Índice general

Agradecimientos

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

En especial a mi esposa, quien me motivó desde que vio surgir en mí la inquietud de superación personal y académica, por su apoyo en los momentos difíciles y amor incondicional.

A mis hijos, que aún sin saber el sacrificio que estoy haciendo siempre están conmigo.

A mis padres, por haberme inculcado los deseos de salir adelante, el de apoyar a gente que necesita de mi ayuda y a no rendirme fácilmente.

A mis hermanos Frank, Maritza, Marina y Marely, porque entre ellos aprendí las cosas de la vida, a valerme por mí mismo y hacerme responsable de mis actos.

A mis dos grandes familias, Medina y Marín, por estar siempre conmigo y yo estaré siempre con ellos.

A mi asesora la Dra. Xiaoou Li, por sus brillantes ideas para el desarrollo de éste trabajo de tesis, por su amabilidad al atenderme cuando existían en mí dudas sobre la investigación y por su generosidad al darme la oportunidad de desenvolverme en otros ámbitos. 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. A todos mis maestros que compartieron conmigo sus conocimientos en sus respectivas áreas. Y a las secretarias de la Sección de computación, porque sin ellas no realizaríamos trámite administrativo alguno.

A la generación apática 2000 de la Sección de Computación, por los debates que mantuvimos para resolver tareas y por la jornadas de guasa que nos aventamos para olvidar la presión de los cursos.

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

Resumen

Las bases de datos activas (BDA) son extensiones de bases de datos (BD) tradicionales, las cuales además de tener un comportamiento pasivo (modificar ú obtener datos solicitados por el usuario) reaccionan ante la presencia de uno ó más eventos en la BD. El comportamiento activo de una BD se modela con las reglas Evento-Acción-Condición (reglas ECA). La mayoría de las BDA comerciales utilizan el enfoque de las reglas ECA y ofrecen su propio lenguaje de definición de reglas ECA. Sin embargo, los administradores de BDA no pueden llevar a cabo una simulación la simulación del comportamiento de las reglas ECA antes de implementarlas en la BDA. Además, la sintaxis de definición varía entre un sistema y otro, y no existe una manera general adecuada para todos los sistemas. La red de Petri (PN) es una herramienta de modelación y simulación gráfica y matemática, la cual puede extenderse fácilmente con conceptos de PN orientadas a objetos, híbridas, coloreadas, etc. Además, puede utilizar en un rango amplio de aplicaciones. Las BDA son un área nueva y prometedora de la aplicación de PN.

En esta tesis se propone, una PN extendida, la Red de Petri Coloreada Condicional (CCPN). Los elementos de la regla ECA (evento, condición y acción) pueden modelarse como lugares y transiciones de una CCPN. Entonces, la base de reglas ECA se modela con una CCPN y la ejecución de la CCPN simula el comportamiento de la BDA.

Además, se desarrolló una interfaz gráfica basada en el concepto de CCPN. Bajo el ambiente de ECAPNSim, automáticamente se genera una CCPN a partir de un conjunto de reglas ECA, además, el comportamiento de la base de reglas ECA es simulado con la CCPN obtenida.

ECAPNSim se conecta fácilmente a una BD para detectar eventos y ejecutar acciones especificadas en la base de reglas. Tres ejemplos pequeños de bases de reglas ECA se tomaron de la literatura para demostrar la factibilidad del modelo CCPN y del software ECAPNSim.

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 the event-condition-action (ECA) rules. Most of the commercial ADB use ECA rule approach and they provide to their own syntax of active rule language. Nevertheless, ADB administrators cannot carry out a simulation of the ECA rules behavior before its implementation in ADB. Also the syntaxes vary with databases, there doesn't exist a general one which suitable for any database. Petri net (PN) is a graphical and mathematical tool of modeling and simulation, it can be extended easily with many concepts such as object oriented, hybrid, colored, etc. On the other hand, it can be used in a variety of application field; Active database is a new and promising application area of PN.

In this thesis, an extended PN is proposed, named Conditional Colored Petri Net (CCPN). ECA rule elements event, condition and action can be modeled as places and transitions in a CCPN. Then, an ECA rule base may be modeled as a CCPN model, and the execution of the CCPN simulates the active behavior of the active database.

Furthermore, we develop a graphical interface ECAPNSim based on CCPN concept. Under ECAPNSim environment, a CCPN model can be automatically generated from an ECA rules set, therefore the ECA rule base behavior will be simulated with this produced CCPN model.

Additionally, ECAPNSim can be easily connected to a database to detect events and execute the actions specified in the rule base. For case study, Postgres database is adopted as a passive database to connect with ECAPNSim. Three small ECA rule bases from literatures are illustrated to demonstrate the feasibility of CCPN model and ECAPNSim software.

Capítulo 1

Introducción

En las primeras generaciones de bases de datos ha permanecido pasivo y solamente responden a las acciones que los usuarios requieren explícitamente [1]. Una nueva generación denominada base de datos activas, tiene como objetivo realizar acciones automáticas mediante la ocurrencia de ciertos eventos dentro de la base de datos. La especificación de las acciones que se disparan a través de la satisfacción de condiciones se lleva a cabo con reglas activas, cuyo modelo se denomina evento-condición-acción con su forma general siguiente [2]:

on `evento`

if `condición`

then `acción`

Un *evento* es un suceso - tal como inserción, actualización y borrado - el cual cambia el estado de la base de datos. El concepto contiene de forma intrínseca una dinámica de la base de datos la cual cambiará el estado de acuerdo a que los eventos satisfagan ciertas condiciones. Cuando ocurre un evento, de forma automática se activan una o más reglas, o bien, se preparan para ser disparadas. Dicho evento se llama activador de las reglas. Una vez que una regla se activa, las *condiciones* de la regla son instanciadas. Ejemplos de algunas condiciones que pueden aparecer en una base de reglas activas son: verificar si los niveles de inventario en algún almacén están por debajo del mínimo considerado o si la actualización de una tupla llevó a la BD a un estado inconsistente. Si la condición se cumple, las *acciones* de la regla se llevan a cabo.

Ejemplos de acciones son hacer pedidos de productos si la cantidad inventariada de estos pro-

ductos ha caído por debajo del mínimo o la cancelación de la transacción (si este movimiento produjo un estado inconsistente de la BD). Este tipo de reglas pueden usarse para propósitos muy diversos. Un ejemplo de aplicación es una alarma ó una alerta, donde las reglas monitorean el sistema y notifican al administrador o al usuario si un evento fuera de lo común ha ocurrido. También pueden usarse para mantener la integridad de los datos. Otro ejemplo de su uso se aplica en el mantenimiento de datos derivados, tales como índices y consultas materializadas. Si una consulta ha sido materializada (es decir, procesada y almacenada), necesita actualizarse si las relaciones de la BD en la cual se definió ha sufrido cambios. Para que las acciones mantengan actualizadas las consultas a la BD se necesita que éstas consultas sean codificadas como reglas activas. La sintaxis para su definición aún no se encuentra estandarizada y puede ser diferente entre un sistema y otro.

La implementación de comportamiento activo a las BD no es algo particularmente nuevo. La mayoría de los sistemas de BD comerciales incluyen mecanismos de activación de reglas denominados "triggers". Además, un sinnúmero de prototipos de investigación se han desarrollado con la finalidad de que la implantación de bases de reglas activas se realice de manera más sencilla. Las propuestas de sistemas activos [3] [4] [5] [6] [7] para BD relacionales, a menudo poseen características similares, las cuales se originan a partir de la naturaleza de la BD pasiva en la que se encuentra la base de reglas activas. Debido a que los lenguajes de consulta de las BD relacionales, como SQL, proporcionan facilidades para expresar condiciones y para realizar acciones en la BD, el lenguaje de definición de reglas regularmente es una extensión del lenguaje de consulta. En general, los mecanismos de ejecución de reglas propuestos por las BD relacionales solamente soportan uno o un rango limitado de modos de acoplamiento de reglas, y además tienen ciertas limitaciones para la detección de eventos compuestos.[2]

1.1. Antecedentes generales

Los sistemas de bases de datos activos han sido desarrollados para aplicaciones que necesitan de reacciones automáticas en respuesta a la ocurrencia de ciertas condiciones o a la ocurrencia de ciertos eventos. Este tipo de comportamiento es expresado mediante la definición de reglas ECA (evento-condición-acción). Generalmente, las reglas ECA y su ejecución están representadas por un lenguaje de reglas, por ejemplo, definiendo TRIGGERs en base de datos activas (como Chimera, SAMOS, OSAM, Starburst, Postgress, etc.). Además, la predicción o el análisis del comportamiento

de una base de datos se puede realizar a través de diferentes enfoques, tal como enfoques algebraicos, métodos que grafican el disparo de reglas, entre otros. Sin embargo, en tales bases de datos activas la representación de las reglas y el procesamiento de las mismas se encuentran definidos en módulos separados.

Un sistema de base de datos activa combina el procesamiento de reglas, basado en eventos, con la funcionalidad de las bases de datos tradicionales. Cuando hay cambios en la base de datos un evento ocurre, entonces una ó más reglas pueden ser disparadas. Una vez que una regla es disparada, las condiciones de esta regla son verificadas y si las condiciones se satisfacen, entonces las acciones de la regla se llevan a cabo.

La sintaxis del procesamiento de reglas activas incluye dos modelos: el modelo de conocimiento y el modelo de ejecución. El modelo de conocimiento indica qué puede ser dicho acerca de las reglas activas en ese sistema. El modelo de conocimiento esencialmente soporta la descripción de la funcionalidad activa, las características tratadas dentro de este modelo a menudo tienen una representación directa dentro de la sintaxis del lenguaje de reglas. El modelo de ejecución especifica cómo un conjunto de reglas es tratado en tiempo de ejecución.

No es sencillo monitorear la ejecución de las reglas ECA basadas en un lenguaje de reglas. Sin embargo, si un modelo formal es desarrollado para representar todas las reglas así como las relaciones que existen entre ellas, entonces el administrador de la ADBMS puede ser abstraído de operaciones o procesos basados en este modelo. Por consiguiente, la complejidad del manejo de una base de datos activa puede ser reducida considerablemente. Existe una herramienta de modelación y simulación muy poderosa denominada redes de Petri (PN), mediante la cual podría modelarse y simularse el comportamiento de una base de reglas activas.

Las redes de Petri son una herramienta gráfica y matemática para modelar sistemas concurrentes, asíncronos, distribuidos, paralelos, indeterministas y/o estocásticos [19]. Se utilizan para la modelación y simulación de sistemas manejados por eventos. De manera gráfica y visual se puede analizar el comportamiento del sistema modelado. Con las propiedades que presentan las redes de Petri se pueden detectar inconsistencias en los sistemas modelados. La teoría de redes de Petri ofrecen métodos para analizar las estructuras modeladas, como el árbol de alcanzabilidad, la matriz de incidencia y la ecuación de estado, el análisis de la invariancia de la red, reglas para reducir PN grandes y la simulación del sistema modelado. Las redes de Petri pueden extenderse para mode-

lar sistemas que no pueden ser modelados fácilmente por una red de Petri tradicional, existiendo actualmente diferentes extensiones de redes de Petri, diseñadas para sistemas específicos.

1.2. Planteamiento del problema

Las bases de datos tradicionales son consideradas pasivas y responden solamente a las acciones que los usuarios de manera explícita requieren. Sin embargo, actualmente se requiere de un mecanismo automático que active las reglas, mediante la ocurrencia de ciertos eventos dentro de la base de datos. En la base es necesario estructurar un conjunto de reglas activas las cuales especifican las acciones que se disparan cuando satisfacen ciertas condiciones. El modelo se denomina evento-condición-acción (ECA) que en su forma general es: on evento; if condición; then acción.

Los eventos son sucesos que propician cambios en los estados de la base de datos manteniendo una dinámica que cambia estados en función de acciones que se disparan por eventos que satisfacen un conjunto de condiciones. Cuando ocurre un evento, de forma automática, de forma automática se activan una o más reglas, o bien, se preparan para ser disparadas.

Los sistemas de base de datos activos son muy poderosos, pero el desarrollo aún de aplicaciones pequeñas de reglas activas es una tarea difícil debido a la naturaleza impredecible y sin estructura de su procesamiento. Durante el procesamiento las reglas, éstas activan y desactivan a otras, y los estados intermedio y final de la BD dependen de cuáles reglas se activan y en qué orden se ejecutan. La ejecución de las reglas ECA es indeterminista, asíncrona, concurrente, paralela, etc. Estas características hacen difícil la búsqueda de un modelo formal para la ejecución y el análisis de las reglas ECA. Por estas razones, para definir una base de reglas activas, los lenguajes de reglas ó los lenguajes de consultas son los que más se utilizan en los sistemas manejadores de BD activos (ADBMS) existentes.

Dentro de la literatura, pocos investigadores usan la teoría de redes de Petri para el procesamiento de las reglas [10] [11] [12] [13]. Uno de los trabajos más interesantes es el presentado en [10], donde los autores proponen un modelo de red de Petri para monitorear el flujo de información en la ejecución de reglas y se presenta un sistema manejador de flujos de información para ejemplificar el modelo que están proponiendo. Sin embargo, la estructura obtenida a partir de este modelo tiene mucha redundancia de datos, debido al uso de muchos BEGIN OFs y END OFs para describir eventos, condiciones y acciones.

La simulación de sistemas orientados a eventos, como las reglas ECA, pueden modelarse aplicando la teoría de red de Petri. Se han desarrollado una variedad de editores de redes de Petri [14] para la modelación de los sistemas manejados por eventos. La mayoría de estas herramientas soportan redes de Petri coloreadas, redes de Petri con tiempo, redes lugar/transición, PN estocásticas, redes de Petri de alto nivel y redes de Petri orientadas a objetos. Estas herramientas presentan algunas características en común como el editor gráfico, animación del desplazamiento de tokens, simulación rápida, análisis simple o avanzado, análisis estructural, espacios de estados, formato de archivo intercambiable, reducción de redes, generador de código, etc. Estos editores de PN fueron desarrollados pensando en propósitos generales, como son la modelación, simulación y análisis de sistemas de eventos discretos o sistemas híbridos.

1.3. Motivación del trabajo de tesis

La definición de las reglas en la mayoría de los ADBMS se realiza usando un lenguaje de programación de BD, un lenguaje de consulta, o como objetos en una OODB. Además, el soporte por parte del programador es de gran importancia si las reglas activas se consideran como una tecnología fundamental en ambientes de negocios. Los ADBMS deben incluir importantes herramientas de apoyo para los desarrolladores de bases de reglas activas. Estas herramientas deben de proporcionar la capacidad de realizar consultas dentro de la base de reglas y rastrear o monitorear su comportamiento. Para poder obtener todas estas características es necesario la aplicación de diferentes modelos para hacer la representación y el procesamiento de las reglas. Estos enfoques tienen algunas desventajas, como la existencia de redundancia en los datos de las reglas, así como la necesidad de combinar todos estos modelos para lograr cubrir las características de la base de reglas que deseamos modelar. Otra desventaja que presentan estos enfoques es que no puede realizarse una simulación del comportamiento de las reglas.

Un enfoque diferente y novedoso para modelar y simular ADBMSs son las PN, las cuales combinan la representación y el procesamiento de las reglas en un mismo modelo. Muchos investigadores han desarrollado editores de PN para simular el comportamiento de un modelo de PN. Algunos de estos editores de PN pueden ser encontrados en una base de dato de herramientas de PN [14]. Sin embargo, ninguno de éstos fue desarrollado pensando en los sistemas de BDA porque no pueden generar una PN que modele una base de reglas activas. Nuestro trabajo está enfocado hacia esta

dirección en la aplicación de PN.

Son pocos los investigadores que usan la teoría de PN en sistemas avanzados de BD para el procesamiento de reglas [10] [11][12][13]. El trabajo más interesante es el presentado en [10], donde los autores proponen un modelo de PN (Action Rule Flow Petri Net, ARFPN) para monitorear el flujo de la ejecución de las reglas, además, en este trabajo se muestra un sistema de manejo de flujos para ejemplificar el uso de su modelo de PN. Sin embargo, su modelo tiene mucha estructura redundante debido al uso de muchos BEGIN OFs y END OFs para describir eventos, condiciones y acciones. En nuestro enfoque este inconveniente será cubierto porque implementaremos transiciones condicionales.

1.4. El proyecto de tesis

En este trabajo de tesis se propone un modelo estructural el cual combina la representación de las reglas así como su procesamiento en un solo modelo, el cual es llamado Red de Petri Coloreada Condicional (CCPN, Conditional Colored Petri Net). CCPN puede modelar tanto las reglas como su complicada relación de interacción en un medio gráfico. Si la base de reglas de una base de datos activa es modelada con CCPN, la simulación del disparo de las reglas puede llevarse a cabo. Una interfaz gráfica es desarrollada para llevar a cabo la simulación automática de la base de reglas, así como la conexión de la interfaz con una base de datos para porporcionarle un comportamiento activo.

En [15], trabajo de tesis de maestría, se implementa una herramienta para la modelación y simulación de PN, denominada PetrA. PetrA es un editor gráfico de PN desarrollado en el lenguaje C-objetivo, en las plataformas Open Step y Mac OS X. Las similitudes que presentan PetrA y ECAPNSim son el ambiente de desarrollo (Mac OS X), el uso de un lenguaje orientado a objetos (ECAPNSim está desarrollado completamente en Java) y la simulación del sistema modelado con PN. La diferencia primordial entre ambos sistemas es la capacidad que tiene ECAPNSim para el diseño de CCPN y generarlas automáticamente a partir de la definición de una base de reglas ECA en un archivo de texto. Además, ECAPNSim maneja "tokens" con datos para evaluar la condición de la regla ECA, la cual se incluye en la transición de la CCPN. Hay dos maneras de agregar "tokens" en ECAPNSim: la primera de ellas, al igual que en PetrA y cualquier editor gráfico de PN, es con el ícono de colocación de "tokens", el cual se selecciona y posteriormente se escogen los

lugares donde se introducirán estos elementos. La segunda forma de agregar los "tokens" es con el uso de una consola de usuario, donde se escriben instrucciones en SQL (Standard Query Language) como si se estuviese trabajando directamente en una BD, la consola se conecta con ECAPNSim por medio de "sockets" de Java y envía la instrucción SQL. Por su parte, ECAPNSim genera un "token" con la instrucción recibida y finalmente lo coloca en el lugar que le corresponda dentro de la CCPN. Otra diferencia fundamental entre ECAPNSim y PetrA es la posibilidad del primero de conectarse a la BD Postgres, con el fin de ejecutar en ella la acción de la regla ECA.

1.5. Descripción del documento

En este trabajo de tesis, desarrollamos un modelo extendido de PN, en el cual las reglas ECA se representan, analizan y procesan completamente en un solo modelo. Además, éste modelo puede extenderse para que sea útil en sistemas de BD temporales, orientados a objetos, entre otros. Para probar la factibilidad del modelo propuesto, desarrollamos un editor gráfico (ECAPNSim) en el cual convertimos una base típica de reglas ECA en un modelo estructural, al cual hemos llamado Red de Petri Coloreada Condicional (Conditional Colored Petri Net, CCPN). En ECAPNSim podemos descubrir la relación que existe entre las reglas, así como llevar a cabo una simulación del comportamiento de la base de reglas. Finalmente, conectamos ECAPNSim con una BD, donde ECAPNSim funcionó como base de reglas activas.

En la tesis se presentan varias aportaciones con respecto a la literatura actual:

1. Primeramente se hace un análisis de las ADBMSs, la manera en que proporcionan el comportamiento activo y el inconveniente que tienen de no poder realizar una simulación de la base de reglas, antes de su implementación sobre una BD. Tomando como una alternativa el uso de la teoría de PN.

