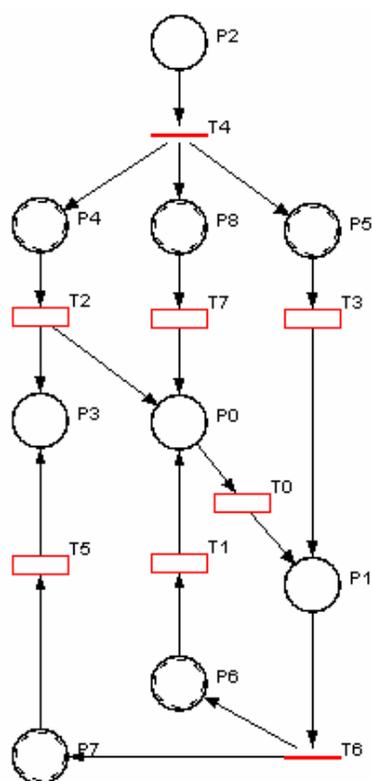


Figura 7.7: CCPN obtenida automáticamente por ECAPNSim de las seis reglas ECA.

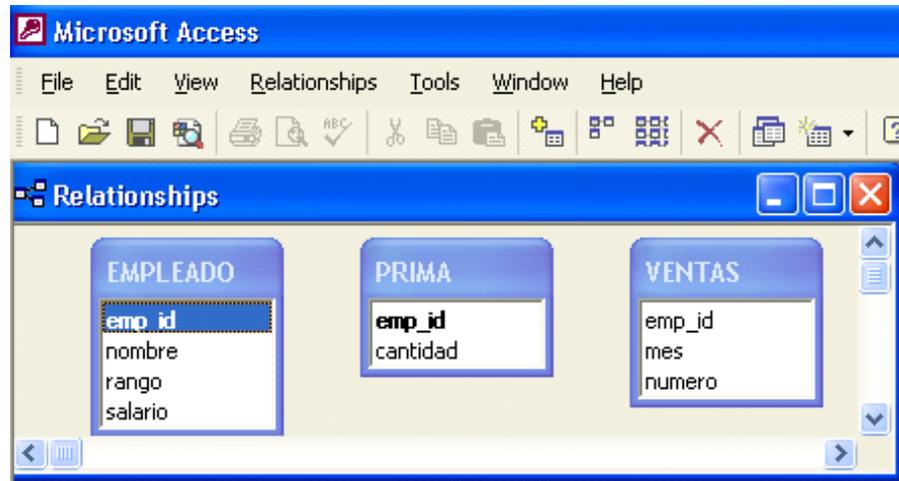


Figura 7.8: Definición del esquema conceptual de la BD en MS Access[®].

imagen puede observarse el cambio producido por el comando escrito en la consola (se modifica a 1900 el valor del campo cantidad de la tabla PRIMA para el registro con emp_id igual a 1). Además, el cambio producido por el disparo de la regla 1, cuya acción específica que el rango para este empleado se incrementa en uno; como el valor que tenía el campo rango para el empleado cuyo emp_id es igual a 1 era 2, ahora el valor para el atributo rango es de 3.

7.5.3. FoxPro

Fox Pro[®] es un sistema manejador de bases de datos relacionales que venía corriendo en ambientes de MS DOS, sin embargo actualmente se encuentra incluido el Visual Fox Pro[®] dentro del Visual Studio[®], ofreciendo un ambiente visual para el desarrollo de sistemas de bases de datos. al igual que MS Access, Visual Fox Pro corre bajo el ambiente de Windows[®]. La definición de las tablas dentro del ambiente de Visual Fox Pro se muestran en la figura 7.15.

El estado de la BD descrito en la figura 7.6, ahora se muestra dentro del ambiente de Visual Fox Pro. Figura 7.16.

Al igual que en Access, se debe especificar la fuente de datos ODBC en el sistema operativo. La base de datos diseñada en Visual Fox Pro tiene por nombre TesisPhD2fp.dbc (figura 7.15).

Si deseamos modificar el valor para el campo rango de la tabla EMPLEADO, enviamos la

The image shows three overlapping table windows from Microsoft Access. The top window is titled 'EMPLEADO : Table' and contains a table with columns 'emp_id', 'nombre', 'rango', and 'salario'. The second window is titled 'VENTAS : Table' and contains a table with columns 'emp_id', 'mes', and 'numero'. The bottom window is titled 'PRIMA : T...' and contains a table with columns 'emp_id' and 'cantidad'. Each window includes a record navigation bar at the bottom.

	emp_id	nombre	rango	salario
▶ +	1	Jose Romero	2	7500
+	2	Juan Mendoza	3	8000
+	3	Martha Villanueva	1	6800
+	4	Marcos Andrade	4	10000
*	0		0	0

Record: 1 of 4

	emp_id	mes	numero
▶	0	0	0

Record: 1 of 1

	emp_id	cantidad
▶ +	1	860
+	2	950
+	3	1500
+	4	2500
*	0	0

Record: 1

Figura 7.9: Estado de la Base de Datos, visto en el ambiente de Access, al iniciar el procesamiento de reglas.

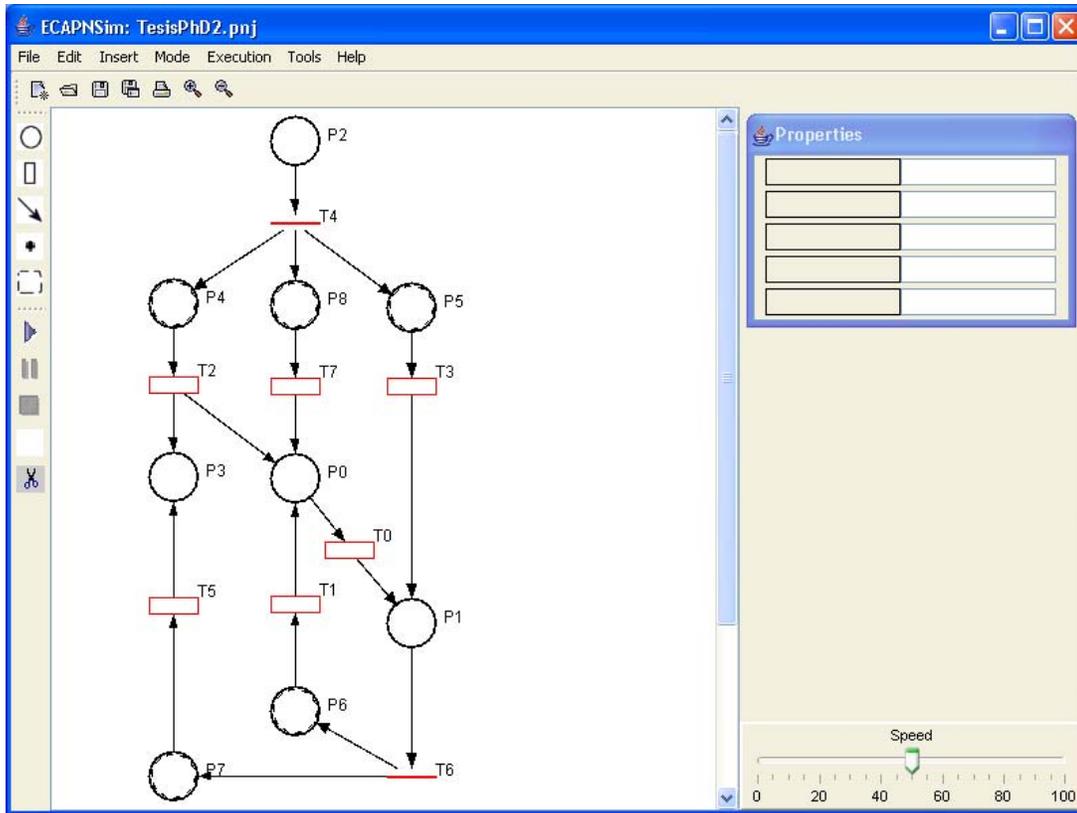


Figura 7.10: ECAPNSim ejecutándose bajo el ambiente de Windows®.

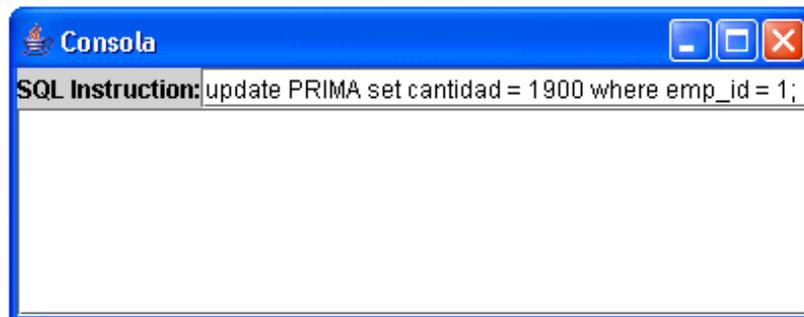


Figura 7.11: Consola con la instrucción de modificación del campo cantidad de la tabla PRIMA.

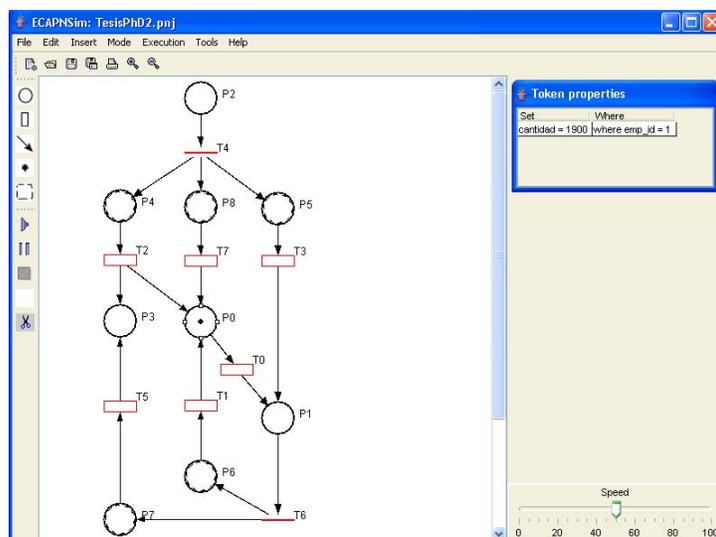


Figura 7.12: Estado de la CCPN al detectar el evento e_0 .

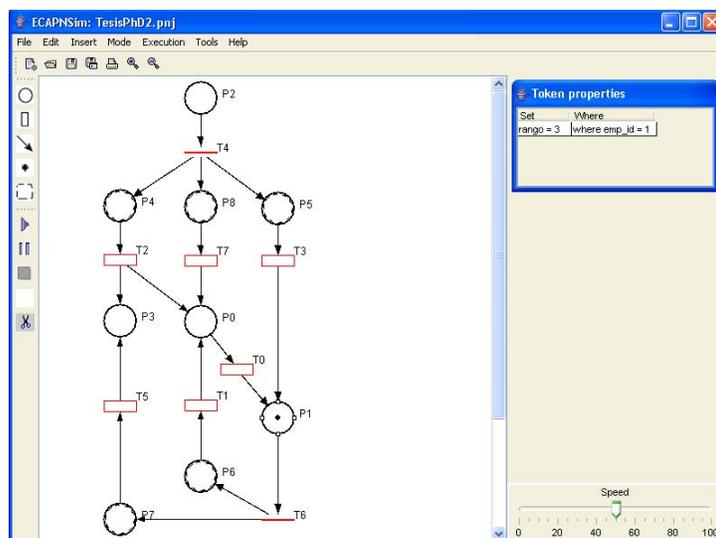


Figura 7.13: Estado de la CCPN después del disparo de la transición t_0 .