2. Se propone un modelo nuevo de PN, al cual hemos denominado CCPN, con el que se realiza la representación y procesamiento de las reglas activas en un solo entorno.

3. El desarrollo de un editor gráfico (ECAPNSim) para la generación de modelos de CCPN.

4. Un algoritmo para generar el modelo estructural CCPN, a partir de una base de reglas activas, definidas como reglas ECA.

5. Llevar a cabo la simulación de la base de reglas en el modelo CCPN, generado por ECAPNSim, para observar su comportamiento antes de su implementación en una BD.

6. Conectar ECAPNSim con una BD, para que funcione como base de reglas activas.

7. Se tomaron tres ejemplos de bases de reglas ECA de la literatura para mostrar la factibilidad de la CCPN.

8. Se diseñaron tres BD en Postgres correspondientes al punto anterior, para demostrar el uso de ECAPNSim y la manera en que ECAPNSim les porporciona un comportamiento activo.

La escritura de la tesis se desarrolló de la siguiente manera:

En el Capítulo 2 se presentan los fundamentos acerca de las BDA, dándose una descripción de sus orígenes y propósitos para los cuales fueron diseñadas, así como las características que debe presentar un DBMS para considerarlo activo. En este capítulo se hace un análisis del modelo de conocimiento y el modelo de ejecución que son parte fundamental de los sistemas activos, así como de los componentes que conforman a cada uno de ellos. Dentro de los componentes del modelo de conocimiento se mencionan las propiedades que deben considerarse para los eventos, para las condiciones y para las acciones; y en el modelo de ejecución se explican los pasos que sigue un ADBMS durante el proceso de disparo de reglas.

En el Capítulo 3 se describen los fundamentos teóricos de PN, comenzando por su definición formal, la regla de disparo de transiciones y el poder que tienen para modelar sistemas. Además, se presentan las propiedades que poseen, los métodos de análisis que se les pueden aplicar y se da una breve explicación de algunas extensiones de PN que existen.

En el Capítulo 4 se presenta la sintaxis propuesta de definición de reglas ECA que ofrecen las BDA. Se describe la manera en que una base de reglas ECA se convierte en una CCPN. Se dá una definición formal de la CCPN y la regla de disparo de transiciones ajustada a este modelo. Se propone un algoritmo para realizar la conversión de las reglas ECA en nua CCPN, se aplica a un ejemplo y se dan unos comentarios al respecto.

En el Capítulo 5 se hace un análisis de los simuladores de PN existentes en la actualidad, presentando las herramientas y funcionalidades que ofrecen. Se presenta el análisis y diseño de nuestro editor gráfico ECAPNSim, el cual se desarrolló con el lenguaje de programación orientado a objetos JAVA, dando una descripción del diagrama de clases y de las clases que conforman al

sistema. Se muestra una aplicación del ECAPNSim a una base de reglas. Al final se presentan comentarios sobre el capítulo.

En el Capítulo 6 probamos la factibilidad del modelo CCPN, tomando tres casos de estudios encontrados en la literatura. Generamos con ECAPNSim el modelo CCPN correspondiente a cada uno de ellos, realizamos la simulación del comportamiento de la base de reglas y al final, conectamos ECAPNSim a una BD generada en Postgres. Finalmente se comenta sobre los resultados obtenidos en las pruebas.

En el Capítulo 7 se mencionan los resultados que se obtuvieron con el presente trabajo de investigación así como las limitaciones que presenta. Además se enlistan los trabajos futuros por realizar para mejorar los resultados que se tienen hasta el momento.

Capítulo 2

Fundamentos de Base de Datos Activa

Un Sistema Manejador de Bases de Datos (DBMS, por sus siglas en inglés) consiste de una colección de datos interrelacionados entre sí y un conjunto de programas para acceder a estos datos. La colección de datos, generalmente llamada BD, contiene información acerca de una empresa en particular. El objetivo principal de un DBMS es el de proporcionar un ambiente que sea tanto conveniente como eficiente para recuperar y almacenar información en la BD [16].

Los DBMS están diseñados para manipular grandes cantidades de información. El sistema define la estructura en la que será almacenada la información y provee mecanismos para manipular esta información. En suma, el DBMS da seguridad a la información almacenada, a pesar de caídas del sistema o intentos de accesos sin autorización. Si los datos están siendo compartidos por varios usuarios, el sistema debe evitar posibles resultados indeseados.

Tradicionalmente, los DBMS se han considerado como medios de almacenamiento de información, la cual es accedida posteriormente a través de programas definidos por el usuario o de interfaces interactivas. En este contexto, se han utilizado diferentes herramientas y sistemas para tener facilidad de acceso a la información. Sin embargo, el dominio de aplicación de las BD se ha extendido hacia el procesamiento de información cada vez más compleja, el manejo de grandes cantidades de datos, o donde se necesita un fuerte rendimiento del sistema. Pero el ambiente de trabajo de los DBMS no satisface a estos requerimientos. Esto a conducido a realizar investigaciones en el área de BD, con el afán de que el sistema por si mismo ofrezca una mayor funcionalidad a los programas de aplicación. Dando lugar a sistemas con mayores servicios para modelar aspectos

de la estructura y del comportamiento de la aplicación. Entre los campos que han recibido atención especial en los últimos años se encuentran la programación de BD, BD temporales, espaciales, multimedia, deductivas y activas.

Los DBMSs son pasivos en el sentido de que los comandos (como consultar, actualizar o eliminar registros) son ejecutados por el usuario o por los programas de aplicación. Sin embargo, algunas situaciones no se modelan efectivamente por éste enfoque, donde se necesita realizar algún tipo de movimiento, al presentarse un estado específico de la BD.[2]

Los sistemas de BDA pueden modelar el enfoque planteado anteriormente, al trasladar el comportamiento activo hacia el DBMS. De esta manera, las BDA son capaces de monitorear y reaccionar a circunstancias específicas y de relevancia para una aplicación. Un sistema de BDA debe de tener un modelo de conocimiento (un mecanismo de descripción) y un modelo de ejecución (una estrategia en tiempo de ejecución) para ofrecer un comportamiento activo.

Un enfoque común para el modelo de conocimiento utiliza reglas que se compone de tres elementos: un evento, una condición y una acción. El elemento evento de una regla describe que es lo que debe de ocurrir para que la regla sea capaz de responder a la ocurrencia de ese evento. El elemento condición examina el contexto en el cual el evento a tomado lugar. Y el elemento acción describe la tarea que será llevada a cabo por la regla si el evento ha tomado lugar y la condición se ha evaluado como verdadera.

La mayoría de las BDA definen reglas con los tres componentes descritos, este tipo de reglas es conocida como reglas evento-condición-acción ó regla ECA. En algunas propuestas el evento ó la condición pueden no estar o considerarse como implícita. Si no se especifica el evento, entonces da como resultado una regla condición-acción o regla de producción. Si no se especifica la condición, entonces el resultado es una regla evento-acción.[17]

A simple vista, la introducción de reglas activas en un sistema de BD puede parecer una tarea muy sencilla, pero en la práctica las propuestas que se han presentado han sido hechas para soportar una amplia gama de funcionalidades. Entre las diferentes aportaciones que se presentan están la expresividad del lenguaje de eventos, el ámbito de acceso a los estados de la BD desde la condición y la acción, y la coordinación en la evaluación de la condición y la acción con respecto al evento. La funcionalidad de un sistema específico será influida por un número de factores, incluyendo la naturaleza del modelo de datos pasivo que está siendo extendido, y las categorías de aplicación que

serán soportadas.

2.1. Aplicaciones de Base de Datos Activa

Como mencionamos anteriormente, la investigación en el área de BD apunta a extender el rango de servicios dentro del sistema de BD para representar conceptos de aplicación. Por eso, las capacidades adicionales son muy dependientes de las aplicaciones diseñadas. En el caso de reglas activas, podemos distinguir tres tipos de categorías: [2]

Extensiones de Sistemas de BD. Las reglas activas pueden usarse como un mecanismo primitivo para soportar la implementación de otras partes de un sistema de BD. Por ejemplo, las reglas ECA se han utilizado para establecer restricciones de integridad, vistas materializadas, datos derivados, coordinación de cálculos distribuidos, modelos de transacción, modelado de datos avanzados y actualizaciones automáticas de una BD por medio de la pantalla. La funcionalidad de estas extensiones generalmente está soportada por una sintaxis de alto nivel, además de funciones de mapeo en el conjunto de reglas activas.

Aplicaciones de BD Cerradas. Esta categoría involucra el uso de la funcionalidad activa para describir el comportamiento que debe manifestarse por el sistema sin hacer referencia a dispositivos o sistemas externos. Por ejemplo, las reglas podrían usarse para describir acciones de reparación en una BD de modelación, monitorear las ventas en una BD de control de existencias ó hacer una propagación de cálculos en una BD de diseño arquitectural.

Aplicaciones de BD Abiertas. En esta categoría de aplicación, una BD se usa junto con dispositivos de vigilancia para grabar y responder a situaciones fuera de la BD. Por ejemplo, las reglas pueden usarse en aplicaciones médicas para detectar cambios de temperatura en la condición de un paciente, en aplicaciones de transporte para anticipar calles con tráfico pesado y en control de tráfico aéreo para detectar peligros potenciales en el movimiento de los aviones.

2.2. Modelo de Conocimiento

El modelo de conocimiento de un sistema de BDA indica qué puede decirse sobre las reglas activas en ese sistema. Este modelo es diferente al modelo de ejecución, el cual determina cómo se comportan un conjunto de reglas en tiempo de ejecución. Como el modelo de conocimiento

Evento	**Fuente** $\subset$ {operación de estructura, invocación de comportamiento, transacción, abstracto, excepción, reloj, externo} **Granularidad** $\subset$ {elemento, subconjunto, conjunto} **Tipo** $\subset$ {primitivo, compuesto} **Operadores** $\subset$ {o, y, secuencial, cerradura, veces, negación} **Modos de consumo** $\subset$ {reciente, cronológico, acumulativo, continuo} **Rol** $\in$ {obligatorio, opcional, ninguno}
Condición	**Rol** $\in$ {obligatorio, opcional, ninguno} **Contexto** $\subset$ {DB_T, $Vínculo_E$, DB_E, DB_C}
Acción	**Opciones** $\subset$ {operación de estructura, invocación de comportamiento, reglas actualizadas, cancelar, informar, externa, hacer en lugar de} **Contexto** $\subset$ {DB_T, $Vínculo_E$, $Vínculo_C$, DB_E, DB_C, DB_A}

Cuadro 2.1: Dimensiones para el modelo de conocimiento.

esencialmente soporta la descripción de la funcionalidad activa, las características del modelo tienen una representación directa dentro de la sintaxis del lenguaje de reglas. [2]

El modelo de conocimiento de una regla activa tiene tres componentes principales: un evento, una condición y una acción. Las dimensiones asociadas a estos tres componentes se presentan en la tabla 2.1 y se describen en las siguientes secciones.

Evento

Un evento es algo que sucede en un punto del tiempo. Por lo tanto, al especificar un evento se da una descripción del suceso que será monitoreado. La naturaleza de la descripción y el medio en el cual el evento puede ser detectado depende fuertemente de la **Fuente** o generador de eventos. Alternativas posibles de fuente de eventos son:[17]

- *Operación de estructura*, donde el evento es obtenido por una operación en alguna parte de la estructura de la BD (por ejemplo insertar una tupla, modificar un atributo, o acceder a una tupla).

- *Invocación de comportamiento*, donde el evento se obtiene por la ejecución de alguna operación definida por el usuario.
- *Transacción*, donde el evento se obtiene por un comando de transacción (por ejemplo abort, commit, begin-transaction).
- *Abstracto o definido por el usuario*, en este caso, un mecanismo de programación se usa para permitir a un programa de aplicación que envíe explícitamente una señal de que un evento ha ocurrido.
- *Excepción*, donde el evento se obtiene como resultado de la producción de alguna excepción (por ejemplo un intento de acceso a algunos datos sin la autorización apropiada).
- *Reloj*, en cuyo caso el evento es obtenido en algún punto del tiempo. Este tipo de eventos puede ser absoluto: 13 de noviembre de 1998 a las 15:00 horas; relativo: 10 días después de que las acciones sean vendidas; o periódico: el primer día de cada mes.
- *Externos*, en este caso los eventos se obtienen por lo que ocurra fuera de la BD.

La **Granularidad** de un evento indica si un evento está definido para todos los objetos de un conjunto, para un subconjunto dado ó para elementos específicos del conjunto.

Un evento es de dos **Tipos**:

- primitivo, donde el evento es obtenido por la ocurrencia de un sólo suceso de bajo nivel, el cual pertenece a una de las categorías descritas en la Fuente de eventos.
- compuesto, donde el evento se alcanza por la combinación de eventos primitivos o compuestos, usando un rango de operadores que forman parte del álgebra de eventos.

El rango de operadores de eventos varía de sistema a sistema. Lo operadores más comunes son: disyunción, <E1 o E2> se produce cuando ha ocurrido el evento E1 ó el evento E2; conjunción, <E1 y E2> sucede cuando han ocurrido ambos en cualquier orden; secuencia, <sec(E1, E2)> ocurre cuando pasa E1 antes que E2; cerradura, <cerradura E en Int> es generado solamente una vez la primera vez que se detecta E durante el tiempo Int; historia, <veces (n, E) en Int> se detecta

cuando el evento E ocurre n veces durante el intervalo de tiempo Int; negación, <neg E en Int> detecta la no-ocurrencia del evento E en el intervalo de tiempo Int.

Una gran variedad de álgebras de eventos se han propuesto para determinados sistemas [36], sin embargo, el manejo de eventos compuestos representa un reto desde el punto de vista de su semántica y eficiencia que aún se tienen que estudiar más a fondo. Cuando se detectan eventos compuestos, puede haber varias ocurrencias de eventos del mismo tipo que pueden usarse para formar un evento compuesto, la manera en que son considerados puede definirse utilizando políticas de consumo. Cuatro posibles políticas de consumo se describen a continuación: contexto reciente, el cual considera el conjunto de eventos más recientes que pueden usarse para formar el evento compuesto; contexto cronológico, el cual considera los eventos en orden cronológico; contexto continuo, el cual define una ventana corrediza e inicia una nueva composición con cada evento primitivo que toma lugar; y contexto acumulativo, el cual acumula todos los eventos primitivos hasta que el evento compuesto es finalmente obtenido. [2]

El **Rol** de un evento indica si los eventos siempre deben de ser dados para las reglas activas, o si no es necesario especificarlos. Si el rol es opcional entonces el evento puede ó no especificarse y, por consiguiente, se definen reglas condición-acción, diferentes a las reglas ECA por la falta del evento. Si el rol es ninguno entonces los eventos no pueden especificarse, y solamente pueden definirse reglas condición-acción. Si el rol es obligatorio entonces solamente se pueden definir reglas ECA porque es necesario especificar el evento.

Condición

El Rol de una condición, al igual que el evento, indica si la condición puede o no definirse en la regla. En las reglas ECA, la condición generalmente es opcional, es decir que una regla ECA puede tener o nó la parte condicional. Cuando no se especifica la condición en una regla ECA, o donde el rol es ninguno, da como resultado una regla evento-acción. En sistemas donde son opcionales el evento así como la condición, al menos una de ellas debe especificarse. [2]

El Contexto indica el entorno en el cual la condición se evalúa. Los componentes de una regla no son evaluados de manera aislada a la BD o a los demás componentes. El procesamiento de una regla sencilla puede estar potencialmente asociado con al menos cuatro diferentes estados de la BD: DB_T, la BD al inicio de la transacción actual; DB_E, la BD cuando el evento toma lugar; DB_C, la

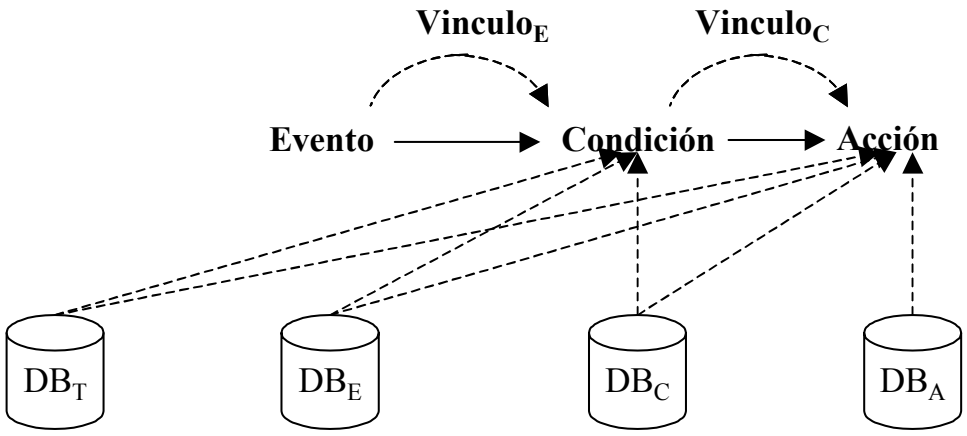

Figura 2.1: El contexto dentro del cual una regla es procesada.

BD cuando la condición es evaluada; y DB_A, la BD cuando la acción es ejecutada. Los sistemas de reglas activas pueden soportar servicios dentro de la condición de la regla que le permite acceder ninguno o varios de los estados DB_T, DB_E, y DB_C, y, además, pueden dar acceso a los vínculos asociados con el evento ($Bind_E$). La disponibilidad de información a los diferentes componentes de una regla se muestra en la figura 2.1.

Acción

El rango de tareas que pueden realizarse por una acción está especificado como sus Opciones. Las acciones pueden modificar la estructura de la BD o del conjunto de reglas; también pueden realizar alguna invocación de comportamiento dentro de la BD ó una llamada externa; o bien, pueden informar al usuario o al administrador del sistema de alguna situación imprevista; cancelar una transacción o tomar alguna alternativa de acción utilizando el hacer-en-lugar-de (do-instead). [2]

El Contexto de la acción es similar al de la condición, e indica la información que está disponible para la acción, como se muestra en la figura 2.1.

Modo de la condición $\subset$ {inmediato, pospuesto, imparcial}
Modo de la acción $\subset$ {inmediato, pospuesto, imparcial} **Granularidad de la transición** $\subset$ {tupla, conjunto} **Política de efecto final** $\in$ {Si, No}
Política de ciclo $\subset$ {iterativo, recursivo}
Prioridades $\in$ {dinámico, numérico, relativo, ninguno}
Calendarización $\in$ {todos paralelos, todos secuenciales, saturación, alguno}
Manejo de errores $\subset$ {cancelar, ignorar, retroceso, contingencia}

Cuadro 2.2: Dimensiones para el modelo de ejecución.

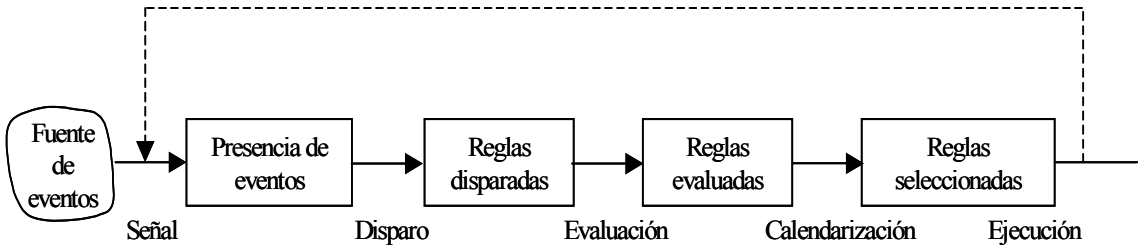

Figura 2.2: Principales pasos que toman lugar durante la ejecución de reglas.

2.3. Modelo de Ejecución

El modelo de ejecución especifica cómo se maneja a un conjunto de reglas en tiempo de ejecución y está caracterizado por las dimensiones presentadas en la tabla 2.2.

Aunque el modelo de ejecución de un sistema de reglas está estrechamente relacionado con aspectos del DBMS sobre el que está montado, hay un número de pasos en la evaluación de las reglas, mostradas en la figura 2.2 y se describen a continuación:

1. Señal se refiere a la aparición de un evento causado por una fuente de eventos.

2. Disparo toma los eventos en la fase anterior y dispara las reglas correspondientes. La asociación de una regla con la ocurrencia de su evento forma una instanciación de regla.

3. Evaluación verifica si se cumple la condición de las reglas seleccionadas. El conjunto de reglas

en conflicto se forma a partir de todas las instanciaciones de reglas cuyas condiciones son satisfechas.

4. Calendarización indica cómo se procesa el conjunto de reglas que se encuentran en conflicto.

5. Ejecución lleva a cabo las acciones de las instanciaciones de las reglas seleccionadas. Durante la ejecución de las acciones otros eventos pueden ser señalizados produciendo el disparo de reglas en cascada.

Estas fases no necesariamente se ejecutan contiguamente, esto dependerá de los modos de acoplamiento para el Evento-condición y Condición-acción. El modo de acoplamiento Condición-acción indica cuando la acción está para ser ejecutada de acuerdo a la evaluación de la condición. Las opciones de modos de acoplamiento ofrecidos más frecuentemente son:[2]

- inmediata, en este caso la condición (acción) se evalúa (ejecuta) inmediatamente después del evento (condición).

- pospuesto, en este caso la condición (acción) se evalúa (ejecuta) dentro de la misma transacción como el evento (condición) de la regla, pero no necesariamente a la primera oportunidad.

- imparcial, en este caso la condición (acción) se evalúa (ejecuta) inmediatamente dentro de una transacción diferente del evento (condición).

La naturaleza de las relaciones entre los eventos y las reglas que disparan es parcialmente capturado por la granularidad de la transición. Esto nos indica si la relación entre las ocurrencias de eventos y las instanciaciones de reglas es de 1 a 1 o de muchos a 1. Cuando la granularidad de la transición es tupla, la ocurrencia de un evento sencillo dispara a una regla sencilla. Cuando la granularidad de la transición es conjunto, una colección de ocurrencia de eventos se utilizan juntos para disparar una regla.

Otra característica que influye en la relación entre los eventos y las reglas que disparan es la Política de efecto final, la cual indica que es mejor considerar el efecto final de la ocurrencia de eventos que el efecto que produce la ocurrencia de uno de los eventos.

La pregunta de qué pasa cuando los eventos son señalizados por la evaluación de la condición o acción de una regla es considerada por la Política de ciclo del modelo de ejecución. En general,

hay dos opciones. Si la Política de ciclo es iterativa, entonces los eventos señalizados durante la evaluación de la condición o durante la ejecución de la acción son procesados al término de la misma; es decir, que la evaluación de la condición y la ejecución de la acción no se suspenden para permitir la respuesta de los eventos señalizados por estas condiciones o acciones. Por el contrario, si la Política de ciclo es recursiva, los eventos señalizados durante la evaluación de la condición o la ejecución de la acción causan que la condición o la acción sean suspendidas. En la práctica, una política de ciclo recursiva probablemente sólo es considerada en sistemas que soportan el procesamiento inmediato de reglas, y algunos sistemas soportan políticas de ciclo recursivo para reglas de respuesta inmediata y una política de ciclo iterativa para reglas pospuestas.

La fase de Calendarización de evaluación de reglas determina que sucede cuando muchas reglas se disparan al mismo tiempo. Los dos principales puntos que hay que considerar son los siguientes: [2]

- *La selección de la siguiente regla que será ejecutada.* Este punto ha recibido mucho atención por parte de la comunidad de los sistemas expertos, ya que se ha visto como parte fundamental del entendimiento y control de un conjunto de reglas. Ciertamente, el orden de las reglas puede influir fuertemente en el resultado y refleja el tipo de razonamiento que el sistema persigue. Ejemplos de enfoques Dinámicos conocidos son aquellos que asignan una prioridad a las reglas basándose en la insistencia de la actualización ó en la complejidad de la condición.

  Las prioridades estáticas son a menudo determinadas por el sistema o por el usuario como un atributo de la regla. En el caso más reciente, una regla se selecciona desde una colección de reglas disparadas simultáneamente para ejecución usando mecanismos de Prioridad. Las reglas pueden colocarse en orden utilizando un esquema numérico, en el cual a cada regla se le asigna un valor absoluto como prioridad o indicando prioridades relativas de reglas manifestando explícitamente que una determinada regla se ejecuta antes que otra cuando ambas se disparan al mismo tiempo.

- *El número de reglas a dispararse.* Las posibles opciones pueden ser (1) ejecutar todas las instanciaciones de regla de manera secuencial; (2) ejecutar todas las instanciaciones de regla de forma paralela; (3) ejecutar todas las instanciaciones de una regla específica antes que alguna otra regla sea considerada, esto es conocido como ejecutar una regla por saturación; y

(4) ejecutar solamente una o algunas instanciación(es) de reglas. El enfoque más apropiado depende de la tarea que vaya a ser mantenida por la regla.

Un aspecto final por considerar es la manera en que se manejarán los errores durante la ejecución de las reglas. La mayoría de los sistemas simplemente cancelan la transacción. Sin embargo, otras alternativas pueden ser más convenientes: una de ellas es ignorar la regla que generó el error y continuar con el procesamiento de otras reglas; otra de ellas es regresar al estado donde el procesamiento de reglas comenzó y reiniciar el procesamiento de reglas o continuar con la transacción; o bien, una alternativa más es adoptar algún plan de contingencia que se esfuerce por recuperarse del estado de error, posiblemente utilizando el mecanismo de manejo de excepciones del sistema de BD pasiva.

2.4. Sistemas de reglas activas

Algunas BD relacionales y OO proporcionan facilidades para la definición de reglas ECA. A continuación se describen BD relacionales y OO que implementan sistemas de reglas activas.

Sistemas relacionales

La incorporación del comportamiento activo en BD relacionales no es algo nuevo y la mayoría de los sistemas comerciales tienen mecanismos de disparo de reglas. Se han desarrollado varios prototipos de investigación [10] [8], los cuales buscan dar un apoyo para que las reglas activas sean más fáciles de entender y mantener. Las propuestas desarrolladas de sistemas de reglas activas por lo regular presentan características comunes, las cuales tienen orígenes en la BD pasiva a la que le están proporcionando comportamiento activo. Por ejemplo, las reglas generalmente se disparan por operaciones en la BD definidas en el sistema (insertar una tupla, modificar una tupla), porque en los sistemas relacionales muy poco se pueden crear operaciones definidas por el usuario. Además, los lenguajes de las BD relacionales, como SQL, proporcionan rutinas para expresar condiciones y para realizar modificaciones, el lenguaje de descripción de reglas normalmente es una extensión del lenguaje de consulta que presenta la BD pasiva. En general, los mecanismos activos propuestos para las BD relacionales solamente tienen un modo de acoplamiento ó un rango limitado, y en la detección de eventos compuestos también tienen un rango limitado. [2]

Las tablas 2.3 y 2.4 muestra como cuatro propuestas de BDA en sistemas relacionales (llamadas Starburst, POSTGRES, Ariel, y SQL-3) caen dentro de los esquemas descritos en las secciones anteriores.

Dimensión	Starburst	POSTGRES	Ariel	SQL-3
Modelo de Conocimiento				
Evento:				
Fuente	Oper. de est.	Oper. de est.	Oper. de est.	Oper. de est.
Granularidad	Conjunto	Conjunto	Conjunto	Elemento, Subconjunto, Conjunto
Operadores	o			
Politica de consumo	Acumulativa	N/A	N/A	N/A
Papel	Obligatorio	Obligatorio	Opcional	Obligatorio
Condición:				
Papel	Opcional	Opcional	Opcional	Opcional
Contexto	Vínc_E, DB_C	Vínc_E, DB_C	Vínc_E, DB_C	Vínc_E, DB_E, DB_C
Acción:				
Opciones	Oper. de est. Cancelar Act. reglas	Oper. de est. Cancelar Do instead	Oper. de est.	Oper. de est. Invoc. de comp Externo Do instead
Contexto	Vínc_E, Vínc_C, DB_A	Vínc_E, DB_A	Vínc_E, DB_A	Vínc_E, DB_E, DB_A

Cuadro 2.3: Dimensiones aplicadas a los sistemas activos de BD relacionales.

Starburst El sistema de reglas activas Starburst agrega funcionalidad activa a una BD relacional extensible y se ha usado para examinar un número de aplicaciones internas de la BD, incluyendo restricciones de integridad y vistas materializadas.[2]

Dimensión	Starburst	POSTGRES	Ariel	SQL-3
Modelo de Ejecución				
Modo condición	Pospuesto	Inmediato	Inmediato, pospuesto	Inmediato
Modo de acción	Inmediato	Inmediato	Inmediato	Inmediato Pospuesto
Granularidad de Trans.	Conjunto	Tupla	Conjunto	Tupla Conjunto
Política de efecto final	Si	No	Si	No
Política de ciclo	Iterativa	Recursiva	Iterativa	Recursiva
Prioridades	Relativa	Ninguna	Numérica	Ninguna
Calendarización	Secuencial	Secuencial	Secuencial	Secuencial
Manejo de errores	Cancelar	Cancelar	Cancelar	Retroceder

Cuadro 2.4: Dimensiones aplicadas a los sistemas activos de BD relacionales.

La característica más importante que presenta Starburst es su modelo de ejecución basado en conjuntos, en el cual las reglas son disparadas por el efecto final de un conjunto de cambios en la BD. Cuando una operación que es monitoreada por una regla toma lugar, la naturaleza del cambio es anotado en una tabla de transiciones. Las anotaciones en estas tablas son revisadas para tomar en cuenta operaciones de actualizaciones subsecuentes; por ejemplo, si una tupla es insertada y después actualizada, el efecto-final es anotado como la inserción de una tupla modificada; si una tupla es insertada y después eliminada, el efecto-final es que no se ha realizado ninguna operación. La información que es almacenada en las tablas de transición es utilizada para disparar reglas en ciertos puntos de la regla, que puede tomar lugar durante la transacción o al final de la misma. En este contexto, los eventos no disparan a las reglas directamente. El hecho de que un evento ha ocurrido es grabado en una tabla de transición para consideraciones posteriores, y la coordinación o el orden de eventos específicos no es tomado en cuenta por el calendarizador de la ejecución de reglas.

Cada regla que está monitoreando un evento particular tiene acceso al efecto-final de todas las actualizaciones de relevancia para ese evento, y que no han sido consideradas por la regla; cuando

una regla es disparada y ejecutada muchas veces dentro de una transacción sencilla, los cambios a los que tiene acceso son aquellos que han tomado lugar desde la última ejecución de la regla.

El sistema de reglas de Starburst puede considerarse como conservador en su diseño y que el conjunto de servicios que proporciona es fijo y modesto. Sin embargo, la semántica de la ejecución de las reglas en Starburst es aún bastante compleja, lo que puede deberse al hecho de que proporcionar muchas funciones probablemente nos lleve a conjuntos de reglas que son difíciles de entender y mantener.[5].

POSTGRES El sistema de reglas activas es solamente una de las extensiones que presenta el modelo relacional de POSTGRES, además de las rutinas de representación de objetos y de operaciones definidas por el usuario. El sistema de reglas de POSTGRES está considerado dentro de los sistemas de BD relacionales, porque su funcionalidad orientada a objetos no es utilizada en la definición de las reglas activas. Este sistema de reglas, al igual que el de Starburst, son considerados muy conservadores, aunque una diferencia importante que tienen es que el sistema de reglas de POSTGRES está orientado a tuplas. Además, las acciones que se llevan a cabo al ser detectado un evento toman efecto inmediatamente, en lugar de posponerlo hasta un cierto punto de la ejecución de la regla. [2]

Ariel Una característica de Ariel que lo distingue de Starburst o POSTGRES es que el evento puede ser opcional. Ariel ofrece principalmente reglas condición-acción [2]. La factibilidad de las reglas condición-acción comparadas con las reglas ECA depende del contexto en que se estén aplicando. El sistema Ariel es una implementación de un DBMS relacional con un sistema de reglas incorporado. El sistema de reglas de Ariel (ARS) está basado en el modelo de sistemas de producción. El diseño de Ariel adopta trabajos previos en sistemas de producción como OPS5, haciendo los cambios necesarios para mejorar la funcionalidad y rendimiento del sistema de producción en un ambiente de BD. Los cambios realizados incluyen a una extensión del lenguaje de reglas POSQUEL con una sintaxis como lenguaje de consulta; una red de discriminación para la validación de la condición de la regla adaptado al ambiente de la BD; y agrega medidas para integrar el procesamiento de reglas con comandos y transacciones de BD orientadas a conjuntos. Una característica distinguible de Ariel es que se trata de una implementación de una DBMS relacional con un sistema de reglas que está fuertemente acoplado con el procesador de consultas. El mecanismo para verificar las

condiciones de las reglas en Ariel, llamado A-TREAT, es una variación del algoritmo TREAT [3]; este algoritmo está mejorado con características para incrementar la velocidad en la verificación de predicados de selección en las condiciones de la regla, reduce la cantidad de información de estado que se encuentra en la red, y maneja eventos, transiciones y patrones basados en condiciones de una manera uniforme. [1].

SQL-3 La mayoría de los sistemas comerciales soportan mecanismos de disparo de reglas. Sin embargo, el modelo de conocimiento y el de ejecución de estos mecanismos varían entre un sistema y otro. Con la finalidad de proporcionar un soporte más consistente para mecanismos activos en sistemas relacionales, el disparo de reglas está incluido en la versión estándar SQL-3. [2]

El estándar SQL-3, como muchos de los sistemas comerciales, soporta disparo de reglas de bajo nivel (con una granularidad de tupla para transiciones). El disparo de reglas a nivel de declaraciones es ejecutado una sola vez en respuesta a una operación de modificación a una tabla, sin importar como pueden ser afectadas muchas tuplas debido a la modificación. Sin embargo, quizá la característica más importante del estándar SQL-3 es que hace explícito cómo el disparo de las reglas interactúan con otras características encontradas en las BD relacionales y en particular, con los mecanismos declarativos para checar integridad de datos.[2]

Sistemas Orientados a Objetos

Las BD orientadas a objetos (OODBs), a diferencia de sus predecesoras las BD relacionales, siempre han mantenido una asociación estrecha entre el comportamiento definido por el usuario y la información. Este comportamiento generalmente se expresa como métodos adjuntos a las clases que forman la estructura de la BD [2]. Lo anterior junto con la ocultación de ciertos aspectos de la estructura de un objeto utilizando encapsulamiento, significa que ciertas tareas que se realizan como parte del comportamiento activo en BD relacionales se pueden llevar a cabo utilizando métodos en sistemas orientados a objetos. A pesar de esto, abundan propuestas [2] [8] [7] [9] para hacer extensiones a las OODBs y proporcionarle comportamiento activo, dándose las primeras de ellas a pocos años de haberse dado a conocer el primer trabajo sobre OODBs pasivas. El interés puesto en las investigaciones de OODB probablemente fue motivado por la tendencia de las OODBS para ser usadas en aplicaciones avanzadas, donde la necesidad de tener funciones para manejo total del

comportamiento es mayor que en la mayoría de los dominios tradicionales. Además, la naturaleza de ciertos dominios ha motivado la investigación en el desarrollo de BDA que son significativamente más poderosas que las descritas para BD relacionales en las tablas 2.3 y 2.4, como las que se muestran en las tablas 2.5, 2.6, 2.7 y 2.8. Una diferencia común entre las BD relacionales y las OODBs es que los eventos primitivos en OODBs a menudo están asociados con la invocación de los métodos, lo que es mejor que acceder o modificar la estructura de la BD [3]. Esto en parte se deriva del deseo de evitar la reducción de independencia de datos, al conectar el comportamiento activo directamente a la estructura de la BD y en parte se debe al hecho que algunos sistemas utilizan arquitecturas en capas, en las cuales no es sencillo obtener los eventos de las operaciones básicas que se aplican a la estructura. A continuación se presenta una descripción de ocho proyectos que desarrollan BDA orientadas a objetos: HiPAC, EXACT, NAOS, Chimera, Ode, SAMOS, Sentinel y REACH. [2]

HiPAC El proyecto HiPAC es el pionero de muchas de las más importantes ideas en BDA, como los modos de acoplamiento y los eventos compuestos, aunque el diseño final no fue implementado completamente. HiPAC estaba asociado con la OODB pasiva PROBE y los objetos en este modelo fueron usados para almacenar las reglas ECA de la extensión activa [2]. Otras características distintivas de HiPAC son: la ejecución en paralelo del disparo de las reglas como subtransacciones, la extensión del álgebra de consulta con un operador de cambios que permite acceder a las relaciones delta que monitorean el efecto final de un conjunto de cambios y la identificación de aplicaciones en tiempo real que pueden beneficiar a los servicios que ofrece la BDA. [3]

EXACT EXACT es un manejador de reglas activas extensible, el cual agrega ciertas funciones para darle comportamiento activo a la OODB ODAM, donde las instancias, clases, reglas y eventos son representados uniformemente como objetos de BD. EXACT presenta dos controversias, específicamente: que la información de control pocas veces se refiere a reglas sencillas sino a conjunto de reglas y que las reglas que soportan aplicaciones distintas requieren de diferentes modelos de ejecución. Para salvar estos inconvenientes, EXACT proporciona un modelo de reglas extensible donde las colecciones de reglas se desarrolladan para que compartir características similares. EXACT se ha utilizado para experimentación en el desarrollo de servicios a BD avanzadas. [2]

Dimensión	HiPAC	EXACT	NAOS	Chimera
Modelo de Conocimiento				
Evento:				
Fuente	Oper. de est.	Invoc. de comp.	Oper. de est.	Oper. de est.
	Invoc. de comp.	Transacción	Invoc. de comp.	Inv. de comp.
	Transacción	Excepción	Transacción	
	Reloj	Reloj, Externo	Reloj, Programa	
	Abstracto	Abstracto	Abstracto	
Granularidad	elemento	elemento	Conjunto	Conjunto
	Conjunto	Conjunto		
Operadores	o, sec., veces	y, o	y, o, neg, sec.	
Pol de cons.		Acumulativa		Acumulativa
Papel	Obligatorio	Obligatorio	Obligatorio	Obligatorio
Condición:				
Papel	Opcional	Obligatorio	Opcional	Obligatorio
Contexto	Vínc$_E$, DB$_C$	Vínc$_E$, DB$_C$	Vínc$_E$, DB$_C$	Vínc$_E$, DB$_T$, DB$_C$
Acción:				
Opciones	Oper. de est.	Invoc. de comp.	Oper. de est.	Operación de est.
	Invoc. de comp.	Actualizar reglas	Invoc. de comp.	Invoc. de comp.
	Actualizar reglas	Cancelar	Actualizar reglas	Cancelar
	Cancelar	Externo	Cancelar, Externo	
	Externo		Do instead	
Contexto	Vínc$_E$, Vínc$_C$, DB$_A$	Vínc$_E$, Vínc$_C$, DB$_A$	Vínc$_E$, Vínc$_C$, DB$_A$	Vínc$_E$, Vínc$_C$, DB$_T$, DB$_A$

Cuadro 2.5: Dimensiones aplicadas a sistemas activos orientados a objetos.