The image shows two overlapping database window screenshots. The top window, titled 'EMPLEADO : Table', displays a table with four columns: emp_id, nombre, rango, and salario. It contains four data rows and a summary row with emp_id 0. The bottom window, titled 'PRIMA : T...', displays a table with two columns: emp_id and cantidad. It contains four data rows and a summary row with emp_id 0. Both windows have navigation controls at the bottom, including arrows and a record number (1) out of a total of 4 records.

	emp_id	nombre	rango	salario
▶ +	1	Jose Romero	3	7500
+	2	Juan Mendoza	3	8000
+	3	Martha Villanueva	1	6800
+	4	Marcos Andrade	4	10000
*	0		0	0

Record: 1 of 4

	emp_id	cantidad
▶ +	1	1900
+	2	950
+	3	1500
+	4	2500
*	0	0

Record: 1

Figura 7.14: Estado final de la BD después del disparo de t_0 .

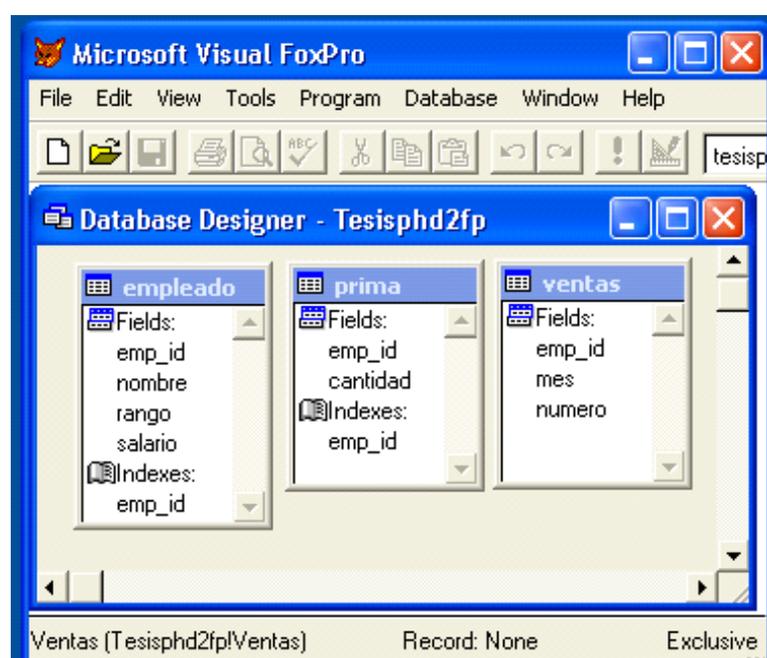


Figura 7.15: Definición del esquema conceptual de la BD en Visual Fox Pro®.

The image shows three data windows from Visual Fox Pro. The 'Empleado' window displays a list of employees with their IDs, names, ranks, and salaries. The 'Prima' window displays a list of bonuses with employee IDs and amounts. The 'Ventas' window is empty, showing only column headers for employee ID, month, and quantity.

Empleado			
Emp_id	1		
Nombre	Jose Romero		
Rango	2		
Salario	7500.00		
Emp_id	2		
Nombre	Juan Mendoza		
Rango	3		
Salario	8000.00		
Emp_id	3		
Nombre	Martha Villanueva		
Rango	1		
Salario	6800.00		
Emp_id	4		
Nombre	Marcos Andrade		
Rango	4		
Salario	10000.00		

Prima		
Emp_id	1	
Cantidad		860.00
Emp_id	2	
Cantidad		950.00
Emp_id	3	
Cantidad		1500.00
Emp_id	4	
Cantidad		2500.00

Ventas		
Emp_id	Mes	Numero

Figura 7.16: Estado de la Base de Datos, visto en el ambiente de Visual Fox Pro, al iniciar el procesamiento de reglas.

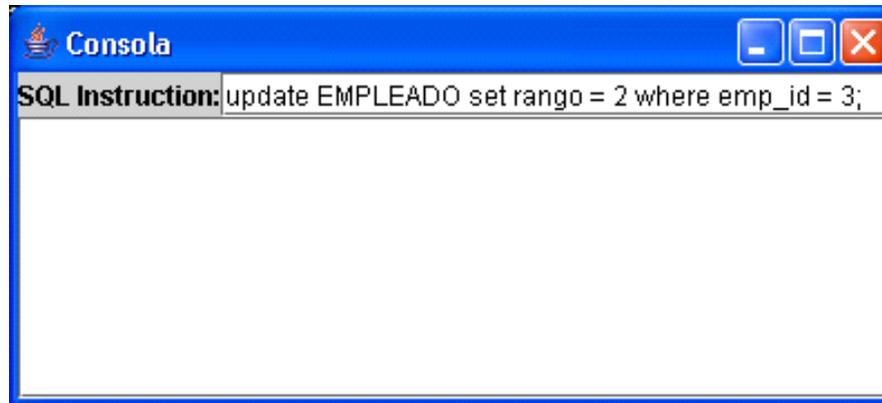


Figura 7.17: Consola con la instrucción de modificación del campo rango de la tabla EMPLEADO.

instrucción “update EMPLEADO set rango = 2 where emp_id = 3;” a ECAPNSim por medio de la consola auxiliar (figura 7.17), se genera el evento definido como e_1 .

Se genera el token correspondiente a la modificación y se coloca en el lugar p_1 correspondiente, figura 7.18. Debido a que el evento e_1 es utilizado en la habilitación de dos reglas, se tiene que duplicar la información del evento en dos tokens, y éstos sean utilizados en la evaluación de las condiciones de las reglas. Por lo tanto, p_1 envía el token del evento a la transición $t_6 \in T_{copy}$ y ésta a su vez envía un token a p_6 y otro idéntico a p_7 , $p_6, p_7 \in P_{copy}$, figura 7.19.

Las transiciones $t_1, t_5 \in T_{regla}$ son habilitadas (estas transiciones corresponden a las reglas 2 y 5, respectivamente), figura 7.20.

Se evalúan las condiciones almacenadas en t_1 y t_5 , donde puede observarse que solamente la condición de t_1 se cumple. En el caso de la condición de t_5 (regla 5, if update.rango = 15) el valor para el rango modificado es de 2, y no se cumple que sea igual a 15. Por otro lado, la condición de la transición t_1 (regla 2, if update.rango > old.rango), el valor actualizado del rango es 2 y el valor anterior de este campo para el empleado cuyo campo emp_id era 1, por lo tanto se cumple la condición y la transición se dispara (figura 7.21), creando el token correspondiente a la acción de la regla 2 (update PRIMA set cantidad = old.cantidad + update.rango * 10 where emp_id = update.emp_id; donde old.cantidad = 1500, update.rango = 2, entonces la expresión queda como update PRIMA set cantidad = 1520 where emp_id = 3;) y enviándolo al lugar p_0 , $t_1^\bullet = \{p_0\}$, que representa la acción de

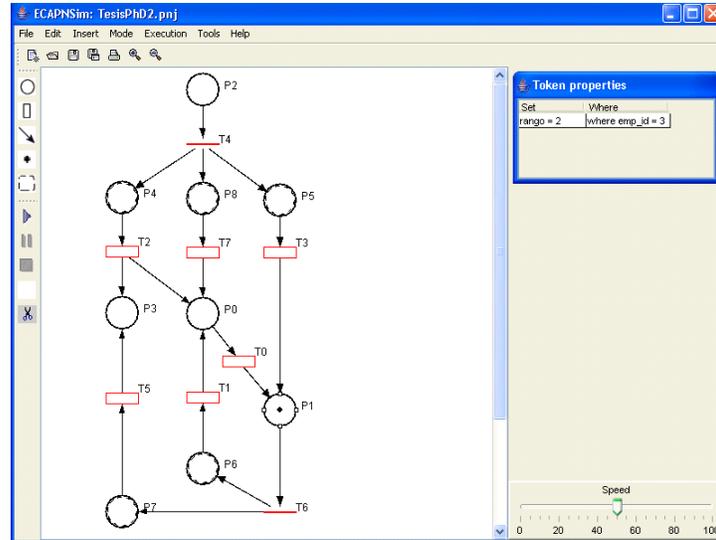


Figura 7.18: Estado de la CCPN al detectar el evento e_1 .

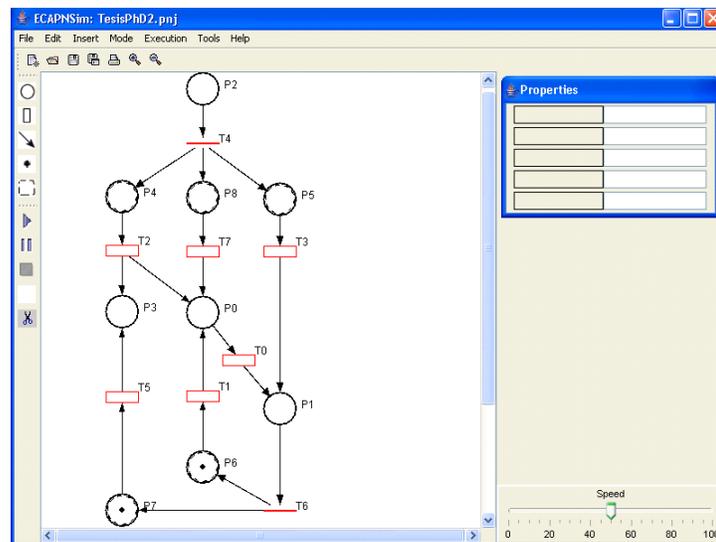


Figura 7.19: Estado de la CCPN después del disparo de $t_6 \in T_{copy}$.

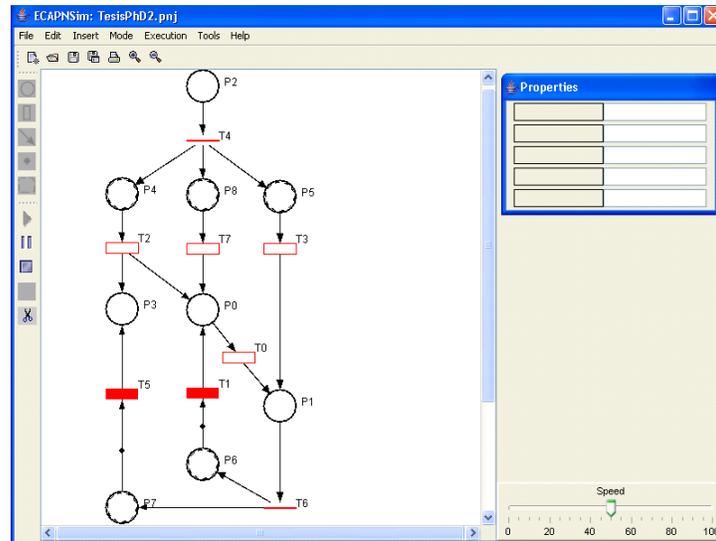


Figura 7.20: Activación de las transiciones $t_1, t_5 \in T_{regla}$.

la regla (update_PRIMA_cantidad).