NAOS NAOS es un sistema de reglas activo para la OODB O2, el cual fue implementado como parte del kernel de O2, en lugar de agregarlo como una capa en el nivel más alto. [2]

Como NAOS proporciona un soporte total para dos lenguajes, O2C y OQL, es capaz de aceptar expresiones declarativas en las condiciones, usando OQL, y una potente declaración de acciones

Dimensión	HiPAC	EXACT	NAOS	Chimera
Modelo de Ejecución				
Modo condición	Inmediato Pospuesto Imparcial	Inmediato Pospuesto	Inmediato Pospuesto	Inmediato Pospuesto
Modo de acción	Inmediato Pospuesto Imparcial	Inmediato Pospuesto	Inmediato	Inmediato
Granularidad de Trans.	Conjunto	Tupla	Tupla, Conjunto	Conjunto
Política de efecto final	Si	No	Si	Si
Política de ciclo	Iterativa Recursiva	Recursiva	Iterativa Recursiva	Iterativa
Prioridades	Relativa	Relativa Numérica	Relativa	Relativa
Calendarización	Paralela	Secuencial	Secuencial	Secuencial
Manejo de errores	Contingencia	Cancelar	Cancelar Ignorar	Cancelar

Cuadro 2.6: Dimensiones aplicadas a sistemas activos orientados a objetos

utilizando O2C. El modelo de ejecución de NAOS es un poco raro, porque cuando realiza búsquedas de primero en profundidad, realiza un procesamiento recursivo de reglas inmediatas, pero cuando realiza búsquedas de primero a lo ancho, aplica el procesamiento de reglas pospuestas.

Chimera El sistema de reglas activas Chimera es único entre los que aquí se mencionan, porque está construido sobre una BD deductiva orientada a objetos. El uso de un lenguaje deductivo para expresar condiciones motiva a una visualización orientada a conjuntos del procesamiento de reglas y la información es enviada desde el evento a la condición por medio de consultas a un historial de eventos [2]. Otra característica diferente de Chimera es que las condiciones y acciones de la regla pueden acceder a estados de la BD que ya sucedieron con anterioridad, ya sea al inicio de la transacción actual o bien cuando la ejecución de la regla disparada ha terminado [8]. Chimera es

Dimensión	Ode	SAMOS	Sentinel	REACH
Modelo de Conocimiento				
Evento:				
Fuente	Invoc. de comp.	Invoc. de comp. Transacción Reloj Abstracto	Invoc. de comp. Transacción Reloj	Oper. de est. Invoc. de comp. Transacción Reloj
Granularidad	elemento Conjunto	elemento Conjunto	elemento Conjunto	Conjunto
Operadores	y, o, neg, sec., veces	y, o, neg, sec., veces	y, o, sec., veces, alguno	y, o, sec., veces, neg
Politica de consumo	N/A	Cronológica	Reciente Cronológica Contínua Acumulativa	Cronológica
Papel	Ninguno	Obligatorio	Obligatorio	Obligatorio
Condición:				
Papel	Obligatorio	Obligatorio	Opcional	Obligatorio
Contexto	DB_C	$Vínc_E$, DB_C	$Vínc_E$, DB_C	$Vínc_C$, DB_C
Acción:				
Opciones	Invoc. de comp.	Oper. de est. Invoc. de comp. Informe Externo Cancelar	Oper. de est. Actualizar reglas Informe Cancelar	Operación de est. Invoc. de comp. Informe Externo Cancelar
Contexto	DB_A	$Vínc_E$, DB_A, $Vínc_C$	$Vínc_E$, DB_A	$Vínc_E$, $Vínc_C$, DB_A

Cuadro 2.7: Dimensiones aplicadas a sistemas activos orientados a objetos basados en C++.

implementado al compilar declaraciones de usuario en una forma interna, la cual es interpretada por un sistema en tiempo de ejecución que almacena la BD y los datos del sistema de reglas, utilizando ALGRES, que es un manejador de almacenamiento relacional extendido. [2]

Dimensión	Ode	SAMOS	Sentinel	REACH
Modelo de Ejecución				
Modo condición	Inmediato	Inmediato	Inmediato	Inmediato
		Pospuesto	Pospuesto	Pospuesto
		Imparcial		Imparcial
Modo de acción	Inmediato	Inmediato	Inmediato	Inmediato
	Pospuesto	Pospuesto		
	Imparcial	Imparcial		
Granularidad de Trans.	Tupla	Tupla	Tupla	Conjunto
	Conjunto			
Política de efecto final	No	No	No	No
Política de ciclo	Iterativa	Recursiva	Recursiva	Iterativa
				Recursiva
Prioridades	Ninguna	Relativa	Relativa	Numérica
Calendarización	Secuencial	Secuencial	Paralela	Paralela
Manejo de errores		Cancelar	Cancelar	Contingencia

Cuadro 2.8: Dimensiones aplicadas a sistemas activos orientados a objetos basados en C++.

Ode El sistema de BD Ode está definido utilizando el lenguaje de programación orientado a objetos O++, que es una extensión de C++ que da facilidades para el manejo de BD. En este sistema se proporcionan dos categorías de reglas que están divididas semánticamente en restricciones y disparadores de reglas, cada una con una sintaxis y un modelo de ejecución diferentes. Aunque con el disparo de reglas se manejan restricciones, los autores argumentan que la distinción aclara el papel que juega cada una de ellas, y facilita su implementación. Tanto las restricciones como los disparos de las reglas se definen en el nivel de clases y están sujetos a la herencia así como a otras propiedades de las clases.[2]

Una restricción consiste de un predicado en el estado de un objeto, una acción sencilla que será ejecutada si la condición llega a ser falsa. Una vez que la acción se ha ejecutado, el sistema checa la condición de nuevo. Si aún es falsa, la transacción se cancela. Las restricciones afectan a todas las instancias de una clase y están activadas permanentemente.[7]

A diferencia de las restricciones, el disparo de reglas tiene que ser activado explícitamente en objetos particulares. Si una regla está activada y si la condición llega a ser verdadera, su acción se ejecuta.

SAMOS Proporciona servicios de comportamiento activo en la capa más alta de la ObjectStore comercial OODB. Como los desarrolladores de SAMOS no tuvieron acceso al código fuente de ObjectStore, el sistema activo se implementó como una capa superior en la arquitectura de ObjectStore en lugar de colocarlo como parte del kernel. [2]

Quizá la característica más importante de SAMOS es su detector de eventos, donde la semántica de éste detector está basado en el uso de PN, el cual está implementado usando una gráfica que refleja la estructura de la PN. El lenguaje de eventos por sí mismo representa a usuarios con una serie de operadores que son compartidos por otros sistemas que tienen lenguajes de eventos, como HiPAC, Sentinel o REACH.

Sentinel Es una extensión activa de C++ basado en el sistema OpenOODB de Texas Instruments. El objetivo de este proyecto es el de proporcionar mecanismos para la especificación de eventos, representación de reglas como objetos de BD y la integración de un sistema de reglas con un manejador sofisticado de transacciones [2]. En particular, los modos de consumo que son ampliamente utilizados para la descripción de eventos compuestos en sistemas de reglas fueron implementados primero en Sentinel.

REACH Tiene mucho en común con Sentinel, ya que también es una extensión activa de OpenOODB. El énfasis en el proyecto REACH es el desarrollo de un esquema para facilitar la comprensión de cómo el manejador de reglas interactúa con modelos de transacción complejos, con la finalidad de soportar aplicaciones abiertas. [2]

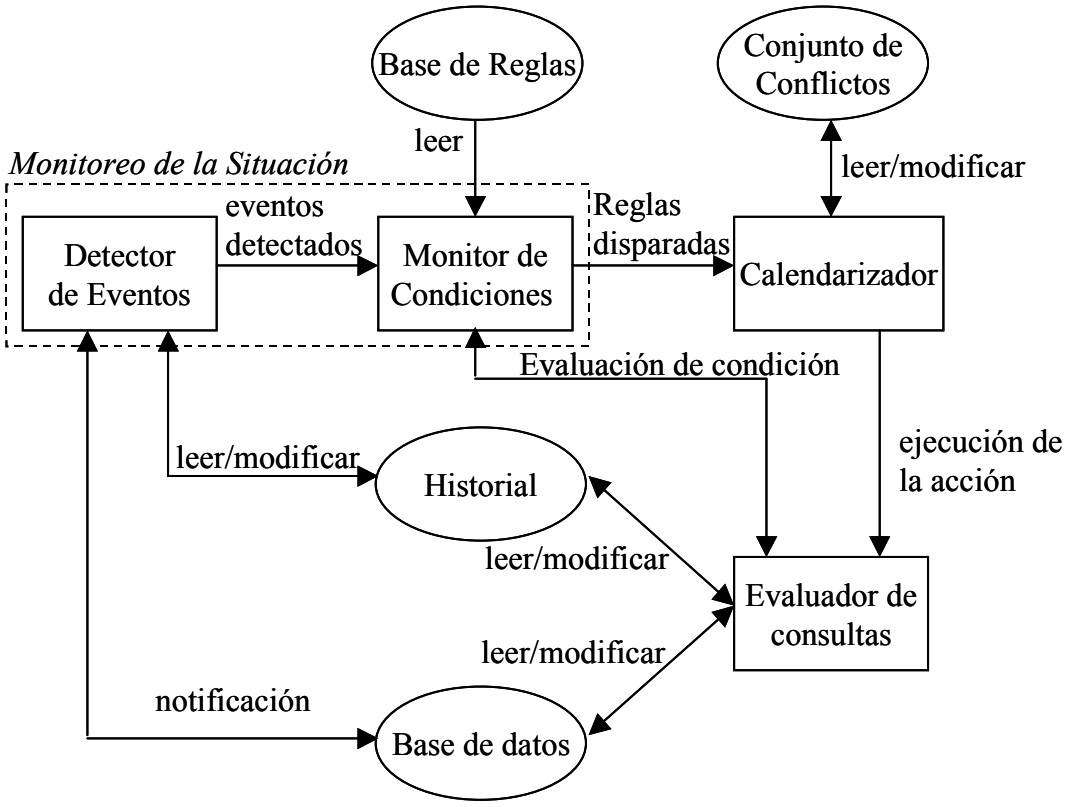

Figura 2.3: Arquitectura abstracta de un sistema de reglas activas.

2.5. Arquitectura de las Bases de Datos Activas

En la figura 2.3 se hace una representación abstracta de la arquitectura de un sistema de reglas activas. En esta figura se explican los principales procesos (rectángulos) y datos almacenados (elipses) usados para implementar las operaciones de sistemas de las BDA mostrados en la figura 2.2.

Los procesos principales son los siguientes: [2]

1. El Detector de Eventos monitorea los eventos de interés al sistema de reglas han tomado lugar, si se informa de la presencia de algunos eventos primitivos en la BD o en fuentes externas: los eventos compuestos son construidos a partir de la entrada de los eventos primitivos junto

con la información referente a eventos que hayan sucedido anteriormente, los cuales pueden obtenerse del historial de eventos.

2. El Monitor de Condiciones evalúa las condiciones de las reglas asociadas con los eventos que han sido percibidos por el detector de eventos. En los sistemas que soportan reglas condición-acción, no hay declaraciones explícitas de los eventos que deben ser monitoreados, aunque en implementaciones reales se deben monitorear al menos eventos primitivos.

3. El Calendarizador compara las reglas que se hayan disparado recientemente con aquellas que se dispararon anteriormente, modifica el conjunto de conflictos y dispara algunas reglas que son programadas para un procesamiento inmediato.

4. El Evaluador de Consultas ejecuta consultas o acciones en la BD. Requiere del acceso al estado actual de la BD ó a estados que se hayan presentado anteriormente para poder ver cómo evoluciona la BD.

La funcionalidad de cada uno de estos componentes depende mucho de los modelos de conocimiento y de ejecución del sistema de BDA, el que a la vez está influenciado por el ambiente dentro del cual se desarrolló. Pueden identificarse dos categorías principales de BDA: [2]

En capas. El componente activo está desarrollado como una capa de software en el nivel mas alto de un sistema de BD pasivo, al cual no se le hace modificación alguna. Este enfoque tiene la ventaja de que no se requiere acceder al código fuente del sistema de BD pasiva, y que el sistema activo resultante puede ser fácilmente portable para su uso en otros sistemas pasivos.

Integrados. El componente activo está desarrollado al cambiar parte del código fuente de un sistema de BD pasivo existente.

2.6. Desarrollo de aplicaciones activas

Los mecanismos activos que se proporcionan dentro de los sistemas de BD no siempre son utilizados por los usuarios. Ciertamente, la experiencia obtenida en la aplicación de BDA indica que aunque tales servicios son adecuados para realizar un sinnúmero de diferentes tareas, no son sencillas de utilizar. En la definición del comportamiento activo en un sistema pueden presentarse las siguientes dificultades: [2]

- No es fácil determinar que partes de una aplicación consideran mecanismos activos y cuales otras técnicas y además, qué inconvenientes se presentan al utilizar una base de reglas.
- La funcionalidad de una base de reglas grande puede ser difícil de entender, con reglas interactuando en medios complejos y con descripciones complicadas de cómo el control fluye a través de una aplicación.
- Las herramientas asociadas con un sistema de reglas activo pueden ser pocas, con un soporte mínimo para visualizar, monitorear o trazar las reglas activas.

Diseño de reglas

Hay técnicas bien establecidas [2] para el diseño de BD que recalcan la descripción de los aspectos estructurales de la información en un dominio y cuáles de ellos se utilizan junto a mecanismos para la descripción de procesos que capturan la semántica de aplicaciones compuestas. Aunque el comportamiento activo está estrechamente ligado a las estructuras almacenadas en la BD y las acciones de las reglas llevan a cabo acciones que pueden ser vistas como procesos, no está claro aún qué técnicas son más adecuadas para las funcionalidades proporcionadas por las reglas activas. En particular, dados los requerimientos de una aplicación, las técnicas de diseño deben dar orientación para saber qué aspectos deben implementarse usando mecanismos activos y aquellos que se representan mejor utilizando otros mecanismos.

Análisis de reglas

Cuando se está diseñando una base de reglas activas hay varios puntos que se deben de tener en cuenta:

- *Terminación.* La ejecución de las acciones de una regla puede causar la ocurrencia de algunos eventos nuevos. Estos eventos pueden, sucesivamente, disparar otras reglas, llevando a posibles cadenas de disparo de reglas. El problema de terminación puede observarse en la figura 2.4. Donde la regla R1 dispara a las reglas R2 y R3; la regla R2 dispara a la regla R4, terminando en este punto ésta secuencia de disparo; sin embargo, R3 vuelve a disparar a R1, provocando un ciclo infinito entre R1 y R3.

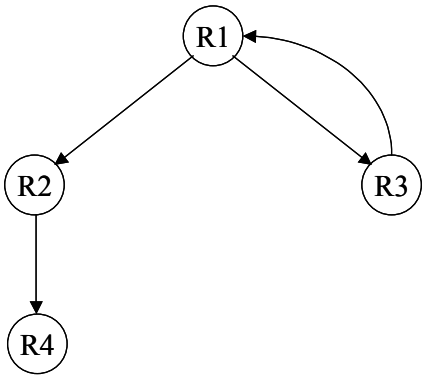

Figura 2.4: Gráfica que representa posibles dependencias en el disparo de reglas.

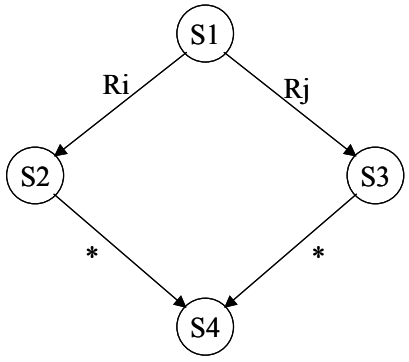

Figura 2.5: Gráfica que representa confluencia en el comportamiento del disparo de reglas.

- *Confluencia*. La confluencia puede entenderse considerando que el disparo de una regla en un estado de la BD lleva a la creación de un nuevo estado. Si más de una regla se dispara al mismo tiempo, entonces más de un estado sucesor potencial existe, como se muestra en la figura 2.5, donde los estados S2 y S3 son estados sucesores de S1 que resultan del disparo las reglas R_i y R_j, respectivamente. Una base de reglas es confluente si al dispararse alguna de las dos reglas R_i y R_j en un estado inicial S1, se garantiza que se llegará a un solo estado final. Sea cual sea el orden en el que alguna secuencia simultánea de disparo de reglas se seleccione para dispararse (representado como * en la figura 2.5).

2.7. Comentarios finales

El problema que presentan las BDA actuales es que no proporcionan un medio en el cual se realice una modelación y simulación de las reglas en un mismo entorno. Debido a esto es difícil llevar a cabo un análisis completo del comportamiento que puede seguir la base de reglas activas al estar en operación sobre una BD pasiva. La mayoría de los enfoques que se presentan dividen el procesamiento de las reglas de la representación de las mismas, lo que nos lleva a realizar doble trabajo para tener un control sobre el comportamiento activo que se pretende implementar.

Como alternativa de solución a este inconveniente, en este trabajo de tesis se propone la utilización de la teoría de PN. Con la cual podemos llevar a cabo la representación de las reglas de manera gráfica, donde es más sencillo detectar problemas que puedan aparecer en una base de reglas activas, así como de realizar una simulación del comportamiento de la base de reglas sin necesidad de tener contacto directo con la BD.

Capítulo 3

Fundamentos de redes de Petri

Históricamente hablando, 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 este 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. [18]

Una PN es un tipo particular de gráfica dirigida bipartita, compuesta 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 se representan por círculos y las transiciones por barras ó rectángulos. En su forma más simple, una PN se representa por una transición unida con sus lugares de entrada y lugares de salida. Esta red básica se utliza 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.

En este capítulo se dan los fundamentos basicos de PN. En la sección 3.1 se presenta la definición

formal de PN, así como definiciones preliminares de la teoría de PN. En la sección 3.2 se describen las reglas de activación y disparo de transiciones de una PN, se muestra un ejemplo donde se aplican estas reglas y la distribución de los tokens al disparse las transiciones activadas. En la sección 3.3 se muestra el poder de las PN para representar diferentes tipos de sistemas; como sistemas de ejecución secuencial, con conflicto, concurrentes, sincronizados, fusionados, confusos, mutuamente exclusivos y con prioridad. En la sección 3.4 se describen las propiedades de las PN: alcanzabilidad, limitada, segura, conservadora y viva. En la sección 3.5 se presentan los métodos de análisis de PN que existen, como el árbol de alcanzabilidad, matriz de incidencia, ecuación de estado, análisis de invariancia, reglas de simplificación de PN y la simulación de PN. Finalmente, en la sección 3.6, se explica el motivo por el cual existen extensiones de PN y se explican algunas de ellas.

3.1. Definiciones preliminares

Formalmente, una PN se define como sigue: [19]

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

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

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

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

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

$M_0 : P \rightarrow \{0, 1, 2, 3, \cdots\}$ es la marca inicial.

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

Una *marca* es la asignación de *tokens* a los lugares de una PN. Un token es un concepto primitivo que forma parte de las PN (como los lugares y las transiciones), el cual es asignado a cada uno de los lugares. Los tokens se usan para definir la ejecución de una PN, por lo tanto, la cantidad y la posición de los tokens cambia durante la ejecución.

Algunas interpretaciones típicas de las transiciones y de sus lugares de entrada y de salida se muestran en la tabla 3.1.

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"

Cuadro 3.1: Algunas interpretaciones de transiciones y lugares.