El estado de la CCPN después del disparo de t_1 se muestra en la figura 7.22, donde se puede ver que el valor del token que se generó contiene los valores correspondientes a la acción de la regla 2.

El estado final de la BD después del disparo de la regla 2 se muestra en la figura 7.23. en esta imagen ya se encuentra modificado el valor del rango para el empleado cuyo campo emp_id es 3, antes de la ejecución del comando tenía un valor de 1. Además, el disparo de la regla 2 generó una modificación automática a la BD, la cual, de acuerdo a la acción de ésta regla, se modificó el valor del campo cantidad en la tabla PRIMA para el empleado con emp_id igual a 3, asignándole el valor de 1520.

7.5.4. Postgresql

Postgresql es un sistema de base de datos relacional de código abierto, desarrollado a partir del código de Postgres (universidad de Berkeley en California) y de las capacidades que ofrece el lenguaje SQL. Postgresql se ejecuta tanto en ambientes de Linux como ambientes de Macintosh.

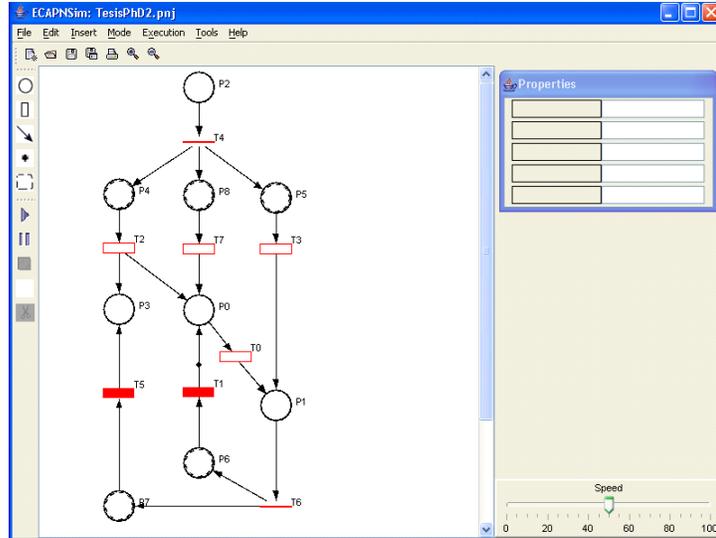


Figura 7.21: Disparo de la transición t_1 .

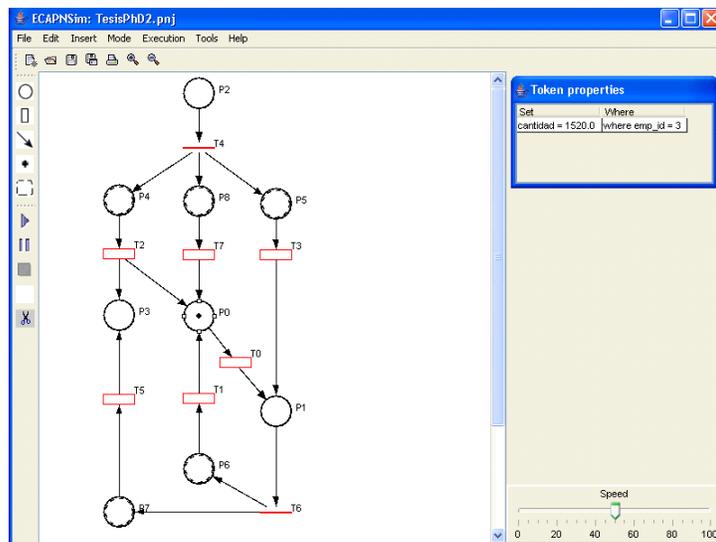


Figura 7.22: Estado final de la CCPN después del disparo de t_1 .

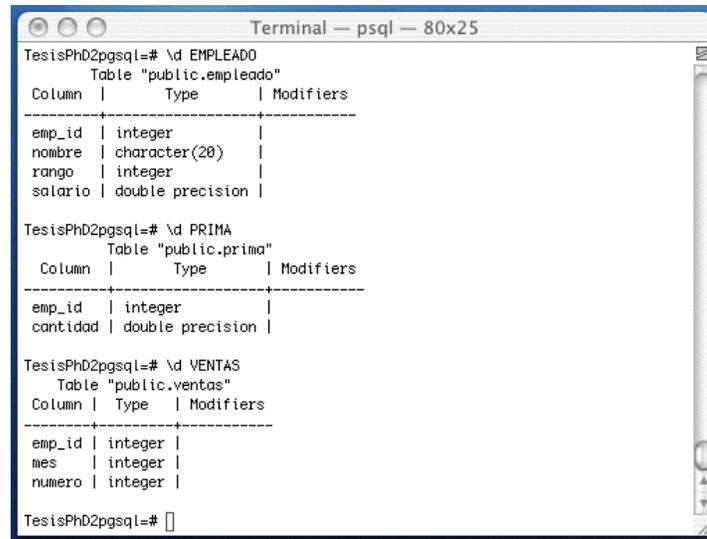
The image shows three windows from a database management system. The 'Empleado' window displays a list of employees with their IDs, names, ranks, and salaries. The 'Prima' window displays a list of bonuses with employee IDs and amounts. The 'Ventas' window shows a table with columns for employee ID, month, and number, but it is currently empty.

Emp_id	Nombre	Rango	Salario
1	Jose Romero	2	7500.00
2	Juan Mendoza	3	8000.00
3	Martha Villanueva	2	6800.00
4	Marcos Andrade	4	10000.00

Emp_id	Cantidad
1	860.00
2	950.00
3	1520.00
4	2500.00

Emp_id	Mes	Numero
--------	-----	--------

Figura 7.23: Estado final de la BD después del disparo de la transición t_1 .



```

Terminal — psql — 80x25
TesisPhD2pgsql=# \d EMPLEADO
Table "public.empleado"
Column | Type | Modifiers
-----|-----|-----
emp_id | integer |
nombre | character(20) |
rango | integer |
salario | double precision |

TesisPhD2pgsql=# \d PRIMA
Table "public.prima"
Column | Type | Modifiers
-----|-----|-----
emp_id | integer |
cantidad | double precision |

TesisPhD2pgsql=# \d VENTAS
Table "public.ventas"
Column | Type | Modifiers
-----|-----|-----
emp_id | integer |
mes | integer |
numero | integer |

TesisPhD2pgsql=#

```

Figura 7.24: Definición de las tablas utilizadas, dentro del ambiente de Postgresql.

En este ejemplo se ejecutó ECAPNSim en el ambiente de Macintosh, utilizando como repositorio de datos una BD desarrollada en Postgresql.

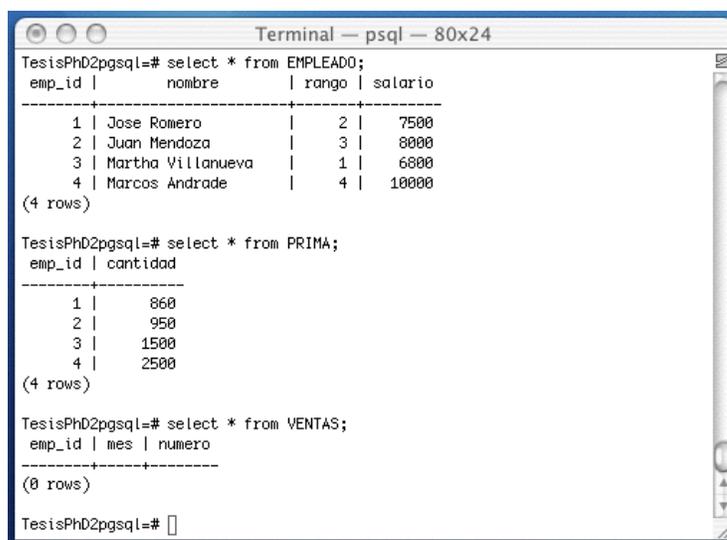
La definición de las tablas en el ambiente de postgresql se muestra en la figura 7.24.

Al igual que en los casos anteriores, se parte del mismo estado inicial, el cual se muestra en la figura 7.25, ahora en el ambiente de Postgresql.

La conexión de ECAPNSim con la BD de postgresql se hace mediante el driver JDBC que ofrece Postgres en formato JAR, el archivo se llama *postgres.jar*, y contiene clases que permiten la conexión de un programa desarrollado en Java con una BD definida dentro de Postgresql.

En esta subsección se muestra la ejecución de las reglas 3, 4 y 6, las cuales se habilitan con el evento e_2 , relacionado con la inserción de un registro a la tabla de VENTAS. Para tal efecto, se agrega un registro que contenga las ventas que realizó en el mes de agosto (mes ocho) el empleado con identificador igual a 1, quien realizó un total de 60 ventas durante el mes. La instrucción que se teclea en la consola es “insert into VENTAS values (1, 8, 60)”. Figura 7.26.

Con esta instrucción, ECAPNSim genera un token cuyos valores serán los datos del registro que se está agregando. El token generado se coloca en el lugar correspondiente a la inserción de



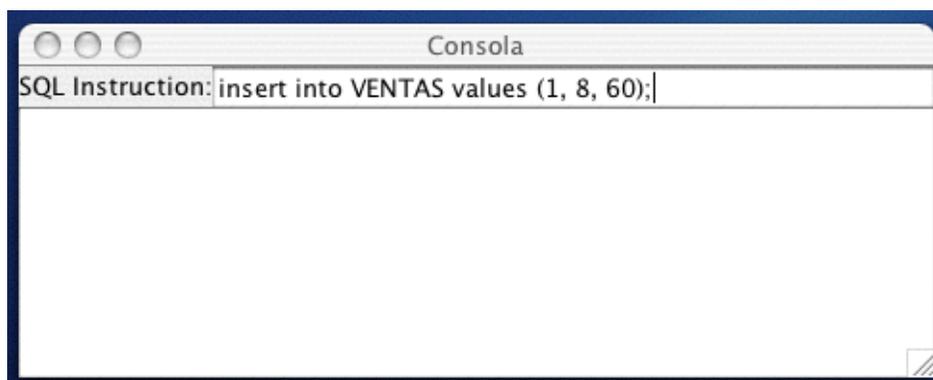
```
Terminal — psql — 80x24
TesisPhD2pgsql=# select * from EMPLEADO;
 emp_id | nombre           | rango | salario
-----+-----+-----+-----
      1 | Jose Romero      |      2 |    7500
      2 | Juan Mendoza     |      3 |    8000
      3 | Martha Villanueva |      1 |    6000
      4 | Marcos Andrade   |      4 |   10000
(4 rows)

TesisPhD2pgsql=# select * from PRIMA;
 emp_id | cantidad
-----+-----
      1 |    800
      2 |    950
      3 |   1500
      4 |   2500
(4 rows)

TesisPhD2pgsql=# select * from VENTAS;
 emp_id | mes | numero
-----+-----+-----
(0 rows)

TesisPhD2pgsql=#
```

Figura 7.25: Estado inicial de la BD.



```
Consola
SQL Instruction: insert into VENTAS values (1, 8, 60);
```

Figura 7.26: Consola donde se muestra la instrucción con se agrega un nuevo registro a la tabla VENTAS.

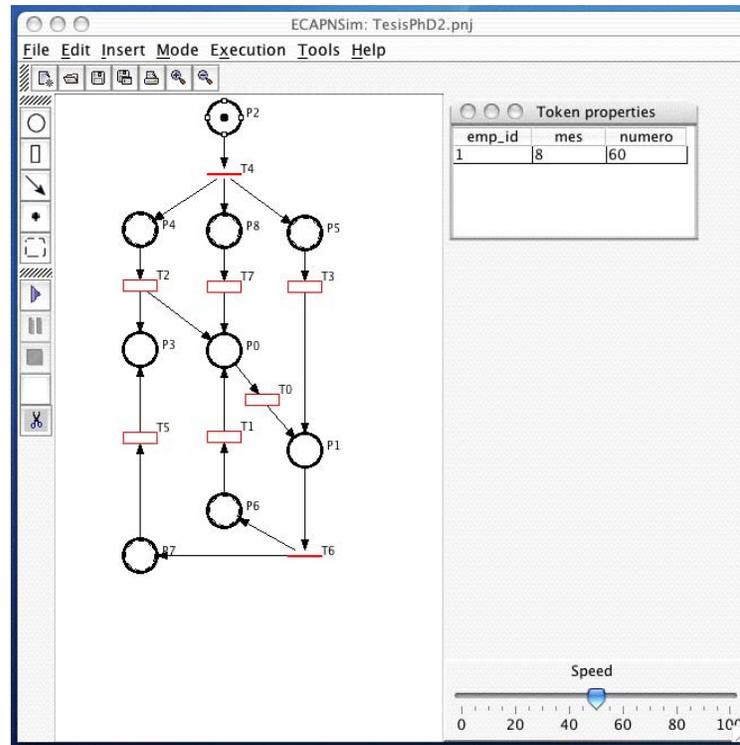


Figura 7.27: Colocación del token correspondiente a la inserción del nuevo registro en la tabla VENTAS.

registros de la tabla VENTAS, el cual es p_2 (figura 7.27). Se comienza con la animación del juego de tokens. p_2 es lugar de entrada de la transición $t_4 \in T_{copy}$, entonces se mandan copias del token hacia los lugares $p_4, p_5, p_8 \in P_{copy}$, $t_4^* = \{p_4, p_5, p_8\}$, los cuales son los lugares de entrada para las transiciones t_2 (regla 3), t_3 (regla 4), t_7 (regla 6) $\in T_{regla}$, respectivamente (figura 7.28).

Con el token replicado en los lugares p_4, p_5 y p_8 , ECAPNSim evaluará las reglas representadas por las transiciones t_2, t_3 y t_7 . La condición de la regla 3 almacenada en la transición t_2 puede constatarse que su evaluación resulta verdadera ($60 > 50$), y por lo tanto las dos acciones de la regla 3 se ejecutan en la BD. En la primera acción se modifica el campo salario de la tabla EMPLEADO, asignándoles 100 pesos más al salario que tenía, quedando con un valor de \$7600.00; además, se modifica el campo cantidad de la tabla PRIMA, aumentándole 1000 pesos más al valor

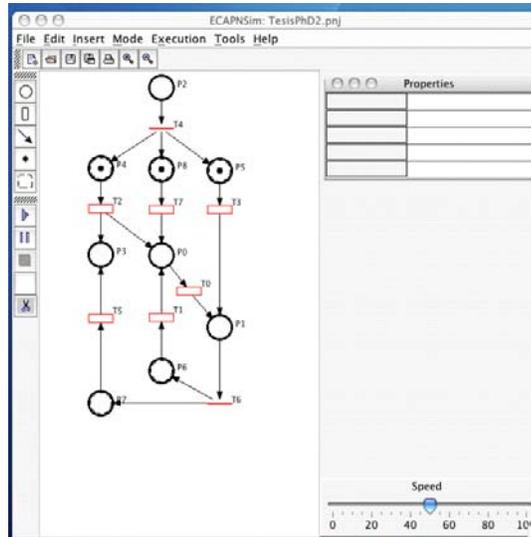


Figura 7.28: Estado de la CCPN después del disparo de la transición $t_4 \in T_{copy}$

que tenía, es decir, el nuevo valor para este campo del empleado 1 es de \$1,860.00. La condición almacenada en la transición t_3 no se satisface, porque las ventas registradas no son mayores de 100 ($60 \not> 100$), y por tal motivo la acción de ésta regla no se ejecuta. Finalmente, la condición compuesta de la regla 6, denotada por la transición t_7 , se satisface ($60 > 50 \wedge rango < 15$), el valor del rango para el `emp_id = 1` es de 2, por lo tanto se cumple que $2 < 15$ y la transición t_7 es disparada; la acción de la regla 6 indica que la cantidad de prima asignada se decrementa en \$1,000.00, la cual había sido incrementada en exactamente la misma cantidad por la otra reglas, quedando finalmente como estaba (\$860.00). Figura 7.29.

El estado final de la CCPN después del disparo de las transiciones se muestra en la figura 7.30, y el estado final de la BD después de la ejecución de las acciones de las reglas disparadas se muestra en la figura 7.31.

7.6. Comentarios

Actualmente existen SMDB que soportan la definición de reglas ECA mediante “triggers”; sin embargo, los “triggers” presentan muchas restricciones que limitan la potencia que un sistema de

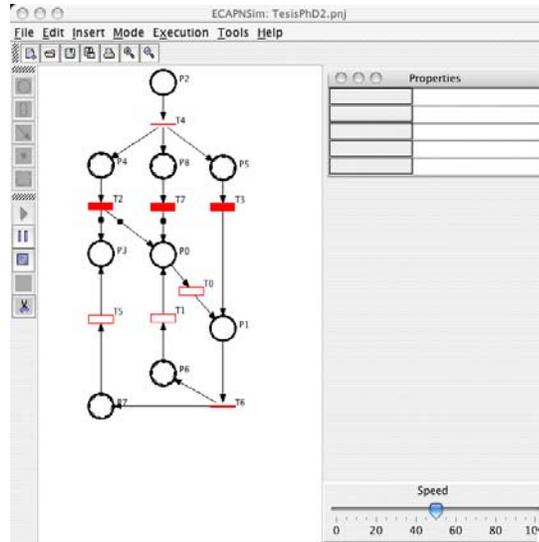


Figura 7.29: Disparo de las transiciones las t_2 y t_7 cuyas condiciones se satisfacen.

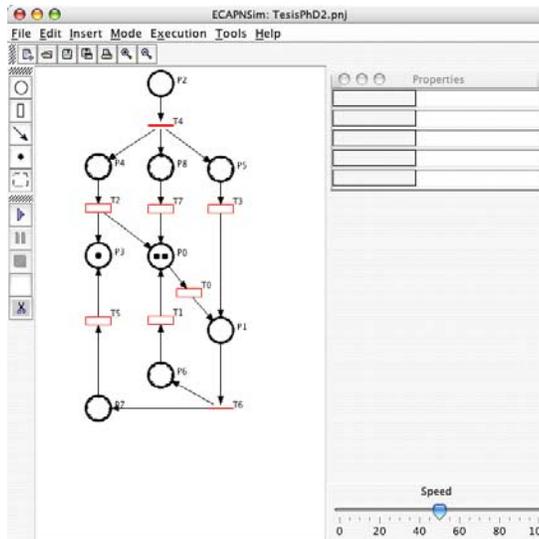


Figura 7.30: Estado de la CCPN después del disparo de las transiciones t_2 y t_7 .

```

TesisPhD2pgsql=# select * from EMPLEADO order by emp_id;
 emp_id | nombre           | rango | salario
-----+-----+-----+-----
      1 | Jose Romero     |      2 |    7600
      2 | Juan Mendoza   |      3 |    8000
      3 | Martha Villanueva |      1 |    6800
      4 | Marcos Andrade |      4 |   10000
(4 rows)

TesisPhD2pgsql=# select * from PRIMA order by emp_id;
 emp_id | cantidad
-----+-----
      1 |      860
      2 |      950
      3 |     1500
      4 |     2500
(4 rows)

TesisPhD2pgsql=# select * from VENTAS;
 emp_id | mes | numero
-----+-----+-----
      1 |   8 |      60
(1 row)

TesisPhD2pgsql=#

```

Figura 7.31: Estado de la BD después del disparo de las reglas 3 y 6.

BDA debe ofrecer.

Por otro lado, se tienen prototipos de investigación que también soportan la definición de reglas ECA, y ofrecen mayor potencia que el manejo de “triggers”. Además, soportan la definición de eventos compuestos como la disyunción, conjunción, secuencia, entre otros. Sin embargo, a semejanza de los anteriores sistemas, la definición de las reglas ECA se lleva a cabo en base a una sintaxis propia del programa y solo trabajan sobre un SMBD.

ECAPNSim es una interfaz que genera la CCPN a partir de la definición de reglas ECA en su forma *on-if-then*. Esta interfaz lleva a cabo la simulación del comportamiento de la CCPN, así como ejecución en cualquier SBD relacional que soporte una conexión por medio del puente ODBC-JDBC.

Además, aprovechando la independencia de plataforma que ofrece el lenguaje Java (utilizado en la implementación de ECAPNSim), es posible proporcionarle el comportamiento activo a SBD relacionales que corran en cualquier plataforma que soporte Java.

ECAPNSim implementa también un módulo para llevar a cabo el análisis estático de la CCPN, utilizando el algoritmo descrito en el capítulo anterior.

Capítulo 8

Caso de estudio

Se desarrolló un caso de estudio para las reglas que existen en una institución bancaria, de acuerdo a los procesos, restricciones y movimientos que automáticamente pueden ser ejecutados, de acuerdo a ciertos parámetros que existan en el contenido de la BD.