La PN es una herramienta de modelación gráfica y matemática que se aplican 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 [19]. Para la modelación de éstos sistemas, se utilizan las PN de dos maneras: gráfica 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 se utilizan 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 $w(p_j, t_i) = 1$. 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 $w(t_i, p_j) = 1$. 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,

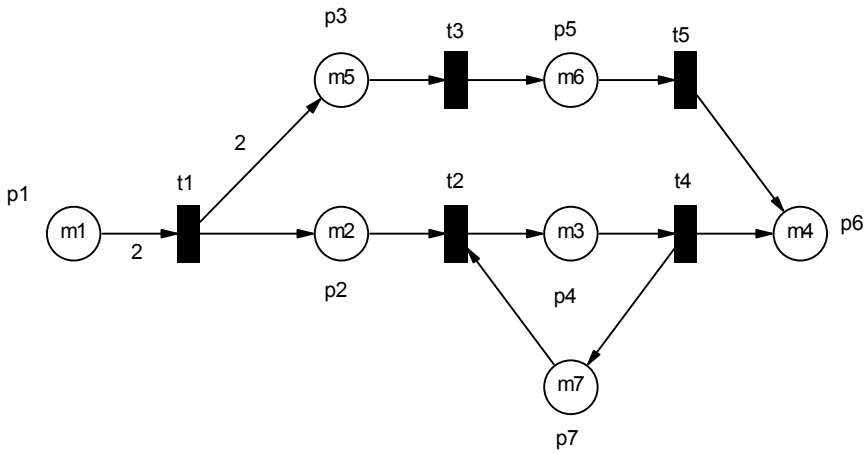

Figura 3.1: Una red de Petri sencilla.

etiquetado con el número de arcos que representa o su peso k. Un círculo que contiene un punto en su interior representa a un place que contiene un token. [19]

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

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

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

$w(p_1, t_1) = 2,\ w(p_i, t_1) = 0,\ para\ i = 2, 3, ..., 7$;

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

. . . .

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

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

. . . .

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

3.2. Disparo de transiciones

La ejecución de una PN se controla por el número de tokens y su distribución en la red. Los tokens se alojan en los lugares y controlan la ejecución 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 se ejecuta a través del disparo de transiciones, el cual se lleva a cabo con la aplicación de la regla de activación (*enabling rule*) y posteriormente se aplica la regla de disparo (*firing rule*), las cuales gobiernan el flujo de los tokens por la red. [20]

1. *Regla de activación de transiciones*: Una transición t se dice que será *activada* 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 $M(p) \geq w(p,t)$ para algún p en P.

2. *Regla de disparo de transiciones*:

   a) Una transición activada t puede dispararse o no dependiendo de la interpretación adicional que tenga la transición y

   b) El disparo de una transición activada t elimina de cada lugar de entrada p_i el mismo número de tokens que el valor del peso k del arco dirigido que conecta a p_i con t. Además, agrega a cada lugar de salida p_j el número de tokens que sea igual al peso k del arco dirigido que conecta a la transición t con el lugar p_j.

Matemáticamente, el disparo de t en M alcanza una marca nueva:
$M\prime(p) = M(p) - w(p_1,t) + w(t,p_2)$, para $t \in T$ y $p1, p2 \in P$

Solamente las transiciones activadas 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 activada 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

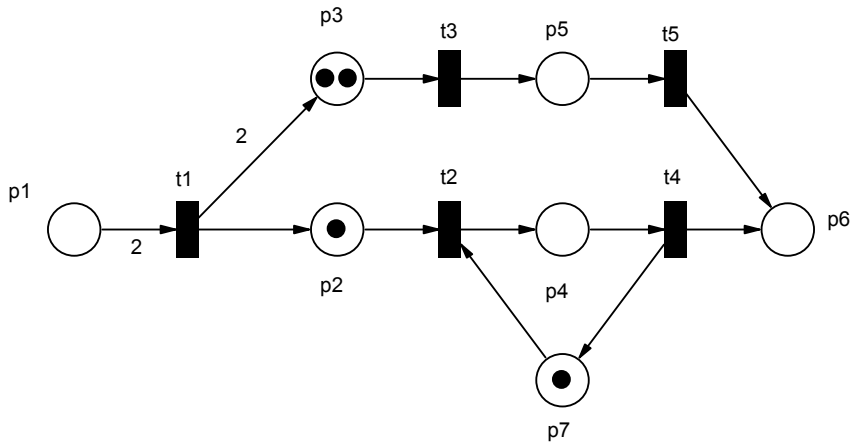

Figura 3.2: Red de Petri resultante al disparar t_1 en la red del ejemplo 1.

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*.[19]

Para mostrar un ejemplo de la activación 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 activada. Al realizar el disparo de t_1 obtenemos una nueva marca, digamos M_1. De acuerdo con la regla de disparo de transiciones 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 activadas. Si se dispara a t_2 se obtiene una nueva marca, digamos M_2:

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

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

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

3.3. Poder de representación de las redes de Petri

Las características típicas manifestadas por las actividades de un sistema dinámico de eventos discretos (DEDS), como concurrencia, toma de decisiones, sincronización y prioridades, son mode-

ladas efectívamente 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. [18]

1. *Ejecución secuencial.* En la figura 3.3(a), la transición t_2 se dispara 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 sistema.

2. *Conflicto.* Las transiciones t_1 y t_2 están en conflicto en la figura 3.3(b). Ambas están activadas, pero el disparo de una de ellas conduce a la desactivación 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 se resuelven 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 éstos a la vez, 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 se captura por el tipo de transición mostrada en la figura 3.3(d). Aquí, t_1 se activa 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 es 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.

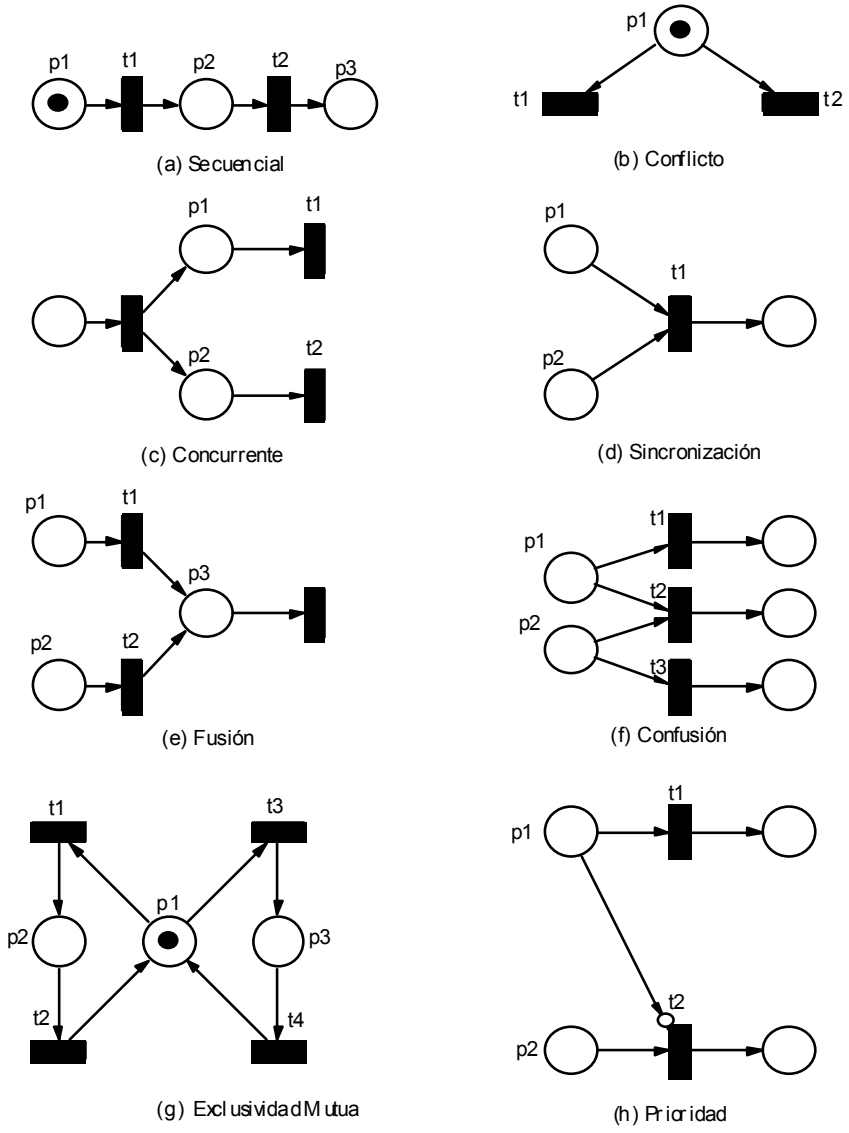

Figura 3.3: Primitivas de redes de Petri para representar características de los sistemas a modelar.

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, t_2 y t_3 también están en conflicto.

7. *Exclusividad mutua.* Dos procesos son mutuamente exclusivos si no pueden llevarse 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 clásicas no tienen mecanismos para representar prioridades, pero la implementación de prioridad en una PN se logra con el uso de un *arco inhibidor*. El arco inhibidor conecta un lugar de entrada a una transición y gráficamente se representa por un arco que termina con un círculo pequeño. La presencia de un arco inhibidor conectando a un lugar de entrada con una transición cambia completamente las condiciones para activar a la transición. En la presencia de un arco inhibidor, una transición se considera como activa 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 se activa si p_1 contiene un token, mientras t_2 es activada si p_2 contiene un token y p_1 no tiene tokens. Esto le da una mayor prioridad a t_1 sobre t_2.

3.4. Propiedades de las redes de Petri

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 [21]. A contin-

uación se da una descripción de algunas de las propiedades de comportamiento más importantes: alcanzabilidad (reachability), limitada (boundedness), acotamiento (conservative) y viva (liveness).

Alcanzabilidad

Una cuestión importante en el diseño de DEDS es saber si un sistema alcanza 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.[21]

Para detectar si el sistema modelado alcanza 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. Es notorio que un sistema real puede alcanzar un estado dado como resultado de manifestar diferentes patrones permisibles de comportamiento funcional, los cuales pueden transformar la marca M_0 en la marca deseada M_i. La existencia de secuencias adicionales de disparo de reglas que transforman a M_0 en M_i significa que el modelo de PN no refleja exactamente la estructura y el comportamiento del sistema que se está modelando. Esto también puede indicar la presencia de facetas del comportamiento funcional del sistema real que no fueron previstas, dado que el modelo de PN refleja exactamente las especificaciones requeridas del sistema real. 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 activada en M_0 da como resultado la marca M_1. [20]

Consideremos un sistema de manufactura sencillo, el cual está compuesto de dos máquinas, la máquina M_A y la M_B. Dos tipos de piezas, la tipo A y la tipo B, son procesados por M_A y posteriormente ensambladas por M_B. El modelo de PN para este sistema es mostrado en la figura 3.4. La descripción de los lugares y las transiciones se dan en la tabla 3.2. [20]

La marca inicial de este modelo de PN es $M_0 = (1\ 1\ 1\ 0\ 0\ 0\ 0\ 0)^T$. La marca en la cual la pieza de tipo A espera para procesarse en M_A, M_A está procesando una pieza de tipo B y M_B está desocupada se representa por $M_i = (1\ 0\ 0\ 0\ 1\ 0\ 0\ 0)^T$. M_i es alcanzable desde el estado inicial disparando t_2. La marca $(1\ 0\ 0\ 0\ 0\ 0\ 1\ 0)^T$, que indica el estado en el que una pieza de tipo A está

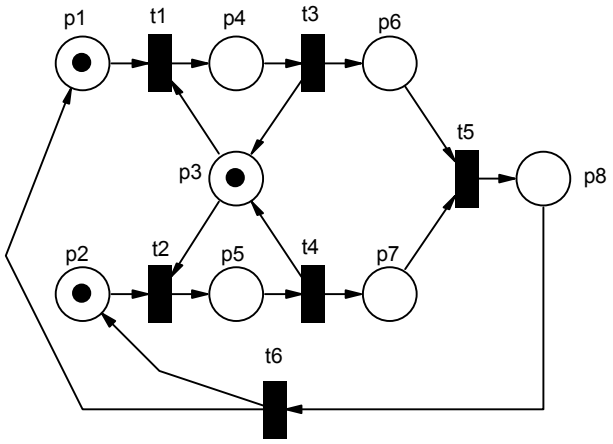

Figura 3.4: Un sistema de manufactura sencillo.

esperando para procesarse en M_A, M_A está desocupada y la pieza de tipo B espera para ensamblarse en M_B, es alcanzada inmediatamente al disparar la transición t_4.

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 se denota 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 se denota 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 [21]. La propiedad de PN que ayuda a identificar la existencia de desbordamientos

Lugar	**Descripción**
p_1	Piezas disponibles de tipo A
p_2	Piezas disponibles de tipo B
p_3	M_A disponible
p_4	M_A procesando piezas de tipo A
p_5	M_A procesando piezas de tipo B
p_6	Piezas disponibles de tipo A para su ensamble
p_7	Piezas disponibles de tipo B para su ensamble
p_8	Pallet listo
Transiciones	**Descripción**
t_1	M_A inicia el procesamiento de la pieza de tipo A
t_2	M_A inicia el procesamiento de la pieza de tipo B
t_3	M_A termina el procesamiento de la pieza de tipo A
t_4	M_A termina el procesamiento de la pieza de tipo B
t_5	M_B ensambla
t_6	Pallet carga piezas

Cuadro 3.2: Etiquetas para la figura 3.4.

en los sistemas modelados es el concepto de *limitado* (*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 3.4 es* 1*-limitada, lo que significa que ambas PN son seguras.*

Acotamiento

Los tokens en una PN representan 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 vuelve importante. [21]

Definición 3.5 *Una* $PN = \{P, T, F, W, M_0\}$ *está estrictamente acotada si para todas* $M \in R(M_0)$, $\sum_{p_i \in P} M(p_i) = C$, *donde* C = *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 las suma de tokens en una red sea siempre un número constante. Esto da como resultado una definición más amplia de esta propiedad:[21]

Definición 3.6 *Se dice que una* $PN = \{P, T, F, W, M_0\}$ *está acotada si existe un vector* $w = (w_1, w_2, \cdots, w_m)$ *donde* m *es el número de lugares, y* $w_i > 0$ *para cada* $p_i \in P$, *tal que* $\sum_{i=1}^{m} w_i M(p_i) = C$, *donde* C = *constante*

La figura 3.5 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 está estrictamente acotada, 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 sólo elemento). Sin embargo, esta PN está acotada con respecto a $w = (1\ 1\ 1\ \ 2\ 2)$. [20]

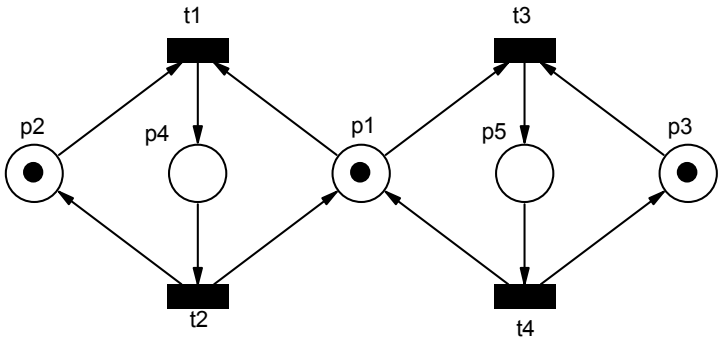

Figura 3.5: Un modelo de red de Petri para un sistema de manufactura, el cual no está acotado.

Viva

El concepto de "viva" está estrechamente relacionado con la situación de *punto muerto* (*deadlock*), el cual ha sido situado en el ámbito de los sistemas operativos. [18]

Una PN que modela un sistema libre de puntos muertos es *viva*. Esto implica que para alguna marca alcanzable M, es posible disparar al menos una transición en la red a través de alguna secuencia de disparo. Sin embargo, este requisito es bastante estricto cuando se representa algún sistema real o escenarios donde se presente un comportamiento libre de puntos muertos. Por ejemplo, la inicialización de un sistema puede modelarse por una transición (o un conjunto de transiciones) las cuales se disparan en un número finito de veces. Una vez iniciada, el sistema manifiesta un comportamiento libre de puntos muertos, aunque la PN que representa a este sistema no tiene el mismo nivel de esta propiedad en diferentes estados. Por esta razón, se describieron diferentes niveles de la propiedad *viva* para una transición t y para una marca M_0, como se muestran a continuación: [21]

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

1. *L0-viva (o muerta) si no hay alguna secuencia de disparo en $L(M_0)$ que pueda disparar a t.*

2. *L1-viva (potencialmente disparable) si t se dispara al menos una vez en alguna secuencia de disparo de $L(M_0)$.*

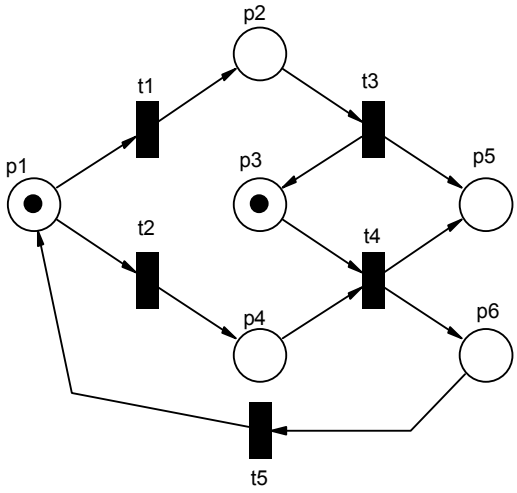

Figura 3.6: Una red de Petri no viva. Pero es estrictamente L1-viva.

3. *L2-viva si t se dispara al menos k veces en alguna secuencia de disparo de* $L(M_0)$.

4. *L3-viva si t se dispara infinitamente en alguna secuencia de disparo de* $L(M_0)$.

5. *L4-viva (o viva) si t es L1-viva (potencialmente disparable) en cada marca de* $L(M_0)$.

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

Ejemplo 3.3 *La PN mostrada en la figura [3.6] es estrictamente L1-viva puesto que cada transición puede dispararse exactamente una vez en el orden de t2, t4, t5, t1 y t3. Las transiciones t1, t2, t3 y t4 en la figura [3.7] son L0-viva (muertas), L1-viva, L2-viva y L3-viva, respectivamente.*

3.5. Métodos de análisis de las redes de Petri

Los métodos para analizar PN pueden clasificarse en los siguientes grupos: 1) el método de árbol de alcanzabilidad, 2) el enfoque de ecuación de matrices, 3) la técnica de simplificación de

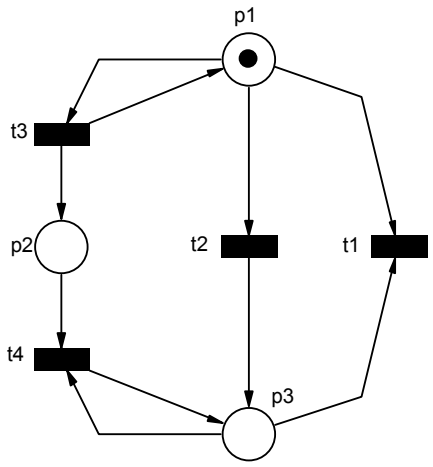

Figura 3.7: Las transiciones $t1$, $t2$, $t3$ y $t4$ son transición muerta ($L0$-viva), $L1$-viva, $L2$-viva y $L3$-viva, respectivamente.

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. [21]

El árbol de alcanzabilidad

Dada una PN, desde su marca inicial M_0, podemos obtener igual número de marcas "nuevas" como el número de transiciones que se encuentren activadas. 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 $\omega >= \omega$.

El árbol de alcanzabilidad para una PN se construye a partir del siguiente algoritmo: [21]

1. Etiquetar la marca inicial M_0 como la raíz del árbol y marcarla como "nueva".
2. Mientras existan marcas "nuevas", hacer lo siguiente:
   - *a*) Seleccionar una marca nueva M.
   - *b*) Si M es idéntica a una marca en la ruta desde la raíz hasta M, entonces marcar a M como "vieja" e ir a otra marca nueva.
   - *c*) Si la marca M no tiene transiciones activadas, rotular a M como "nodo muerto".
   - *d*) Mientras existan transiciones activadas en M, para cada transición activada t en M hacer lo siguiente:
     1) Obtener la marca $M\prime$ que resulta del disparo de t en M;
     2) En la ruta desde la raíz hasta M si existe una marca M" tal que $M\prime(p) \geq M$"(p) para cada lugar p y $M\prime \neq M$", es decir, M" está contenido en $M\prime$, entonces reemplazar $M\prime(p)$ por ω para cada p tal que $M\prime(p) > M$"(p).
     3) Incluir $M\prime$ como un nodo, dibujar un arco con la etiqueta t desde M a $M\prime$ y marcar a $M\prime$ como "nueva".

Matriz de incidencia y ecuación de estado

El comportamiento dinámico de muchos sistemas estudiados en ingeniería es 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 que 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. [21]

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:

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

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 toma lugar. La transición t_i se activa en una marca M si y sólo sí

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

Ecuación de estado: 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:

$M_k = M_{k-1} + A_{u_k}^{T}$, $k = 1, 2, \cdots$

Análisis de invariancia

Los arcos describen las relaciones entre lugares y transiciones, los cuales 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, encontramos 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. [21]

Matemáticamente, representemos una matriz $A = \{a_{ij}^{-}\} - \{a_{ij}^{+}\}$ como una matriz de incidencia. Entonces, la regla de ejecución la podemos describir como la siguiente ecuación de estado:

$M_k = M_{k-1} + Au_k$, $k = 1, 2, \cdots$

Donde M_k es una marca alcanzable inmediatamente desde M_{k-1}, u_k es un vector columna $s \times 1$ solamente con una entrada de un 1 y el resto 0, llamado el vector del k-ésimo disparo. Si

la transición t_i dispara en el k-ésimo disparo, entonces la i-ésima posición de u_k es 1 y las otras posiciones tienen 0. La i-ésima columna de A representa un cambio en la marca como resultado del disparo de la transición t_i.

La solución entera positiva x de $A^T x = 0$ es llamada una *P-invariante*. Multiplicando una $P-invariante$ transpuesta x^T en ambos lados de la igualdad en la ecuación de estado, obtenemos lo siguiente:

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

Dado que $A^T x = 0$, entonces $x^T A = 0$,

$x^T M_k = x^T M_{k-1}$, $k = 1, 2, ...$

Por lo tanto, $x^T M_k = x^T M_0 = constantes$.

Los *P-invariantes* se explican intuitivamente de la siguiente manera. Las entradas diferentes de cero en un *P-invariante* representan el peso asociado con los lugares correspondientes, así que la suma de los tokens en estos lugares es constante para todas las marcas alcanzables desde una marca inicial. Decimos que estos lugares son cubiertos por un *P-invariante* representado por $\|x\|$.

La solución entera y de $Ay = 0$ es llamada una *T-invariante*. Supongamos que el disparo de una secuencia de transiciones lleva a la marca M_0 de regreso a M_0. Y que el i-ésimo elemento del vector de disparo u sea el número de veces que t_i se dispara. El vector u también es llamado el vector contador de disparos. Entonces,

$M_0 = M_0 + Au$

Claramente podemos observar que $Cu = 0$ y u es una *T-invariante*.

Las entradas diferentes de cero en una *T-invariante* representan la cantidad de disparos de las transiciones en la secuencia de disparo que convierte una marca M_0 de nuevo a la marca M_0. Una *T-invariante* representa a todas las transiciones que forman la secuencia de disparo que convierte a $M0$ de nuevo a la misma marca M_0 y el número de veces que esta transición aparece en la secuencia. Sin embargo, no se especifica el orden de disparo de las transiciones.

El hecho de encontrar invariantes ayuda en el análisis de la PN para algunas de sus propiedades. Por ejemplo, si cada lugar en una red es cubierto por un *P-invariante*, entonces este lugar está limitado a un número de tokens. Sin embargo, este enfoque es de uso limitado porque el análisis de invariancia no incluye toda la información de una PN general. Esto sólo se aplica a PN ordinarias.

Los *P-invariantes* y las *T-invariantes* se obtienen resolviendo las ecuaciones lineales:

$A^T x = 0$ y $Cy = 0$

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.[19]

No es difícil de ver que las seis operaciones siguientes conservan las propiedades de PN de que es viva, segura y limitada. Esto es, dadas PN y PN' que representan a la PN antes y después de una de las siguientes transformaciones, entonces PN' es viva, segura o limitada si y sólo sí PN es viva, segura y limitada, respectivamente.

1. Fusión de lugares en serie, como se representa en la figura 3.8(a).
2. Fusión de transiciones en serie, como se representa en la figura 3.8(b).
3. Fusión de lugares en paralelo, como se representa en la figura 3.8(c).
4. Fusión de transiciones en paralelo, como se representa en la figura 3.8(d).
5. Eliminación de lugares en ciclos, como se representa en la figura 3.8(e).
6. Eliminación de transiciones en ciclos, como se representa en la figura 3.8(f).

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 activadas, 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 activada o se establezca un criterio de parada. De esta manera podemos checar el comportamiento que sigue el sistema al establecer ciertas condiciones de en la marca de inicio.

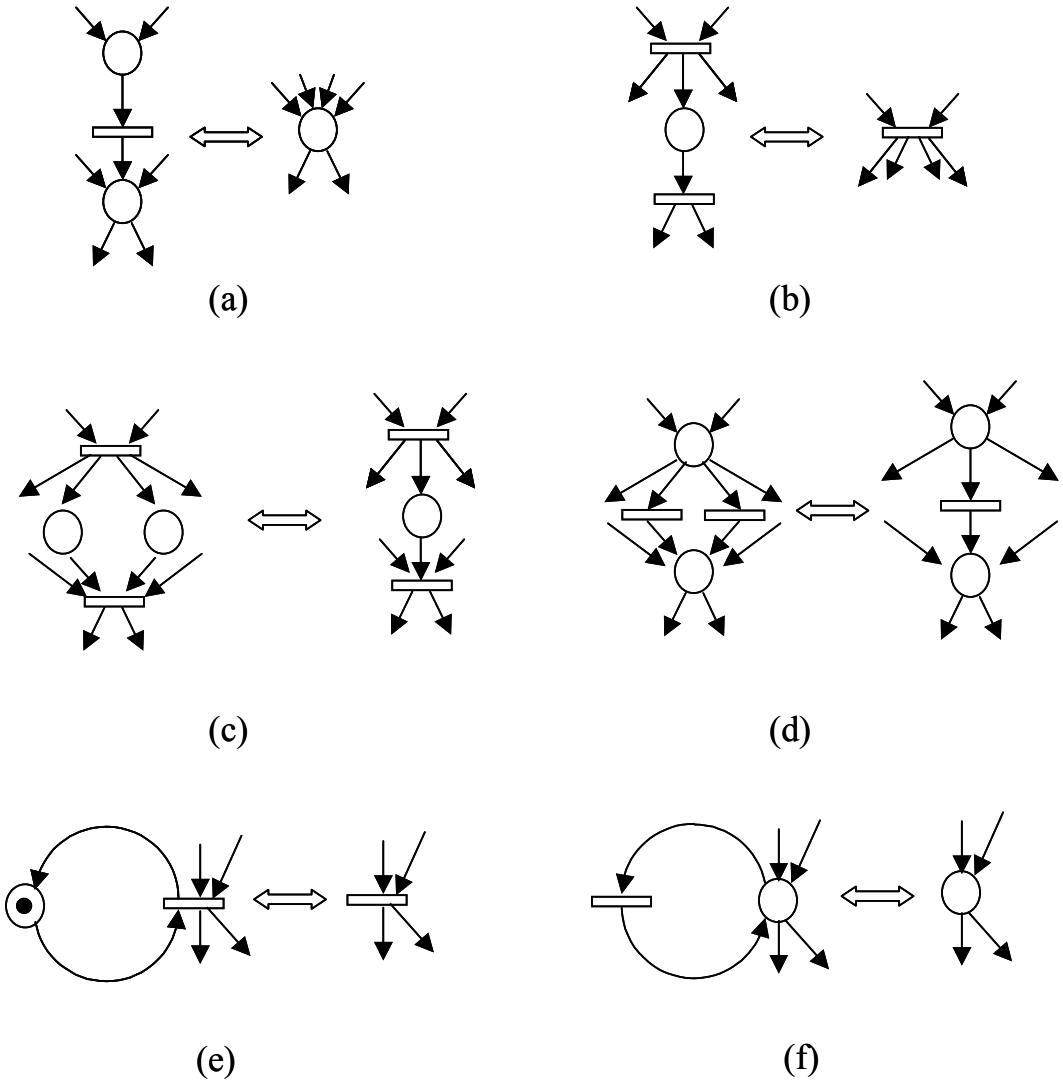

Figura 3.8: Seis transformaciones en redes de Petri que conservan las propiedades de viva, segura y limitada.

3.6. Extensiones de las redes de Petri

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 ú 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.

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 tercera 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.

A continuación se describen los tipos de PN que son de interés para nuestra investigación.

Red de Petri Híbrida

Las PN híbridas son una representación unificada de variables continuas, representadas por cantidades de tokens de lugares continuos (números reales) y de variables discretas, representadas por cantidades de tokens de lugares discretos (números enteros).[22]

Las PN híbridas son adecuadas para modelar la secuencia de pasos en el flujo continuo de material con un control discreto básico. Estas PN fueron extendidas para representar sistemas de transporte basados en cintas transportadoras.

Un lugar continuo está representado por un círculo doble y una transición continua por una barra doble o un rectángulo con fondo negro. Decimos que un lugar continuo pi es alimentado si al menos una de sus transiciones continuas se dispara. Una transición continua tj está activada

fuertemente si la cantidad de tokens de todos sus lugares de entrada es estrictamente positiva. Una transición continua tj está débilmente activada si al menos uno de sus lugares de entrada es alimentado y la cantidad de tokens de los otros son estrictamente positiva. Además, una transición continua se dispara con una velocidad $v(t)$. Esto significa que entre t y $t + dt$ una cantidad $v(t) \cdot dt$ de tokens se elimina de sus lugares de entrada continuos y se agrega a la cantidad de tokens de sus lugares de salida continuos. [22]

Para mostrar un ejemplo de una PN híbrida consideremos un sistema por lotes, donde el material sin refinar está fluyendo en cantidades finitas a través del equipo, las operaciones más frecuentes que se realizan en los recipientes son llenando (cargando) o vaciando (descargando) un reactor. El inicio y término de este tipo de operaciones son eventos discretos que son comunes en los formularios. Un problema que se presenta al representar el comportamiento de un sistema por lotes es que los lotes tienen que ser precisamente identificados tanto en el tiempo como en el espacio (el conducto en el cual el lote está ubicado tiene que ser conocido en cada instante del tiempo). La utilización de una PN es adecuada porque cada lote es representado por un token en un lugar correspondiente a un estado del conducto (o configuración). El otro problema es que, para calcular el balance, el nivel exacto de material dentro de los conductos tiene que ser conocido, en cada instante de tiempo, e independientemente del hecho de que sea el resultado de la presencia de algún número de lotes (o algún fragmento de un lote).

El reactor se encuentra representado en la figura 3.9, la capacidad del volumen de material que soporta el reactor se denota por V, el flujo de entrada por qi, y el flujo de salida por qo[24]. El modelo de PN híbrida para representar a este ejemplo se muestra en la figura 3.10.

La velocidad del disparo de la transición t_3 representa el flujo de entrada qi y la velocidad del disparo de la transición t_4 representa el flujo salida qo. El número de tokens en el lugar continuo p_3 (un número real) representa el volumen V de material en el conducto. El proceso de funcionamiento del reactor está representado con los lugares y transiciones discretos de la PN: entre el disparo de t_1 y el de t_2, el conducto se llena. La interacción de la parte discreta dentro de la parte continua es denotada por el ciclo entre p_2 y t_3. Cuando hay un token en p_2, la velocidad de disparo de t_3 es qi, de otra manera t_3 no está activada y no puede dispararse. La interacción de la parte continua dentro de la parte discreta es denotada por el ciclo entre p_3 y t_2. Cuando la cantidad de tokens en p_3 es $V = V_{alto} = 10001$, entonces t_2 se dispara y el token en p_2 se elimina. Por lo tanto, la

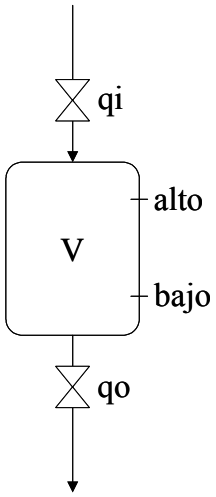

Figura 3.9: El reactor.

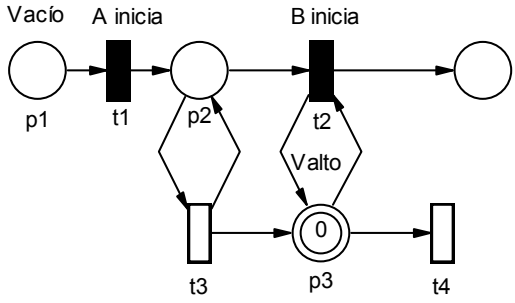

Figura 3.10: Modelo de red de Petri híbrida.

operación de llenado se detiene. [24]

En este modelo, la parte discreta está representada por lugares y transiciones discretos, y la parte continua está representada por lugares y transiciones continuas. Este modelo sólo describe valores reales no negativos (variables continuas); además, un lugar continuo representa solamente a una variable continua. La única manera de describir la interacción entre la parte discreta y la parte continua es a través de ciclos en la red.

En PN híbridas sencillas, solamente las variables continuas que son lineales con respecto al tiempo pueden representarse (velocidades constantes). Y para representar otro tipo de evolución en la red, es necesario introducir otros tipos de lugares continuos.

Red de Petri Coloreada

La PN 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.

Las CPN se llaman así porque permiten el uso de tokens que contienen valores y cada token se diferente a 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 se dispare, debe de tener suficientes tokens 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 se dibujan como rectángulos, los cuales representan las acciones que se llevan a cabo. Y las expresiones de arco describen que es lo que ocurre cuando una transición se dispara. [25]

Un pequeño ejemplo de CPN se muestra en la figura 3.11, 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 3.11) 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 enviarse. 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 envíado/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 bufers de entrada y salida en el proceso de transmisión de la red. [25]

La ventaja principal de este tipo de PN es que, a diferencia de las otras, maneja datos 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 BD.

Red de Petri Difusa

En muchas situaciones, los sistemas necesitan capturar información que se presenta de manera ambigua y con las herramientas que se presentan en la teoría de PN es imposible considerarlas. Por ello, se desarrolló una extensión de PN llamada PN difusa, la cual es capaz de almacenar información incierta. Las PN difusas son una herramienta de modelación muy prometedora en el desarrollo de sistemas expertos.[26]

Se agregaron características de las reglas de producción difusas a los elementos de la PN para almacenarlas. De esta manera se representa conocimiento dentro de una PN difusa. Una regla de producción difusa describe la relación difusa entre dos proposiciones. Si la parte del antecedente de una regla de producción difusa tiene conectivos "AND" ú "OR" se trata de una regla de producción difusa compuesta.

Hay muchas aplicaciones de PN difusas, por ejemplo, la representación de conocimiento puede

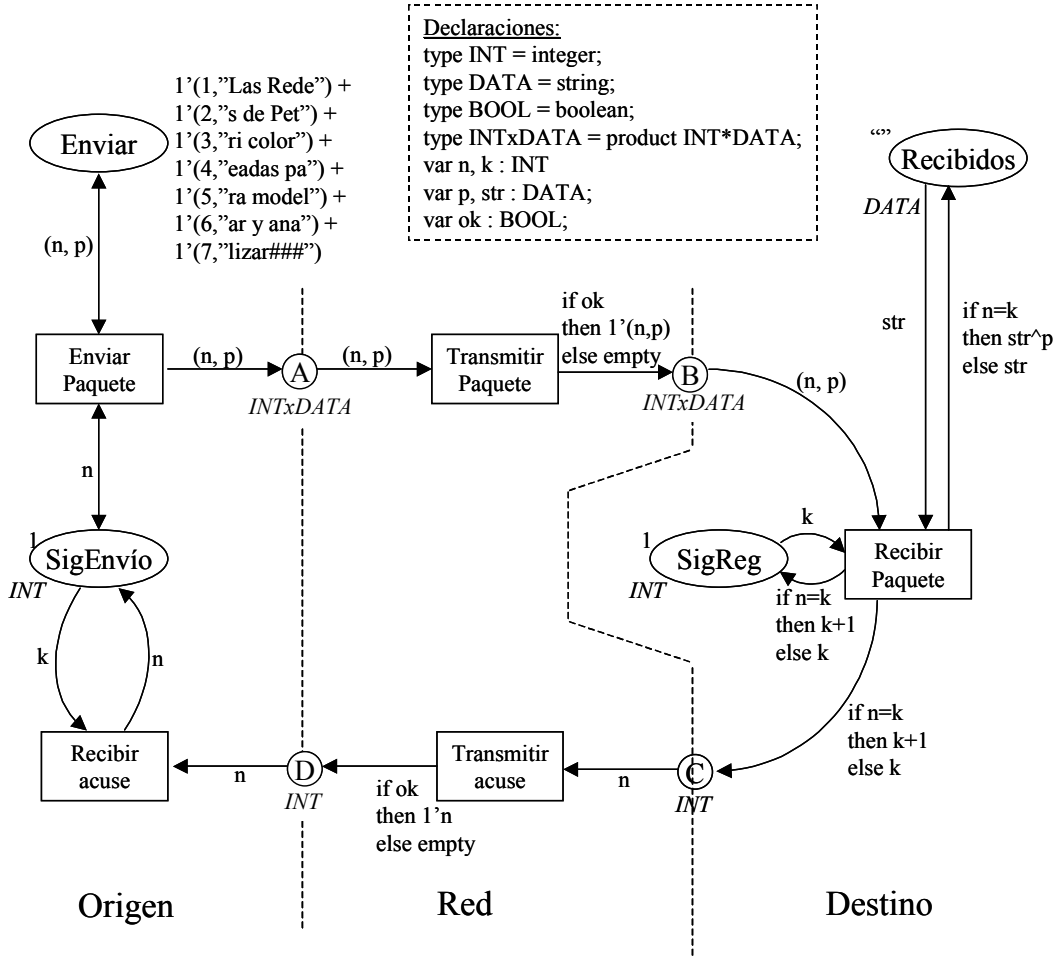

Figura 3.11: Ejemplo de una red de Petri Coloreada.

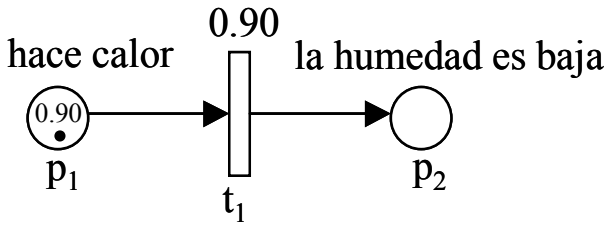

Figura 3.12: Representación de conocimiento con una red de Petri difusa.

implementarse a través de las PN difusas, como se describen en [27], donde las reglas de producción difusa son transformadas en PN difusas. El disparo de las reglas de producción difusa se considera como el disparo de una transición. En [28] los autores proponen un modelo de PN difusa para clasificar el conjunto de datos acerca del iris. Además, las PN difusas se utilizan como una herramienta para procesos inteligentes de monitoreo y control [29]. En [30] las PN difusas son combinadas con redes neuronales para modelar la calidad de los productos de un centro de maquinaria para molinos-CNC. En [31] utilizan un enfoque de PN difusas, para el manejo de incertidumbre en sistemas de robots. En [32] se presenta un modelo de PN difusa para razonamiento aproximado y para probar este método lo aplican a diagnósticos médicos. Entre otras tantas aplicaciones que actualmente existen en la literatura de PN.