Se utilizó el modelo relacional para describir el universo de discurso de la BD para este caso de estudio. Las relaciones y sus atributos se describen a continuación:

- **CLIENTE** (*número, nombre, dirección, y la fecha de inicio*). Almacena información acerca de los clientes del banco, donde se considera el número de cliente, el nombre, su domicilio, y la fecha cuando comenzó como cliente del banco.
- **CUENTA** (*número, número de cliente, saldo, límite, tipo {crédito, ahorro, nómina, cheques}, estado (activada, bloqueada, cancelada)*). Contiene datos sobre las cuentas que maneja el banco, tales como los números de cuenta; el número del cliente propietario de la cuenta; el saldo actual; el total de retiros realizados desde cajero automático; el tipo de la cuenta, la cual puede ser una cuenta de crédito, una cuenta de ahorros, una cuenta de nómina o una cuenta de cheques; además, es necesario conocer si una cuenta está activada, bloqueada o cancelada.
- **TRANSACCIONES PENDIENTES** (*clave, tipo {depósito, venta}, cuenta origen, cuenta destino, cantidad, periodo*). Esta relación se utiliza para almacenar aquellas transacciones que fueron realizadas fuera del horario normal de atención del banco.

- **DEPOSITO** (*fecha de depósito, número de cuenta, cantidad, lugar {sucursal, cajero automático, internet, teléfono}, tipo de moneda*). Relación utilizada para almacenar todos los depósitos realizados en el banco, considerando la fecha y hora cuando el depósito fue realizado, el número de cuenta donde se abonó el depósito, el lugar en donde se llevó a cabo la operación y el tipo de moneda en que se realizó.
- **RETIRO** (*fecha de retiro, número de cuenta, cantidad, lugar {sucursal, cajero automático, internet, teléfono}, ubicación*). Contiene atributos relacionados a transacciones de retiro de dinero de las cuentas del banco. Sus atributos son la fecha de la transacción, el número de cuenta donde se aplica el retiro, la cantidad de dinero retirado, el lugar donde la transacción fue hecha, y la ubicación, es decir, si la transacción fue realizada en México o en el extranjero.
- **PAGO POR SERVICIOS** (*fecha, cuenta origen, cuenta destino, modo de acceso, cantidad, concepto del pago, ubicación*). Esta relación es utilizada para realizar pagos por servicios como el teléfono, internet, gas, entre otros. Los atributos para esta relación son la cuenta origen de donde se descontará el pago; la cuenta a donde se abonará el pago; el modo de acceso, es decir, la manera en que el cliente solicita este servicio, el cual puede ser realizado por internet o por teléfono; la cantidad de dinero que será pagada; el concepto del pago; y la ubicación de donde el cliente solicita el servicio.
- **IMPRESION DE COMPROBANTE** (*datos de la transacción*). Esta relación se utiliza para generar los comprobantes sobre las transacciones que realiza un cliente, y solamente contiene el mensaje que será impreso en el comprobante. Los comprobantes son impresos cuando un cliente realiza un depósito, un retiro, o un pago por servicio.
- **REPORTE** (*tipo de reporte, descripción, cliente, lugar, estado*). El atributo *tipo de reporte* almacena información sobre el reporte que se sometió al banco, es decir, si la tarjeta (ya sea de débito o crédito) de un cliente fue robada, está dañada, o la extravió; o si el reporte trata sobre una queja o reclamación al banco. En el atributo *descripción* se describe la razón del reporte. Además, en un reporte debe considerarse el nombre del cliente que hace el reporte; el lugar donde el cliente hizo el reporte, el cual puede ser sometido por internet, teléfono o en alguna sucursal del banco; y el estado del reporte, donde se especifica si el reporte no es

válido o no procede, si está en espera para ser revisado, si está en proceso de revisión, o si ya ha sido resuelto.

- **SOLICITUD DE TARJETA DE CREDITO** (*número de tarjeta de crédito, cuenta, con fotografía, tarjeta de crédito adicional*). Esta relación tiene como atributos el número de la tarjeta de crédito, el número de cuenta para esta tarjeta de crédito, si la tarjeta contendrá la fotografía del cliente, y si la tarjeta de crédito es una tarjeta adicional.

Para controlar algunas de las tareas realizadas por el banco, se desarrolló un conjunto de reglas ECA. Estas reglas pueden ser ejecutadas automáticamente en la BD, proporcionándole un comportamiento activo a la BD. El conjunto de reglas contiene un total de diecisiete reglas, las cuales se describen a continuación:

Regla 0: Cuando un depósito, un retiro o un pago por servicio es realizado, se imprime un comprobante para el cliente.

Regla 1: Cuando se realiza un retiro de alguna cuenta, si el saldo de la cuenta es mayor al monto solicitado para ser retirado, entonces se realiza la transacción y se sustrae la cantidad retirada del saldo de la cuenta.

Regla 2: Cuando un cliente realiza un pago por cierto servicio, si el saldo de la cuenta del cliente es suficiente para llevar a cabo el pago, entonces se retira la cantidad del pago de la cuenta del cliente.

Regla 3: Cuando un cliente realiza un depósito dentro del horario normal de atención al público, la cantidad por la que se hace el depósito se agrega al saldo de la cuenta del cliente.

Regla 4: Cuando un cliente realiza un depósito fuera del horario normal de atención al público, la transacción de pospone y será realizada al día siguiente.

Regla 5: Cuando un cliente realiza un pago por servicio, si está accedendo por Internet, entonces el banco descuenta \$25 de la cuenta del cliente como pago por el acceso al servicio de Internet.

Regla 6: El mismo caso que la regla anterior, pero ahora si el cliente realiza la transacción por teléfono el descuento es de \$20.

Regla 7: Cuando el cliente realiza un pago por servicio, la cantidad del pago se descuenta de la cuenta del cliente.

Regla 8: Cuando un cliente hace un retiro en un cajero automático y si el cliente ya hizo más de cinco retiros de cajeros automáticos, el banco cobra al cliente \$5 por cada retiro adicional que

haga, descontándolos directamente de la cuenta del cliente.

Regla 9: Cuando un cliente realiza un depósito, y si el tipo de moneda es diferente del peso mexicano y el depósito se realiza dentro del horario de atención al público, se convierte el tipo de moneda y se agrega a la cuenta del cliente.

Regla 10: Similar a la regla anterior, pero si la transacción es hecha fuera del horario laboral, entonces la transacción es pospuesta para el siguiente día hábil.

Regla 11: Cuando un cliente somete una reclamación, pero la queja es rechazada por falta de elementos, el banco descuenta \$200 por concepto de multa al cliente.

Regla 12: Cuando un cliente realiza un retiro en un cajero automático que no pertenece al banco donde el cliente tiene su cuenta, y si el retiro se realiza dentro del país, la comisión cobrada por el banco propietario del cajero automático es de \$10.

Regla 13: Similar a la regla anterior, pero en este caso si el retiro se realiza en el extranjero entonces la comisión que se cobra, por parte del banco propietario del cajero automático, es de \$80.

Regla 14: Cuando una tarjeta de crédito es solicitada por un cliente, si el cliente requiere su tarjeta de crédito con su fotografía incluida, entonces el costo de la tarjeta de crédito se incrementa en \$200.

Regla 15: Cuando una tarjeta de crédito es solicitada por un cliente, si el cliente requiere una tarjeta de crédito adicional, entonces el costo de la tarjeta de crédito se incrementa en \$100.

Regla 16: Cuando se registra un nuevo número de cuenta en el banco, si se trata de una cuenta de crédito, el cliente debe pagar \$350 por el servicio de crédito.

Para cada una de estas reglas, es necesario convertirlas a una forma de regla ECA, de acuerdo al modelo general para reglas ECA, quedando de la siguiente manera:

```
//Definicion de tablas
CLIENTE ( numCliente integer, nombre CHAR( length), domicilio CHAR( length), fechaAlta date ) ;
CUENTA ( numero integer, numCliente integer, saldo float, limite integer, tipo CHAR( length), estado
CHAR( length)) ;
PENDIENTES ( clave integer, tipo CHAR( length), cuentaOrigen integer, cuentaDestino integer, canti-
dad float, periodo integer ) ;
DEPOSITO ( fecha date, cuenta integer, cantidad float, lugar CHAR( length), moneda CHAR( length)
) ;
```

```

    RETIRO ( fecha date, cuenta integer, cantidad float, lugar CHAR( length) , ubicacion CHAR( length)
) ;
    PAGOPORSERVICIO ( fecha date, cuentaOrigen integer, cuentaDestino integer, modoAcceso CHAR(
length) , cantidad float, conceptoPago CHAR( length) , ubicacion CHAR( length) ) ;
    EXPIDECOMPROBANTE ( descripcion CHAR( length) ) ;
    REPORTE ( tipo CHAR( length) , descripcion CHAR( length) , cliente integer, lugar CHAR( length) ,
estado CHAR( length) ) ;
    EXPEDICIONTC ( numero integer, cuenta integer, conFotografia integer, adicional integer ) ;

//Definicion de eventos
e0 : insert_DEPOSITO ;
e1 : insert_RETIRO ;
e2 : insert_PAGOPORSERVICIO ;
e3 : insert_EXPIDECOMPROBANTE ;
e4 : insert_CUENTA;
e5 : or(e0:e1:e2) ;
e6 : update_REPORTE_estado;
e7 : insert_EXPEDICIONTC;

//Definicion de reglas
rule 000,
on e5,
if true,
Then insert into EXPIDECOMPROBANTE values ('expide comprobante') ; //

rule 001,
on e1,
if CUENTA.saldo > insert.cantidad,
Then update CUENTA set value saldo = saldo - RETIRO.cantidad where numero = insert.cuenta;

rule 002,
on e2,

```

```
if CUENTA.saldo > insert.cantidad,
Then insert into RETIRO values ( today, insert.cuentaOrigen, insert.cantidad, 'Sucursal', insert.ubicacion);

rule 003,
on e0,
if insert.fecha < '16:00:00',
Then update CUENTA set value saldo = old.saldo + insert.cantidad where numero = insert.cuenta;

rule 004,
on e0,
if DEPOSITO.fecha >= '16:00:00',
Then insert into PENDIENTES values (, 'deposito', 9999, insert.cuenta, insert.cantidad, 2 ) ;

rule 005,
on e2,
if insert.modosAcceso = 'Internet',
Then insert into RETIRO values (today, insert.cuentaOrigen, 25.00, 'INTERNET', insert.ubicacion ) ;

rule 006,
on e2,
if insert.modosAcceso = 'Telefono',
Then insert into RETIRO values (today, CUENTA.numero, 20.00, 'TELEFONO', insert.ubicacion ) ;

rule 007,
on e2,
if true,
Then insert into RETIRO values (today, insert.cuentaOrigen, insert.amount, 'SUCURSAL', insert.ubicacion
) ;

rule 008,
on e1,
if insert.lugar = 'Cajero automatico' & CUENTA.limite > 10,
```