Como un ejemplo sencillo tenemos la siguiente regla de producción difusa R1: Si el clima está caliente Entonces la humedad es baja (CF = 0.90, CF: factor de certeza), es convertida en la PN difusa mostrada en la figura 3.12.

3.7. Comentarios finales

Las PN son una herramienta poderosa para la modelación y simulación de sistemas manejados por eventos. De manera gráfica y visual se puede analizar el comportamiento del sistema modelado. Posee las propiedades que ayudan en la detección de inconsistencias de los sistemas, las cuales pueden llevar al sistema un estado de "punto muerto". Para realizar el análisis de la PN se cuentan con métodos como el árbol de alcanzabilidad, la matriz de incidencia y la ecuación de estado,

el análisis de la invariancia de la red, reglas para reducir PN grandes y la simulación del sistema modelado. Las PN pueden extenderse para cubrir al máximo sistemas que son difíciles de representar con PN tradicionales, existiendo actualmente un gran número de extensiones de PN.

Capítulo 4

Red de Petri Coloreada Condicional (CCPN)

Las PN se utilizan para modelar sistemas concurrentes, asíncronos, distribuidos, paralelos indeterministas y/o asíncronos [19]. Sin embargo, existen sistemas cuyas características particulares no permiten llevar a cabo una representación adecuada con PN. Para lograr adaptar la teoría de PN a sistemas particulares es necesario agregar la descripción de las características propias de estos sistemas dentro de la definición formal de PN, obteniendo como resultado extensiones del modelo original, como las mencionadas en el capítulo anterior.

Las PN son una herramienta ampliamente utilizada en la modelación de sistemas manejados por eventos (DES, Driven Event Systems), donde los lugares y las transiciones de la PN representan el estado que guarda el sistema en un momento determinado y la ejecución de una acción ante la presencia de un evento, respectivamente. Al igual que los DES, las reglas ECA reaccionan cuando sucede un evento de interés, entonces una base de reglas ECA puede considerarse como un DES. Pero no todas las características de una regla ECA son cubiertas por una PN tradicional. En este trabajo de investigación proponemos una PN extendida, la CCPN, en la cual se contemplan las características de cada uno de los elementos de la regla ECA.

En la sección 4.1 se presentan las diferentes sintaxis que ofrecen las BDA para declarar reglas activas, así como una sintaxis propuesta para llevar a cabo la conversión a una CCPN. En la sección 4.2 se describe ampliamente la manera en que se combinan los aspectos teóricos de las reglas ECA

y de PN, para obtener una versión extendida de PN (la CCPN) y representar adecuadamente reglas ECA. La sección 4.3 presenta una definición formal de la CCPN, describiendo la relación existente entre sus elementos. La regla de disparo de transiciones de la CCPN se presenta en la sección 4.4. El algoritmo de conversión de una base de reglas ECA a una CCPN se describe en la sección 4.5. En la sección 4.6 se presentan ejemplos de aplicación del algoritmo de la sección 4.5. En la sección 4.7 se hacen unos comentarios al respecto de éste capítulo.

4.1. Definición de reglas ECA

La manera en que se definen las reglas ECA varía entre los diferentes sistemas activos de BD. Su forma general de definición es [16]:

```
ON evento
IF condición
THEN acción
```

En SQL:99 la definición de las reglas ECA se hace mediante el uso de "triggers". Un "trigger" es un comando que supervisa el estado de la BD y realiza una acción cuando un evento se hace presente. La sintaxis para declarar un "trigger" en SQL:99 es la siguiente [33]:

```
<definición de ''trigger''> ::=
    CREATE TRIGGER <nombre del ''trigger''>
    { BEFORE | AFTER } <evento del ''trigger''> ON <nombre de la tabla>
        REFERENCING <lista de alias de valores anteriores o nuevos> ]
        FOR EACH {ROW | STATEMENT} ]
        <condición del ''trigger''> ]
    <acción del ''trigger''>


<evento del ''trigger''> ::= INSERT | DELETE
                    | UPDATE [ OF <lista de nombre de campos> ]
<alias de valores anteriores o nuevos> ::= {OLD | NEW} [AS] <identificador>
                     | {OLD_TABLE | NEW_TABLE} [AS] <identificador>
```

Los comandos que se encuentran entre corchetes son opcionales, los que están entre llaves debe existir uno de ellos y los que están entre "<" y ">" es obligatorio que existan. En ésta definición, se especifica si el evento se detecta antes o después de su ejecución en la BD, así como la tabla donde se detecta el evento. La parte condicional es opcional, es decir, con "triggers" de SQL:99 es posible declarar regla ECA y EA. Los "triggers" de SQL:99 sólo ejecutan una sola acción al dispararse.

El sistema de reglas Starburst tiene una manera diferente de definición de reglas. La sintaxis es la siguiente:

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

En esta definición de regla, se declara el nombre de la regla y la tabla que se vigilará para la detección del evento activador de la regla. Los eventos se especifican en la cláusula "when". Al igual que SQL:99 la parte condicional es opcional y, en consecuencia, es posible definir reglas ECA y EA. A diferencia de SQL:99, las reglas definidas en Starburst ejecutan una serie de acciones. Las opciones "precedes" y "follows" se utilizan para establecer prioridades entre las reglas cuando se disparan al mismo tiempo.

En el lenguaje de reglas de CDOL (Comprehensive, Declarative, Object Language), además del evento, condición y acción, se especifica el modo de acoplamiento entre las parejas evento-condición y condición-acción [34]. El modo de acoplamiento se refiere a la prioridad con la cual se considera la condición (acción) cuando se presenta el evento (condición). CDOL ofrece dos modos de acoplamiento: inmediato y pospuesto. A continuación se muestra la definición de una regla en CDOL:

```
active amphibspeed_restriction
{
```

```
event after amphibvehicle:TheAmphib[set_speed(integer:New_speed)]
condition deferred 10 < New_speed
action deferred amphibvehicle:TheAmphib.set_speed(10);};
```

En [35] se persigue la forma general de definición de reglas ECA, pero en un orden diferente. Para definir una regla ECA se especifica primero el evento, posteriormente se menciona el nombre de la regla que será disparada, luego se describe la condición a evaluar y al final la acción que se realizará. Un ejemplo de la definición de una regla se muestra a continuación:

```
Rule r1
WHEN (e1 : Proyecto modificado)
FIRE r1 (''Política para reducción de presupuesto'') :
IF (NEW presupuesto < OLD presupuesto) THEN
DELETE.ENTITY EMPLEADO (num_imss = OLD EMPLEADO.num_imss OF TRABAJO
(Proyecto.nombre = OLD.nombre), categoría = ''temporal''} (e2)
RAISE (e3. Salario.revisión)
```

La BDA Ariel utiliza un lenguaje de definición de reglas llamado ARL (Ariel Rule Language). La sintaxis de ARL se basa en la sintaxis del lenguaje de consulta de Ariel. La forma general de definición de reglas en ARL es [3]:

```
define rule nombre in conjunto_de_reglas
priority valor_prioridad
on evento
if condición
then acción
```

Las reglas de Ariel manejan la prioridad de disparo de las reglas. El evento y la condición son opcionales, pero al menos uno de ellos debe estar presente en la definición de una regla, es decir, ARL define reglas de tipo ECA, EA y CA. Al igual que Starburst, la acción de Ariel se forma por uno o más comandos.

Postgres utiliza los comandos del lenguaje de consulta en la definición de reglas activas. Al igual que los sistemas anteriores [33] [34] [35] [3], también persigue la forma general de definición de una regla ECA [16]. La sintaxis de Postgres para la definición de reglas es la siguiente [6]:

```
CREATE RULE nombre AS ON evento
TO objeto
[WHERE condición]
DO [INSTEAD] [acción | NOTHING]
```

En esta declaración, el evento es un comando "insert", "update", "delete" o "select" de SQL. La variable `objeto` es una tabla o el campo de una tabla. La condición es opcional, por lo tanto, es posible declarar reglas ECA y EA. En la parte de la acción puede especificarse que se haga otra transacción en la BD en lugar del evento detectado, la acción en sí o que no se haga nada.

Los sistemas presentados ofrecen su propia sintaxis para la definición de reglas. En éste trabajo de investigación también se propone una forma diferente de definición de reglas ECA, persiguiendo su forma general. La CCPN se genera a partir de la definición de reglas ECA. La definición de reglas que se propone se ajusta a las necesidades de la CCPN, considerando solamente los elementos principales de la regla (evento, condición y acción), dejando para trabajos posteriores características como la prioridad, modos de acoplamiento, entre otras. La sintaxis propuesta es la siguiente:

```
on evento1 [ | evento2 [ | evento3 [ ··· ] ] ]
if condición1 [ & condición2 [ & condición3 [ ··· ] ] ]
then acción
```

En la primera parte de la regla se especifica el ó los eventos que activan la regla. Con la presencia solamente de uno de ellos la regla se activa. Para la descripción del evento se utiliza la misma nomenclatura aplicada a los lugares de la CCPN. La segunda parte de la regla, la condición, acepta expresiones lógicas unidas por el conectivo lógico "Y", representado por el carácter '&'. La tercera parte de la regla, la acción, describe el movimiento que se realizará en la BD al dispararse la regla. La acción se define en términos del lenguaje de consulta SQL.

Por ejemplo, la regla activa: "Cuando se agrega un registro a la tabla CUENTA, si el saldo es menor que $500.00 y el interés de la cuenta es mayor que 0 %, entonces el interés se reduce a 0 %",

aplicada a la tabla CUENTA(número, nombre, saldo, interés), se convierte a una regla ECA con nuestra sintaxis de la siguiente manera:

```
ON insert_CUENTA
IF saldo<500.00 and interés>0.0
THEN update CUENTA set interés=0.0 where saldo<500.00 and interés>0.0;
```

4.2. Representación de una regla ECA con CCPN

Una regla ECA se activa cuando se presenta el evento que está vigilando. Con la activación de la regla, la parte condicional verifica si se cumple contra los datos del evento detectado. Si la condición se cumple, entonces la regla ECA se dispara y se ejecuta la acción. Para mostrar la diferencia entre activar y disparar una regla, aclaramos lo siguiente: se dice que una regla se activa cuando el evento denotado se hace presente y una regla se dispara cuando se activa y, además, la evaluación de la condición da como resultado un valor verdadero.

El evento de la regla ECA puede representarse como un lugar de entrada de una PN. Los eventos que causan un cambio de estado en la BD son aquellos producidos por los comandos de SQL (Standard Query Language) "INSERT", "UPDATE" y "DELETE". Estos comandos modifican el estado de cada una de las tablas que conforman a la BD y en el caso particular del comando "UPDATE" modifica el estado de uno o varios campos de la tabla.

Clasificamos a los eventos en tres categorías: agregar, modificar y eliminar registros en tablas de la BD. Para particularizar aún mas a los eventos, además de la clasificación por comandos, los clasificamos de acuerdo a la tabla de la BD que se afecta con la ejecución del comando SQL.

La nomenclatura de los lugares de CCPN se compone por el nombre del comando SQL, seguido por el nombre de la tabla donde se ejecuta, separados por un guión bajo. Por ejemplo, el nombre para un lugar que representa el evento de agregar un registro a una tabla llamada "EMPLEADO" sería "INSERT_EMPLEADO". En el caso del comando "UPDATE", el nombre del lugar contiene, además, el nombre del campo que se modifica. Por ejemplo, el nombre del lugar que representa el evento de modificar el campo "SALARIO" de la tabla "EMPLEADO" sería "UPDATE_EMPLEADO_SALARIO".

Si el lugar cuenta con la presencia de un token indica la aparición del evento. Este token, al igual que el token de la CPN, almacena información. La información que contiene es alusiva al

evento que se detectó. Como se mencionó anteriormente, los comandos que provocan un cambio de estado en la BD son "INSERT", "UPDATE" y "DELETE". La sintaxis de éstos comandos es la siguiente:

```
insert into TABLA values (CAMPO1, CAMPO2, ...);
update TABLA set CAMPO = VALOR where CONDICION;
delete from TABLA where CONDICION;
```

Las palabras escritas con letras mayúsculas (TABLA, CAMPO, VALOR y CONDICION) son variables que toman valores de acuerdo a la instrucción deseada. Por ejemplo, si tenemos una tabla "EMPLEADO" con la siguiente estructura:

```
EMPLEADO(nombre, edad, salario)
```

Y queremos agregar al empleado Juan Pérez, de 30 años de edad y con un salario de $5,000.00, en la sintaxis de "INSERT" sustituimos la palabra TABLA por EMPLEADO y (CAMPO1, CAMPO2, ...) por ('Juan Pérez', 30, 5000.00), quedando la instrucción de la siguiente manera:

```
insert into EMPLEADO values ('Juan Pérez', 30, 5000.00);
```

En esta instrucción podemos observar la palabra "insert" y "EMPLEADO", con las cuales identificamos el lugar (INSERT_EMPLEADO) que representa a éste evento. El valor de los campos ('Juan Pérez', 30, 5000.00) se almacena en el token que se coloca en el lugar INSERT_EMPLEADO para manifestar la aparición del evento.

La estructura del token depende del evento. Cuando el evento se genera por una instrucción "INSERT", el token contiene los mismos campos de la estructura de la tabla donde se suscita el evento. Si agregamos un registro en la tabla EMPLEADO los campos del token serán "nombre", "edad" y "salario".

El token de un lugar que representa la modificación ("UPDATE") de un registro tiene dos campos: en el primero de ellos, denominado "SET", se almacena el nombre del campo de la tabla que se está modificando y el valor que se le está asignando; en el segundo , denominado "WHERE", se guarda la parte de la instrucción SQL precedida por la palabra "where", inclusive. Por ejemplo, si el registro del ejemplo anterior se modifica cambiando el salario de Juan Pérez a $6,000.00, la instrucción SQL sería la siguiente:

```
update EMPLEADO set salario=6000.00 where nombre='Juan Pérez';
```

El token correspondiente a éste evento contendrá en el campo "SET" el texto "salario=6000.00"

y el campo "WHERE" el texto "where nombre='Juan Pérez';".

La estructura del token que representa la eliminación de un registro ("DELETE") tiene un solo campo denominado "WHERE", donde se guarda parte de la instrucción SQL, a partir de la palabra "where" y hasta el final de la instrucción. Por ejemplo, si eliminamos el registro modificado en el ejemplo anterior, la instrucción SQL sería:

```
delete from EMPLEADO where nombre='Juan Pérez';
```

El campo "WHERE" del token almacenará el texto "where nombre='Juan Pérez';"

No es necesario almacenar, en la estructura del token, el nombre de la tabla de la BD que se modifica ni el comando SQL que se ejecuta ("insert", "update" ó "delete"), porque éstos datos vienen dados por el lugar que representa al evento y es donde se colocará el token.

Para que la transición de una PN se dispare se debe cumplir que el número de tokens del lugar de entrada sea igual o mayor que el peso del arco. Una regla ECA se dispara si se presenta el evento que se está vigilando (la presencia del token en el lugar de entrada) y la evaluación de la condición da como resultado un valor verdadero. La transición de una PN no tiene la capacidad de realizar la evaluación de expresiones lógicas, solamente evalúa la presencia o ausencia de tokens en sus lugares de entrada. Pero, si agregamos a la transición la capacidad de evaluar expresiones lógicas, entonces la parte condicional de la regla puede representarse como una transición en una PN.

Necesitamos dos tipos de transiciones: la transición-regla y la transición-copy. En la transición-regla las expresiones lógicas son evaluadas con la información contenida en el token del lugar de entrada. Los tokens provenientes de lugares de entrada de tipo "INSERT" contienen la información de todos los campos del registro y la evaluación de la expresión se realiza sin ningún problema. Por ejemplo, una BD contiene la tabla:

```
CUENTA(número, nombre, saldo, interés)
```

Y cada vez que se le agrega un registro se evalúa la expresión "si el saldo es menor que $500.00 y el interés es mayor que 0.0 por ciento". Como en el token se almacenan los valores de los campos "saldo" e "interés" se realiza la verificación de la expresión lógica con éstos datos para determinar si la evaluación da como resultado un valor verdadero o falso. Cuando agregamos el registro (15853, 'Francisco Medina', 5000.00, 2.5) se toman los datos 5000.00 y 2.5 correspondientes a los campos "saldo" e "interés", respectivamente. Fácilmente se observa que el resultado de la evaluación es un valor falso, ya que 5000.00 es mayor que 500.00 y no menor, como se especifica en la condición

"saldo < 500.00".

Los tokens originarios de los lugares de entrada de tipo "UPDATE" contienen información del valor que está actualizándose y la condición "where" de la instrucción SQL. Cuando la expresión lógica solamente verifica el valor de la variable que está actualizándose no hay problema alguno, puesto que la evaluación se realiza con el valor contenido en el token. Sin embargo, si la expresión lógica evalúa, además, un campo diferente al que está modificándose, entonces se obtiene este valor de la BD. Por ejemplo, supongamos que la expresión lógica del ejemplo anterior se evalúa con un token proveniente de un lugar de entrada de tipo "UPDATE", el cual contiene los datos "saldo=4000.00" y "where nombre='Francisco Medina';" para los campos "SET" y "WHERE", respectivamente. Con la información del campo "SET" se evalúa la condición "saldo < 500.00", obteniendo de esta evaluación un valor falso. Para la evaluación de la condición "interés > 0.0" se obtiene el dato del registro que fue modificado, con la ayuda del campo del token "WHERE", teniendo un valor de 2.5 % de interés, obteniendo como resultado de la evaluación un valor verdadero. La primera condición fue falsa y la segunda verdadera, por lo tanto la evaluación de la expresión completa arroja un valor falso.

Los tokens de los lugares de entrada de tipo "DELETE" contienen la parte condicional "where" con la cual se seleccionan los registros que serán eliminados. Esta misma condición se utiliza para obtener una réplica de los registros antes de eliminarlos. Con el duplicado de cada uno de los registros eliminados se evalúa la expresión lógica, de la misma manera que cuando se trata de un token proveniente de un lugar de entrada de tipo "INSERT". Por ejemplo, la instrucción SQL:

```
delete from CUENTA where nombre='Francisco Medina';
```

Elimina el registro (15853, 'Francisco Medina', 4000.00, 2.5), con el cual se evalúa la sentencia "si el saldo es menor que $500.00 y el interés es mayor que 0.0 por ciento".

Además, en la transición-regla, se almacenan datos para describir la acción que se realizará si la regla se dispara. La transición-regla tiene conocimiento de los datos que se guardarán en el token que se enviará al lugar de salida correspondiente a la acción. Por ejemplo, retomando la tabla CUENTA y la regla:

```
ON insert_CUENTA
IF saldo<500.00 and interés>0.0
THEN update CUENTA set interés=0.0 where saldo<500.00 and interés>0.0;
```

Esta regla se activará cuando se agregue un registro a la tabla CUENTA. Si se cumple, además, que el saldo del registro nuevo es menor que $500.00 y el interés es mayor que 0.0 por ciento, entonces se disparará ejecutándose la acción de modificar el valor del interés a 0.0 por ciento.

En la transición-regla se evalúa la condición "saldo<500.00 and interés>0.0" con los valores del token proveniente del lugar de entrada. Si se cumple la condición se genera un token nuevo de tipo "UPDATE" con los campos SET = "interés=0.0" y WHERE = "where saldo<500.00 and interés>0.0;". El token generado se coloca en el lugar de salida de la transición denominado "UPDATE_CUENTA_INTERES".