Then insert into PAGOPORSERVICIO values (Today , insert.cuenta, NULL, NULL, 5.00, 'SERVICIOS_VARIOS' , 'Nacional');

rule 009,

on e0,

if insert.moneda <> 'PESO' & insert.fecha < '16:00:00',

Then update CUENTA set cantidad = old.cantidad - insert.cantidad + insert.cantidad * tipomoneda
where numero = insert.cuenta;

rule 010,

on e0,

if DEPOSITO.moneda <> 'PESO' & DEPOSITO.fecha >= '16:00:00',

Then insert into PENDIENTES values (1, 'deposito', null, insert.cuenta, insert.cantidad, 2));

rule 011,

on e6,

if REPORTE.tipo = 'Reclamacion' & update.estado = 'No procede',

then insert into PAGOPORSERVICIO values (today, NULL, CUENTA.numero, REPORTE.lugar, 200, 'Servicios Varios', 'Nacional');

rule 012,

on e1,

if insert.lugar = 'Cajero automatico externo' & RETIRO.ubicacion = 'Nacional',

Then insert into PAGOPORSERVICIO values (today, NULL, insert.cuenta, 'Automatico', 10, 'Servicios varios', 'Nacional');

rule 013,

on e1,

if insert.lugar = 'Cajero automatico externo' & insert.ubicacion = 'Extranjero',

Then insert into PAGOPORSERVICIO values (today, NULL, insert.cuenta, 'Automatico', 80, 'Servicios varios', 'Extranjero');

rule 014,

```

on e7,
if insert.conFotografia = 1,
  Then insert into PAGOPORSERVICIO values (today, NULL, insert.cuenta, 'Automatico', 200, 'Servicios
varios', 'Nacional');

```

```

rule 015,
on e7,
if insert.adicional = 1,
  Then insert into PAGOPORSERVICIO values (today, NULL, insert.cuenta, 'Automatico', 100, 'Servicios
varios', 'Nacional');

```

```

rule 016,
on e4,
if insert.tipo = 'Credito' ,
  Then insert into PAGOPORSERVICIO values ( today, insert.numero, NULL, 'Automatico' , 350.00,
'Servicios varios' , 'Sucursal' ) ;

```

La CCPN solo para la regla 001 se muestra en la figura 8.1. El lugar $p_0 \in P_{prim}$ representa al evento de la regla ECA, es decir, la inserción de elementos en la relación RETIRO está siendo monitoreado por p_0 . La transición $t_0 \in T_{regla}$ representa a la regla 001, y en ella se almacena la parte condicional de la regla ECA. Cuando el evento representado por p_0 ocurre, un token con información sobre el evento es generado y colocado en el lugar p_0 . La transición t_0 se habilita, recibe el token de p_0 y analiza la información contenida en él, si la condición de t_0 se satisface de acuerdo a la información del token entonces se genera un nuevo token con información sobre la acción de la regla, y se envía hacia el lugar de salida p_1 , el cual representa a la acción de la regla ECA.

La estructura de CCPN para la regla 000 se muestra en la figura 8.2. Los lugares p_0, p_1 y p_2 representan a los eventos primitivos agregar una tupla en la relación DEPOSITO, agregar una tupla a la relación RETIRO y agregar una tupla a la relación PAGOPORSERVICIO, respectivamente. Las transiciones t_0, t_1 y t_2 son transiciones de tipo *copy*, utilizadas para formar la estructura en CCPN para el evento compuesto *disyunción*. El lugar $p_3 \in P_{virtual}$ es un lugar virtual que representa al

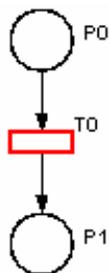


Figura 8.1: CCPN para la regla 001.

evento compuesto *disyunción*. La transición $t_3 \in T_{regla}$ y el lugar $p_4 \in P_{prim}$ representan a la regla y a la acción de la regla, respectivamente.

La CCPN que representa a todo el conjunto de reglas ECA para el banco se muestra en la figura 8.3.

Las correspondencias entre el conjunto de reglas ECA y los elementos de la CCPN es como sigue:

Eventos:

Evento	Lugar	Copy lugares
insert en <i>depósito, retiro, or PagoPorServicio</i>	p_5	
insert en <i>retiro</i>	p_1	$p_8, p_{17}, p_{21}, p_{22}$
insert en <i>PagoPorServicio</i>	p_2	$p_{10}, p_{14}, p_{15}, p_{16}$
insert en <i>depósito</i>	p_0	$p_{11}, p_{12}, p_{18}, p_{20}$
insert en <i>expedición Tarjeta de Crédito</i>	p_7	p_{23}, p_{24}
insert en <i>cuenta</i>	p_4	
update <i>reporte.estado</i>	p_6	

Acciones:

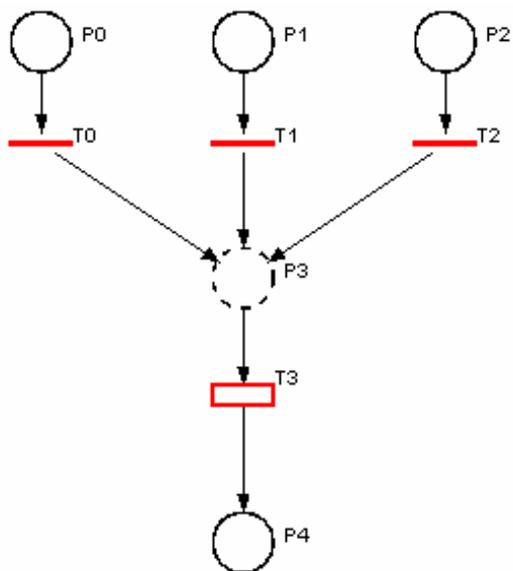


Figura 8.2: CCPN para la regla 000.

Acción	Lugar
insert en <i>reporte</i>	p_3
update <i>cuenta.saldo</i>	p_9
insert en <i>pendientes</i>	p_{13}
update <i>cuenta.cantidad</i>	p_{19}

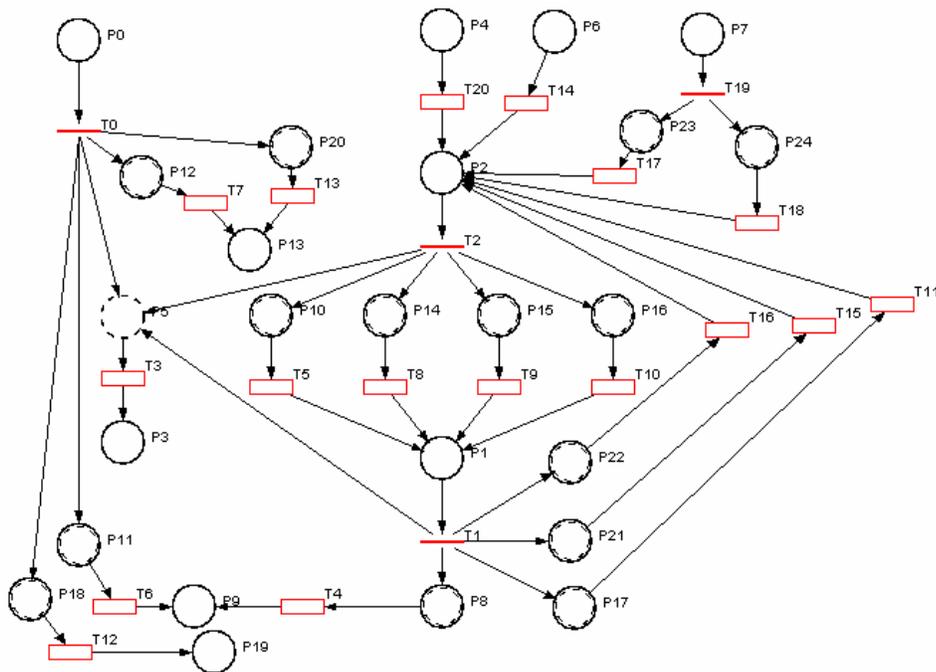


Figura 8.3: CCPN para todo el conjunto de reglas ECA.

Y finalmente, las reglas *ECA*:

Regla	Evento (lugar ent)	Transición	Action (lugar sal)
000	p_5	t_3	p_3
001	p_8	t_4	p_9
002	p_{10}	t_5	p_1
003	p_{11}	t_6	p_9
004	p_{12}	t_7	p_{13}
005	p_{14}	t_8	p_1
006	p_{15}	t_9	p_1
007	p_{16}	t_{10}	p_1
008	p_{17}	t_{11}	p_2
009	p_{18}	t_{12}	p_{19}
010	p_{20}	t_{13}	p_{13}
011	p_6	t_{14}	p_2
012	p_{21}	t_{15}	p_2
013	p_{22}	t_{16}	p_2
014	p_{23}	t_{17}	p_2
015	p_{24}	t_{18}	p_2
016	p_4	t_{20}	p_2

La regla 000 tiene como su evento activador al evento compuesto *disyunción*, el cual está formado por los eventos primitivos *insert_depósito*, *insert_retiro*, *insert_PagoPorServicio*, representados por los lugares p_0 , p_1 y p_2 , respectivamente.

Los lugares p_0 , p_1 y p_2 son utilizados más de una vez, por lo tanto lugares $p \in P_{copy}$ son creados para duplicar los tokens alusivos a éstos eventos.

Capítulo 9

Conclusiones y trabajo futuro

En esta tesis de investigación doctoral se propone un marco teórico como fundamento para el desarrollo de reglas ECA, basado en un modelo formal para su representación, simulación, análisis y ejecución, basado en teoría de PN, denominado CCPN. Las reglas ECA se utilizan para el desarrollo de sistemas de bases de datos activas. Además de la fortaleza de representación gráfica que se tiene en la teoría de PN, se aprovecha también su fortaleza de representación matemática para formalizar el marco teórico propuesto.

El evento de la regla ECA puede ser primitivo o compuesto, la CCPN soporta la definición de eventos primitivos y compuestos. Los eventos primitivos que soporta la CCPN son los relacionados con operaciones sobre la estructura de la BD, eventos de tiempo y eventos externos; por otro lado, los eventos compuestos que soporta son la conjunción, disyunción, negación, secuencia, simultáneo, historia, primero, último y alguno; para los cuales se define un constructor de eventos compuestos basado en la CCPN.

En la CCPN se realiza el análisis de reglas ECA para determinar problemas como el de No terminación y confluencia. En ambos casos se utiliza la matriz de incidencia de la teoría de PN, aplicada en la CCPN. En la matriz de incidencia se buscan las rutas presentes dentro de la CCPN para encontrar ciclos dentro de la CCPN y nodos confluentes.

Para mostrar la aplicación de la investigación, se desarrolló una interfaz, la cual utiliza a la CCPN como medio de representación de reglas ECA, y proporciona comportamiento activo a una base de datos relacional. Las pruebas realizadas fueron sobre los SMBD Postgresql, MS Access,

Oracle, y Visual Fox Pro. Estas conexiones se realizaron mediante el controlador JDBC-ODBC para establecer la comunicación entre ECAPNSim y el SMBD.

Sin embargo, se observó que un problema que se presenta dentro de las PN's es heredado a la CCPN, en lo que respecta al tamaño de la PN. En el caso de la CCPN, entre mayor número de reglas ECA se estén desarrollando, el tamaño de la CCPN también crece.

A continuación se muestra una tabla comparativa entre los sistemas de BDA estudiados y el modelo propuesto de CCPN.

Sistema	Eventos compuestos	Análisis estático	Independencia de plataforma	Simulación
SAMOS	Disyunción, Conjunción, Secuencia, Primero, Historia, Negación.	Si	No	Si
Ariel	No	No	No	No
Postgres	No	No	No	No
Starburst	No	Si	No	No
A-RDL	Secuencia, selección, repetición,	No	No	Si
Ode	Historia, conjunción, negación, relativo, secuencia, alguno, prioridad.	No	No	No

Tabla comparativa.

Sistema	Eventos compuestos	Análisis estático	Independencia de plataforma	Simulación
Snoop	Disyunción, conjunción, secuencia, negación.	No	No	No
EPL	Secuencia, conjunción, disyunción, negación.	No	No	No
ECAPNSim -CCPN	Disyunción, Conjunción, Secuencia, Simultáneo, Negación, Primero, Último, Historia, Alguno, De reloj.	Si	Si	Si

Tabla comparativa (continuación).

Como trabajo futuro se pretende aplicar la CCPN hacia sistemas de bases de datos orientadas a objetos (SBDOO). Teóricamente no hay mucha diferencia en los eventos considerados de los sistemas de bases de datos relacionales con los eventos que ocurren dentro de un SBDOO. Sin embargo, también se adaptará ECAPNSim para poder dar comportamiento activo a un SBDOO.

Se pretende crear el árbol de cobertura de la CCPN obtenida de bases de reglas ECA, con la finalidad de llevar a cabo análisis sobre la secuencia de ejecución de reglas. Con este análisis también detectaremos la presencia de un ciclo en la CCPN, además de saber si una posible marca en la CCPN es alcanzable desde un estado determinado, lo cual es importante para precisar si un disparo infinito de reglas puede concluir o no.

Un área poco explorada es la *adaptabilidad de las reglas ECA*, donde solamente se han propuesto modalidades de adaptabilidad en la activación y desactivación de reglas, pero la estructura de la base de reglas ECA no cambia. Se plantea que la CCPN puede tener un alto grado de adaptabilidad, si se permiten acciones que modifiquen la propia CCPN.

El modo de acoplamiento de reglas ECA, en el modelo de ejecución, que soporta CCPN son el modo inmediato, pero se analizará la factibilidad de trabajar con transacciones e implementar los modos *pospuesto y separado*.

Además, se pretende estudiar el comportamiento de la CCPN en sistemas de BD distribuídas. En vista de que ECA-PNSim se coloca como una capa superior a la BD y a la independencia de plataforma, se podría utilizar como una capa distribuída sobre las diferentes BD.

A partir de esta línea de investigación se pueden generar diversas vertientes. Dos de las líneas de investigación inmediatas al trabajo son, por un lado, el análisis de complejidad de reglas ECA. En este análisis se toma una base de reglas ECA representada mediante una CCPN, y con la información generada en la matriz de incidencia se puede determinar la complejidad de la base de reglas en base a métricas establecidas. [75]

Por otro lado se tiene la aplicación de la CCPN en sistemas de Workflow. Un workflow es un sistema reactivo, cuyo comportamiento puede ser descrito utilizando reglas ECA. Son pocos los trabajos que han utilizado reglas ECA para modelar o ejecutar procesos workflow, algunos trabajos utilizan este tipo de reglas para manejar las excepciones en un workflow. A partir del marco teórico establecido en esta tesis doctoral, es posible utilizar el modelo de CCPN para describir el proceso de workflow como reglas ECA.

Un sistema de workflow puede ser representado mediante reglas ECA, por lo que la CCPN es capaz de modelar sistemas de Workflow. Sin embargo es necesario ajustar el modelo para considerar las características propias de Workflow. Con estos ajustes es posible modelar y simular procesos de Workflow a partir de el modelo extendido de CCPN, con la cual es posible modelar la perspectiva de flujo de control sin necesidad de abstraer datos de control

Otra línea de investigación es la aplicabilidad de la CCPN para ser utilizado como un lenguaje gráfico de definición de reglas ECA. Actualmente, la definición de reglas ECA en los SBA existentes se realiza de manera textual siguiendo una sintaxis determinada, pero con el uso de la CCPN es posible definir directamente las reglas ECA tomando los elementos de la CCPN. De tal forma que

puede desarrollarse una base de reglas ECA en un entorno interactivo usuario-ECAPNSim para seleccionar los elementos de la CCPN que serán utilizados en la definición de las reglas ECA.

Publicaciones

1. Xiaou Li, **Joselito Medina**, Sergio Chapa, “Applying Petri Nets on Active Database Systems”, IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews, 2005, aceptado.
2. Xiaou Li, **Joselito Medina Marín**, and Sergio V. Chapa, A Structural Model of ECA Rules in Active Database, Mexican International Conference on Artificial Intelligence (MICAI’02), LNCS Lecture Notes in Artificial Intelligence Vol. 2313. Mérida, Yucatan, México, April 22-26, 2002, pp. 486-493
3. **Joselito Medina Marín**, Xiaou Li, Termination Analysis in Active Databases: a Petri Net Approach, ISRA2004, Querétaro, Mexico.
4. Xiaou Li, Sergio Chapa, **Joselito Medina Marín**, and Jovita Martínez, An Application of Conditional Colored Petri Nets: Active Database System, 2004 IEEE International Conference on Systems, Man & Cybernetics (SMC2004), 4885-4890, The Hague, Neitherland, 10-13, oct, 2004
5. Xiaou Li, **Joselito Medina Marín**, Composite Event Specification in Active Database Systems: A Petri Nets Approach, the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 97-116, Aarhus, Denmark, October 8-11, 2004
6. **Joselito Medina Marín** and Xiaou Li, “An Active Rule Base Simulator Based on Petri Nets”, International Workshop on Modelling, Simulation, Verification, and Validation of Enterprise Information Systems, MSVVEIS 2005, Miami, Florida, USA, May 24-28, 2005.

7. **Joselito Medina Marín** and Xiaou Li, "ECA rule development via Conditional Colored Petri Net", Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management,, PNCWB 2005, Miami, Florida, USA, Jun 20-25, 2005.
8. **Joselito Medina Marín** y Xiaou Li, Red de Petri Coloreada Condicional y su Aplicación en Sistemas de Bases de Datos Activas, VIII Conferencia de Ingeniería Eléctrica (CIE'02), México D.F., 2002, pp. 238-245
9. **Joselito Medina Marín** y Xiaou Li, ECAPNSim, un simulador para reglas ECA, el XIII Congreso Interuniversitario de Electrónica, Computación y Eléctrica (CIECE), Zacatepec, México, Abril, 9-11, 2003
10. **Joselito Medina Marín** y Xiaou Li, Análisis de terminación de reglas, un enfoque con red de Petri, el 4º Congreso Nacional de Computación (CORE 2003), México, D.F., México, Mayo 6-7, 2003, pp.56-67
11. Lorena Chavarría Báez, **Joselito Medina Marín**, Xiaou Li, "Measuring triggering-interactions complexity on active databases based on conditional colored Petri net model", International Conference on Electrical and Electronics Engineering and X conference on Electrical Engineering (ICEEE/CIE), Acapulco, México, Sept. 8-10, 2004
12. Lorena Chavarría-Báez, **Joselito Medina-Marín**, Xiaou Li, Sergio V. Chapa, Análisis de la complejidad de las interacciones de los disparos en bases de datos activas vía Red de Petri Coloreada Condicional, Cuernavaca, Mexico.

Bibliografía

- [1] Date C.J., “*An Introduction to Database Systems*”, Addison-Wesley, Sixth Edition, System Programming Series, Reading, (MA) 1995.
- [2] Elsmari R., Navathe S. B., “*Sistemas de Bases de Datos, Conceptos fundamentales*”. Segunda Edición. Editorial Addison-Wesley. ISBN 0-201-65370-2.
- [3] Ye-In-Chan, Fwo-Long-Chen, “RBE: a rule-by-example active database system”, *Software - Practice and experience*, Vol. 27, No. 4; April 1997; pp. 365-394
- [4] <http://www.cse.ohio-state.edu/~gurari/course/cis670/sybase-only.html>
- [5] Paton N.W., Diaz O., “Active Database Systems”, *ACM Computing Surveys*, Vol. 31, No. 1, 1999, pp. 64-103
- [6] McGoveran D., Date. C.J., “*A guide to SYBASE and SQL Server : a user’s guide to the SYBASE product*”, Sybase, Inc, 1992.
- [7] Lacy-Thompson T., “*INFORMIX-SQL, A tutorial and reference*”, ISBN-0-13-465121-9, Ed. Prentice Hall, 1990.
- [8] Hursh C.J., Hursch J.L., “*Oracle SQL Developer’s Guide*”, ISBN-0-8306-2529-1, Ed. McGraw-Hill, 1991.
- [9] González-Pérez A., “*SQL Server, Programación y administración*”, ISBN 970-15-0376-7, Ed. Alfaomega ra-ma, 1999.

- [10] Hanson E.N., "*The Design and Implementación of the Ariel Active Database Rule System*", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 1, february 1996.
- [11] <http://dns1.mor.itesm.mx/~emorales/Cursos/RdeC/node70.html>
- [12] http://www.monografias.com/trabajos/iartificial/pagina6_5.html
- [13] Dayal U., Blaustein B., Buchmann A., Chakravarthy U., Hsu M., Ledin R., McCarthy D., Rosenthal A., Sarin S., Cary M.J., Livny M. and Jauhari R., "The HiPAC Project: combining active database and timing constraints", SIGMOD Record, 17(1), pp. 51-70, 1998.
- [14] Widom J., "*The Starburst Active Database Rule System*", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 4, August 1996.
- [15] Stonebraker M., Kemmintz G., "*The POSTGRES Next-Generation Database Management System*", Communications of the ACM, Vol. 34, No. 10, October 1991.
- [16] www.ca.postgresql.org
- [17] Potamianos S., Stonebraker M., "*The POSTGRES Rule System*", Active Database Systems: Triggers and Rules for Advanced Database Processing, Ed. Jennifer Widom and Stefano Ceri. 1996, pages 43-61.
- [18] Hanson E.N., "*The Ariel Project*", Active Database Systems: Triggers and Rules for Advanced Database Processing, Ed. Jennifer Widom and Stefano Ceri. 1996, pages 63-86.
- [19] Miranker D.P., "*TREAT: A better match algorithm for AI production systems*", in Proceedings of the AAAI Conference on Artificial Intelligence, pages 42-47, August 1987.
- [20] Widom J., "*The Starburst Rule System*", Active Database Systems: Triggers and Rules for Advanced Database Processing, Ed. Jennifer Widom and Stefano Ceri. 1996, pages 87-109.
- [21] Simon E., Kiernan J., "*The A-RDL System*", Active Database Systems: Triggers and Rules for Advanced Database Processing, Ed. Jennifer Widom and Stefano Ceri. 1996, pages 111-149.
- [22] Ceri S., Fraternali P., Paraboschi S., and Tanca L., "*Active Rule Management in Chimera*", Active Database Systems: Triggers and Rules for Advanced Database Processing, Ed. Jennifer Widom and Stefano Ceri. 1996, pages 151-176.

- [23] Dayal U., Buchmann A.P., Chakravarthy S., "*The HiPAC Project*", Active Database Systems: Triggers and Rules for Advanced Database Processing, Ed. Jennifer Widom and Stefano Ceri. 1996, pages 177-206.
- [24] Gehani N., Jagadish H.V., "*Active Database Facilities in Ode*", Active Database Systems: Triggers and Rules for Advanced Database Processing, Ed. Jennifer Widom and Stefano Ceri. 1996, pages 207-232.
- [25] Chakravarthy S., "Early Active Database Efforts: A Capsule Summary", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 6, 1995, pp. 1008-1010
- [26] Silberschatz A., Korth H.F., Sudarshan S., "*Database System Concepts*", Third Edition, McGraw-Hill, 1999.
- [27] Dittrich K., Gatzju S., Geppert A., "*The Active Database Management System Manifesto: A Rulebase of ADBMS Features*". A Join Report by the ACT-NET Consortium.
- [28] Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K., "*Composite events for active databases: Semantics, contexts and detection*". In *Proceedings of the Twentieth International conference on Very Large data Bases*, J. Bocca, M. Jarke, and C. Zaniolo, Eds., Morgan-Kaufmann, San Mateo California, 1994, pp. 606-617.
- [29] Zhou M., Venkatesh K., "*Modeling, Smulation, and Control of Flexible Manufacturing Systems. A Petri Net Approach*". Series in Intelligent Control and Intelligent Automation, Vol. 6, World Scientific. 1998.
- [30] Peterson J.L., "*Petri Net Theory and The Modeling of Systems*", Prentice-Hall, Inc., 1981. ISBN 0-13-661983-5.
- [31] Wang J., "*Timed Petri Nets. Theory and Applications*", Kluwer Academic Publishers, 1998.
- [32] Murata T., "*Petri Nets: Properties, analysis, and applications*", *Proceedings of the IEEE*, 77(4):541-580, 1989.
- [33] David R., Alla H., "*Petri Nets for Modeling of dynamic Systems - A Survey*", *Automatica* Vol. 30, No. 2, pp. 175-202. 1994.

- [34] Li X., Yu W, Lara-Rosano F., "*Dynamic Knowledge Inference and Learning under Adaptive Fuzzy Petri Net Framework*", IEEE Transactions on systems, Man, and Cybernetics-Part C: Applications and reviews, Vol. 30, No. 4, November 2000.
- [35] Champagat R., Esteban P., Pingaud H., ValetteR., "*Petri net based modeling of hybrid systems*", Computers in Industry 1998, No. 36, pp. 139-146.
- [36] Cassandras C., Lafortune S., "*Introduction to Discrete Event Systems*", Kluwer Academic Publishers, ISBN 0-7923-8609-4, 1999.
- [37] Andreu D., Pascal J.C., Pingaud H., Valette R., "*Batch Process Modelling Using Petri Nets*", IEEE-SMC, San Antonio, USA, October 1995, pp. 314-319.
- [38] Gatziu E., Ditrich K. R., "*SAMOS*", Active Rules in Database Systems, Norman W. Paton, Editor. 1999, pp. 233-248.
- [39] Gatziu E., Ditrich K. R., "*Detecting Composite Events in Active Database Systems Using Petri Nets*", Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, Texas, February 1994.
- [40] L. Warshaw and L. Obermeyer and D. Miranker and S. Matzner, "*VenusIDS: An Active Database Component for Intrusion Detection*", Applied Research Laboratories, University of Texas, Austin, 1999, url = citeseer.ist.psu.edu/warshaw99venusids.html.
- [41] A. Aiken, J. M. Hellerstein, and J. Widom, "*Static analysis techniques for predicting the behavior of active database rules*", *ACM Transactions on Database Systems*, 20(1), 1995, pp. 3-41
- [42] Baralis E. and Widom J., "An Algebraic Approach to Static Analysis of Active Database Rules", *ACM Transactions on Database Systems*, Vol. 25, No. 3, September 2000, pp. 269-332.
- [43] Karadimce A.P., and Urban S.D., "*Diagnosing anomalous rule behavior in database with integrity maintenance production rules*", In Third Workshop on Foundations of Models and Languages for Data and Objects, (Aigen, Austria, Sept. 1991), pp. 77-102.

- [44] Karadimce A.P., and Urban S.D., "*Conditional term rewriting as a formal basis for analysis of active database rules*", In Fourth Workshop on Research Issues in Data Engineering (RIDE-ADS '94), (Houston, Tex., Feb. 1994), pp. 156-162.
- [45] Krishnamurthy R., Ramakrishnan R., and Shmueli O., "*Framework for testing safety and effective computability of extended Datalog*", in Proceedings of the ACM SIGMOD International Conference on Management of Data. (Chicago, Ill., June 1988), ACM, New York, pp. 154-164.
- [46] Sagiv Y., and Vardi M., "*Safety of Datalog over infinite databases*", in the Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. ACM, New York 1989, pp. 160-170.
- [47] Fraternali P., Paraboschi S., and Tanca L., "*Automatic Rule Generation for Constraint Enforcement in Active Databases*", In U.W. Lipeck and B. Thalheim (Eds.), Modelling Database Dynamics. London: Springer-Verlag, 1993. Selected Papers from the Fourth International Workshop Foundations of Models and Languages for Data and Objects, pp. 153-173.
- [48] Urban S.D., Tschudi M.K., Dietrich S.w. and Karadimce A.P., "Active Rule Termination Analysis: An Implementation and Evaluation of the Refined Triggering Graph Method", *Journal of Intelligent Information Systems*, No. 12, 1999, pp. 27-60.
- [49] Kokkinaki A.I., "*On using Multiple Abstraction Models to Analyze Active Databases Behavior*", Biennial World Conference on Integrated Design and Process Technology, Berlin, Germany, June, 1998.
- [50] Guisheng Y., Qun L., Jianpei Z., Jie L., Daxin L., "Petri Based Analysis Method For Active Database Rules", *IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, 1996, pp. 858-863
- [51] Schlesinger M. and Lörincze G. , "*Rule modeling and simulation in ALFRED*", the 3rd International workshop on Rules in Database (RIDS'97) (or LNCS 1312), Skövde, Sweden, June, pp. 83-99, 1997
- [52] Zimmer D., Unland R., and Meckenstock A., "*Rule Termination Analysis based on a Rule Meta Model*", 1995

- [53] Zimmer D., Unland R., and Meckenstock A., "*Rule Termination Analysis based on a Petri Nets*", 1996
- [54] Zimmer D., Unland R., and Meckenstock A., "*Using Rule Petri Nets for Rule Termination Analysis*", 1997
- [55] Lorena Chavarría Báez, Joselito Medina Marín, Xiaou Li, "*Measuring triggering-interactions complexity on active databases based on conditional colored Petri net model*", International Conference on Electrical and Electronics Engineering and X conference on Electrical Engineering (ICEEE/CIE), Acapulco, México, Sept. 8-10, 2004
- [56] Lorena Chavarría-Báez, Joselito Medina-Marín, Xiaou Li, Sergio V. Chapa, "*Análisis de la complejidad de las interacciones de los disparos en bases de datos activas vía Red de Petri Coloreada Condicional*", Cuernavaca, Mexico, 2004.
- [57] Li X. and Chapa S. V., Optimization of Deductive Database System by Adaptive Fuzzy Petri Net Approach, *IASTED International Conference on Artificial Intelligence and Soft Computing (ASC2001)*, Cancun, Mexico, May 21-24, 2001, pp. 6-11
- [58] Barkaoui K. and Maïzi Y., Efficient answer extraction of deductive database modeled by HLPN, *8th International Conference on Database and expert systems applications (DEXA '97)*, Toulouse, France, September 1-5, 1997 (also LNCS vol. 1308), pp. 324-336
- [59] Li X., Medina Marín J., and Chapa S. V., "A Structural Model of ECA Rules in Active Database", *Mexican International Conference on Artificial Intelligence (MICAI'02)*, Mérida, Yucatan, México, April 22-26, 2002
- [60] Baralis E. Ceri E., Fraternali P., and Paraboschi S., "Support Environment for Active Rule Design", *Journal of Intelligent Information Systems*, No. 7, 1996, pp. 129-149.
- [61] Chen H. and Hanisch H., "Analysis of hybrid systems on hybrid net condition/event system model", *Discrete Event Systems: Theory and Applications*, vol.11, 163-185, 2001
- [62] Chen S.M., Ke J.S., and Chang J.F., "*Knowledge Representation Using Fuzzy Petri Nets*", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 3, September 1990.

- [63] Cheng P., and Forward K., " *Fuzzy Petri Nets*", First International Conference on Knowledge-Based Intelligent Electronic Systems, 21-23 May 1997, Adelaide, Australia. Editor L. C. Jain.
- [64] Tang R., Pang G.K.H., and Woo S.S., " *A Continuous Fuzzy Petri Net Tool for Intelligent Process Monitoring and Control*", IEEE Transactions on Control Systems Technology, Vol. 3, No. 3, September 1995
- [65] Hanna M.M., Buck A., and Smith R., " *Fuzzy Petri Nets with Neural Networks to Model Products Quality from a CNC-Milling Machining Centre*", IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 26, No. 5, September 1996.
- [66] Cao T., and Sanderson A.C., " *A Fuzzy Petri Net Approach to Reasoning about Uncertainty in Robotic Systems*", Robotics and Automation, 1993, Proceedings, 1993 IEEE International Conference on Published: 1993, pp. 317-322 vol. 1.
- [67] Tazaki E., and Yoshida K., " *A Fuzzy Petri Net Model for Approximate Reasoning and its Application to Medical Diagnosis*", Systems, Man and Cybernetics, 1992, IEEE International Conference, 1997. COMPSAC'97. Proceedings, The Twenty-First Annual International, Published: 1997, pp. 438-443.
- [68] Jensen K., " *An Introduction to the Theoretical Aspects of Colored Petri Nets*". *Lecture Notes in Computer Science: A Decade of Concurrency*, vol. 803, edited by J. W. de Bakker, W.-P. de Roever, G. Rozenberg , Springer-Verlag, 1994, pp. 230-272
- [69] Etzion O., " *An Alternative Paradigm for Active Databases*", Research Issues in Data Engineering 1994, Active Database Systems. Proceedings Fourth International Workshop on Published 1994, pp. 39-45.
- [70] Zimmer D., Unland R., Meckenstock A., " *Rule Termination Analysis based on Petri Nets*", Cadlab Report 3, Cadlab, Fustenallee 11, 33102 Paderborn, Germany, February 1996.
- [71] www-db.research.bell-labs.com/project/ode/index.html
- [72] www.daimi.au.dk/PetriNets/tools/quick.html

- [73] Hasan M., "The Management of Data, Events, and Information Presentation for Network Management", Doctoral thesis, University of Waterloo, Waterloo, Ontario, Canada, 1996.
- [74] Eckel B., Thinking in Java, 2nd. Edition, Prentice Hall, June 2000.
- [75] Chavarria Baez L., Medición de la complejidad de la interacción de las reglas ECA en BDA vía CCPN, Tesis de Maestría, 2004.