El otro tipo de transición, la transición-copy, se utiliza cuando un mismo evento dispara dos reglas diferentes. Cada una de las reglas evalúa su parte condicional de manera independiente, por lo que necesitan la información del token del lugar de entrada por separado. La transición-copy tiene como lugar de entrada el evento que es común en dos o más reglas y sus lugares de salida son los lugares de entrada de cada una de las reglas con el mismo evento. La nomenclatura, para los lugares de salida de ésta transición, se compone del nombre del lugar de entrada concatenado con la palabra "COPY", para especificar que se trata de un duplicado del evento original. La función de la transición-copy es crear copias del token del lugar de entrada y enviarlas a sus lugares de salida, que a su vez, son los lugares de entrada de las reglas con un mismo evento en común.

El lugar de salida (la acción) al igual que el lugar de entrada (el evento) forman parte de un mismo conjunto de elementos que representan cambios de estado en la BD. Es decir, existe la posibilidad de que la acción de una regla sea el evento que active a otra, generando así cadenas de reglas. De esta manera, se crean CCPN donde el lugar de salida de una regla es el lugar de entrada de otra.

4.3. Definición formal de CCPN

Para dar una definición formal de CCPN, primero definimos algunos conceptos que son necesarios para dar una descripción más completa.

Decimos que un **tipo de dato** se compone de uno ó más **elementos** y al conjunto de todos estos elementos se le conoce por el mismo nombre que el del tipo de datos. El **tipo de datos para una variable** v se expresa por $Type(v)$. El **tipo de datos de una expresión** $expr$ se expresa por $Type(expr)$. La notación $Type(v)$ es extendida a $Type(A) = \{Type(v) \mid v \in A\}$, donde A es un

conjunto de variables.

El conjunto de los lugares de entrada $p \in P$ de una transición $t \in T$ se representa por ${}^{\cdot}t$ y el conjunto de los lugares de salida $p \in P$ de una transición $t \in T$ se representa por $t^{\cdot}$:

${}^{\cdot}t = \{p \mid p \in P, t \in T, p \text{ es un lugar de entrada de } t\}$

$t^{\cdot} = \{p \mid p \in P, t \in T, p \text{ es un lugar de salida de } t\}$

Definiremos ahora el concepto de *multi-conjunto*. Durante la ejecución de una CCPN, cada lugar tiene un número variable de tokens. Los tokens almacenan valores que pertenecen a un tipo de dato (color asociado a cada lugar). Por ejemplo, en los siguientes tokens:

1‘(1, ”México”)

1‘(1, ”Acapulco”)

1‘(1, ”Monterrey”)

1‘(1, ”Guadalajara”)

Hay un 1‘ al frente de cada valor, lo cual indica que hay exactamente un token que contiene ese valor. En general, varios tokens pueden tener el mismo valor, entonces tendremos un multi-conjunto de valores como:

1‘(1, ”México”) + 2‘(2, ”Acapulco”) + 1‘(4, ”Guadalajara”)

Donde tenemos un token con valor (1, ”México”), dos tokens con valor (2, ”Acapulco”) y un token con valor (4, ”Guadalajara”). Un multi-conjunto es parecido a un conjunto, con la excepción de que en un multi-conjunto un elemento puede aparecer varias veces. Si agregamos el elemento (2, ”Acapulco”) al conjunto:

{ (1, ”México”), (2, ”Acapulco”), (4, ”Guadalajara”) }

No sucede nada porque el elemento ya existe en el conjunto. Sin embargo, si agregamos el elemento (2, ”Acapulco”) al multi-conjunto:

1‘(1, ”México”) + 1‘(2, ”Acapulco”) + 1‘(4, ”Guadalajara”)

obtendremos un multi-conjunto con cuatro elementos en lugar de tres:

1‘(1, ”México”) + 2‘(2, ”Acapulco”) + 1‘(4, ”Guadalajara”)

Los enteros que preceden al operador ‘ se les llama coeficientes. Para la siguiente definición, $\mathbb{N}$ denota al conjunto de los enteros no negativos.

Definición 4.1 *Un multi-conjunto m, de un conjunto no vacío S, es una función $m \in \mathbb{N}$, a la cual representamos formalmente como la sumatoria:*

$\sum_{s \in S} m(s)'s$.

El conjunto de todos los multi-conjuntos de S se denota por S_{MS}. Los enteros no negativos $\{m(s) \mid s \in S\}$ son los coeficientes del multi-conjunto. $s \in m$ si y sólo sí $m(s) \neq 0$. De manera similar a la teoría de conjuntos, se utiliza el símbolo $\emptyset$ para denotar un multi-conjunto sin elementos.

Formalmente una CCPN se define de la siguiente manera: [38] [39]

Definición 4.2 *Una PN Coloreada Condicional (CCPN) es una 9-tupla:*

$$CCPN = \{\Sigma, P, T, A, N, C, Cond, Action, I\}$$

donde:

Σ *Conjunto finito de tipos de dato, los cuales son soportados por los lugares, llamado conjunto de colores. El color es la estructura que tiene el token acorde al lugar donde se coloca. Los colores utilizados en CCPN están compuestos por:*

1. *La estructura de cada tabla de la BD, especificando el tipo de dato de cada campo, para los lugares de tipo "INSERT".*
2. *Dos cadenas de caracteres, para el color de lugares tipo "UPDATE". O*
3. *Una cadena de caracteres para el color de lugares tipo "DELETE".*

$P = \{p_1, p_2, \ldots, p_m\}$ *es un conjunto finito de lugares. Este conjunto está formado por subconjuntos, es decir,*

$P = P_{insert\_tabla} \cup P_{update\_tabla\_columna} \cup P_{delete\_tabla} \cup P_{copy}$
Donde:
$P_{insert\_tabla} = \{p_{it} \mid p_{it} \in P,\ p_{it}$ *representa un lugar de tipo "INSERT"}*
$P_{update\_tabla\_columna} = \{p_{utc} \mid p_{utc} \in P,\ p_{utc}$ *representa un lugar de tipo "UPDATE"}*
$P_{delete\_tabla} = \{p_{dt} \mid p_{dt} \in P,\ p_{dt}$ *representa un lugar de tipo "DELETE"}*
$P_{copy} = \{p_c \mid p_c \in P,\ p_c$ *representa un lugar de tipo "COPY"}*

$T = \{t_1, t_2, \ldots, t_n\}$ *es un conjunto finito de transiciones. Este conjunto está compuesto por dos subconjuntos, es decir,*

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

Donde:

$T_{regla} = \{t_r \mid t_r \in T$, *$t_r$ representa la transición que almacena la parte condicional de la regla ECA}*

$T_{copy} = \{t_c \mid t_c \in T$, *$t_c$ representa la transición que genera duplicados del token proveniente de su lugar de entrada}*

Las transiciones $t \in T_{regla}$ se representan gráficamente por un rectángulo. Mientras que las transiciones $t \in T_{copy}$ se representan gráficamente por una línea gruesa.

A : *conjunto finito de arcos dirigidos de P a T o de T a P. A está formado por los conjuntos A_{in} y A_{out},*

$A = A_{in} \cup A_{out}$

Donde:

$A_{in} = \{(p,t) \mid p \in P, t \in T, (p,t) \in \{PxT\}\}$

$A_{out} = \{(t,p) \mid t \in T, p \in P, (t,p) \in \{TxP\}\}$

$N : A \rightarrow \{PxT\} \cup \{TxP\}$ *es una función nodo que asocia a cada arco un par ordenado $(p,t) \in A_{in}$ ó $(t,p) \in A_{out}$, donde el primer elemento del par ordenado es el nodo origen y el segundo es el destino.*

$C : P \rightarrow \Sigma$ *es una función color, que relaciona a cada lugar $p \in P$ un tipo de dato $Type(p) \in \Sigma$. Cada token en p debe tener un color igual a $C(p)$.*

$Cond : T_{regla} \rightarrow Bool$, *donde* $Bool = \{falso, verdadero\}$, *es una función condición que evalúa la parte condicional de la regla ECA almacenada en la transición $t \in T_{regla}$, dando como resultado un tipo de dato booleano $b \in Bool$. Es decir,*

$\forall t \in T_{regla} : [Type(Cond(t)) \in Bool]$

$Acción : T_{regla} \rightarrow C(p)_{MS}$ *es una función acción que asocia cada transición $t \in T_{regla}$ con un multi-conjunto de un tipo de dato $C(p)$*, donde p es el lugar de salida de t. *Es decir,*

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

$I : P \rightarrow C(p)_{MS}$ *es una función de inicialización que relaciona cada lugar* $p \in P$ *con un multi-conjunto de un tipo de dato* $C(p)$*. Es decir,*

$\forall p \in P : [Type(I(p)) = C(p)_{MS}]$

4.4. Regla de disparo de transiciones

Habiendo dado una definición formal de la CCPN, ahora describiremos su comportamiento. Uno de los elementos más importantes que rigen el comportamiento de la PN son los tokens. Como se había mencionado anteriormente, los tokens de CCPN no son tokens simples, sino que tienen la capacidad adicional de almacenar información de un cierto tipo de dato. Su definición es la siguiente:

Definición 4.3 *Un elemento token es un par* (p, c) *donde* $p \in P$ *y* $c \in C(p)$*. 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:*

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

El conjunto de todas las marcas es expresado por M*.*

Otro elemento fundamental que rige el comportamiento de la CCPN es la transición, a continuación describimos las condiciones bajo las cuales una transición se dispara y cómo se refleja en el cambio de estado de la CCPN.

Una transición $t \in T_{copy}$ se activa en una marca M si y sólo si:

$$\forall p \in {}^{\cdot}t : M(p) > 0.$$

Una transición $t \in T_{regla}$ se activa en una marca M si y sólo si se satisface lo siguiente:

$$\forall p \in {}^{\cdot}t : M(p) > 0 \text{ y } Type(Cond(t)) = verdadero$$

Si $t \in T$ está activada, necesitamos una función $C_{enabled}$ para establecer la activación, así como los tokens que la originaron.

Definición 4.4 *Cuando $t \in T$ está activada, la función de activación $C_{enabled}$ asocia cada par ordenado $(p,t) \in P \times T\}$ con el multi-conjunto de un tipo de dato $C(p)$. Es decir,*

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

Definición 4.5 *Cuando una transición t está activada en una marca M_1, puede cambiar de M_1 a otra marca M_2, respetando lo siguiente:*

(i) Si $t \in T_{copy}$, $\forall p_1, p_2 \in P$, $p_1 \in\ ^{\cdot}t$, $p_2 \in t^{\cdot}$:

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

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

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

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

Gráficamente, una CCPN para una sola regla es como se muestra en la figura 4.1 a), donde el lugar de entrada de la transición representa al evento de la regla ECA, la transición almacena la parte condicional y el lugar de salida de la transición representa la acción de la regla. La figura 4.1 b) muestra una CCPN de dos reglas cuyo evento activador es el mismo para ambas. La transición $t \in T_{copy}$ duplica el token del evento colocándolo en lugares de salida que a la vez son los lugares de entrada de transiciones que guardan la condición de las reglas. La figura 4.1 c) muestra una CCPN de dos reglas, donde la acción ejecutada por una de ellas es el evento activador de la otra.

4.5. Algoritmo de conversión de reglas ECA a CCPN

Para llevar a cabo la conversión de una base de reglas ECA a una CCPN se desarrolló un algoritmo, el cual se auxilia de una estructura denominada "regla", formada por tres campos:

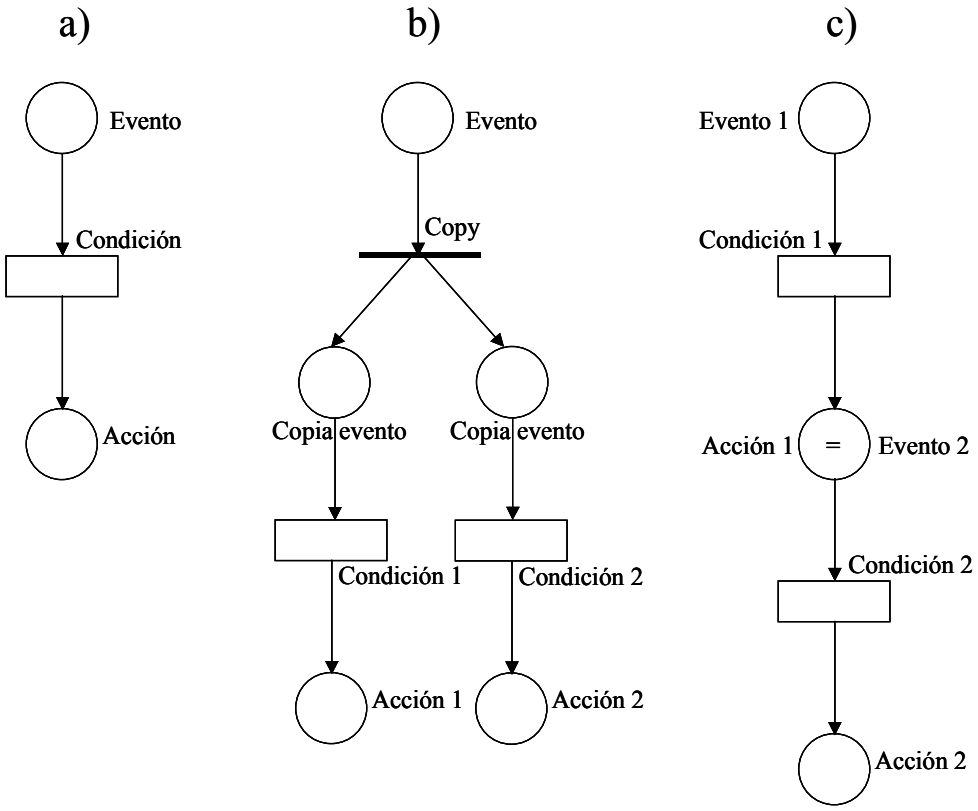

Figura 4.1: Modalidades que pueden presentarse en la creación de una CCPN.

evento, condición y acción. La entrada de este algoritmo es la base de reglas ECA, almacenada en un arreglo de estructuras tipo "regla":

1.- Leer el arreglo de reglas.
    ArregloReglas[1...N], donde N es el número de reglas.
2.- Repetir hasta el paso 45, I = 1...N
3.- Asignar los elementos de la regla I a las variables EVENTO, CONDICION Y
    ACCION, para su conversión en CCPN.
    EVENTO ← ArregloReglas[I].evento
    CONDICION ← ArregloReglas[I].condición
    ACCION ← ArregloReglas[I].acción
4.- Si EVENTO no existe en la CCPN.
     5.- Crear un lugar $p_1 \in P$ para representar a EVENTO.
    6.- Crear una transición $t_r \in T_{regla}$ para representar a CONDICION.
    7.- Conectar un arco dirigido de p_1 a t_r.
    8.- Si ACCION no existe en la CCPN.
        9.- Crear un lugar $p_2 \in P$ para representar a ACCION.
        10.- Conectar un arco dirigido de t_r a p_2.
    11.- En caso contrario (ACCION existe en la CCPN).
        12.- Conectar un arco dirigido de t_r al lugar que representa la
            ACCION existente.
13.- En caso contrario (EVENTO existe en la CCPN).
    14.- Si ya existe una transición $t \in T$ con ${}^{\cdot}t =$ {EVENTO},
        significa que no es un nodo terminal.
        15.- Si ya existe una transición $t_c \in T_{copy}$ con ${}^{\cdot}t =$ {EVENTO}.
            16.- Agregar un lugar EVENTO_COPY
            17.- Conectar un arco dirigido de t a EVENTO_COPY.
            18.- Crear una transición $t_r \in T_{regla}$ para almacenar
                a CONDICION.
            19.- Conectar un arco dirigido de EVENTO_COPY a t_r.
            20.- Si ACCION no existe.

21.- Crear un lugar para representar a ACCION.

22.- Conectar un arco dirigido de t_r a ACCION.

23.- En caso contrario (ACCION ya existe).

24.- Conectar un arco dirigido de t_r a ACCION existente.

25.- En caso contrario (No existe una transición $t_c \in T_{copy}$).

26.- Agregar una transición $t_c \in T_{copy}$.

27.- Conectar un arco dirigido de EVENTO a t_c y eliminamos el arco dirigido anterior de EVENTO.

28.- Agregar dos lugares EVENTO_COPY.

29.- Conectar arcos dirigidos de t_c a cada EVENTO_COPY.

30.- Conectar un arco dirigido de un EVENTO_COPY a la transición que conectaba a EVENTO.

31.- Crear una transición $t_r \in T_{regla}$ para almacenar a condición.

32.- Conectar un arco dirigido del EVENTO_COPY restante a t_r.

33.- Si ACCION no existe.

34.- Crear un lugar para representar a ACCION.

35.- Conectar un arco dirigido de t_r a ACCION.

36.- En caso contrario (ACCION ya existe).

37.- Conectar un arco dirigido de t_r a ACCION existente.

38.- En caso contrario (si es un nodo terminal).

39.- Crear una transición $t_r \in T_{regla}$ para representar a CONDICION.

40.- Conectar un arco dirigido de EVENTO existente (nodo terminal) a t_{r}.

41.- Si ACCION no existe en la CCPN.

42.- Crear un lugar $p \in P$ para representar a ACCION.

43.- Conectar un arco dirigido de t_r a p.

```
        44.- En caso contrario (ACCION existe en la CCPN).
        45.- Conectar un arco dirigido de t_r al lugar que representa
             la ACCION existente.
46.- Fin del algoritmo.
```

Este algoritmo toma como base la primera regla, a partir de la cual se le van agregando las reglas subsecuentes de acuerdo a la relación que tengan con sus antecesoras. Si son reglas independientes, si tienen un mismo evento activador o si la acción de una es a la vez el evento activador de otra, entonces se conectan de acuerdo a estas relaciones. Finalmente, la CCPN resultante representa a la base de reglas ECA, mostrando la correspondencia existente entre ellas.

4.6. Modelación y simulación de reglas ECA con CCPN

Para ilustrar el uso del algoritmo de la sección anterior, se aplica a los siguientes ejemplos:

Ejemplo 1

Para empezar, el algoritmo se aplica a una regla sencilla. Supóngase que se tiene una BD con una tabla denominada CUENTA, la cual tiene los campos numero, cliente, saldo e interés. Una regla activa aplicada a esta BD es por ejemplo: Cada vez que se agregue una cuenta nueva a la BD CUENTA, si el interés se encuentra entre 1 % y 2 %, entonces el interés se redondea a 2 %. La definición de ésta regla en la sintaxis propuesta es:

```
ON insert_CUENTA
IF interés > 1.0 & interés < 2.0
THEN update CUENTA set interés = 2.0 where interés > 1.0 and interés < 2.0;
```

El arreglo de reglas, variable de entrada al algoritmo, solamente tendrá un elemento, la regla anterior. La asignación de las variables EVENTO, CONDICION y ACCION son:

```
EVENTO ← ''INSERT_CUENTA''
CONDICION ← ''interés > 1.0 & interés < 2.0''
ACCION ← ''update_CUENTA_interés''
```

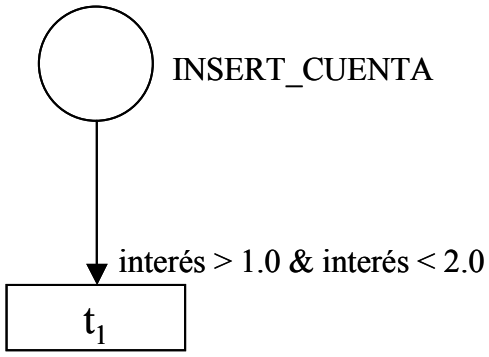

Figura 4.2: Creación del evento (lugar de entrada) y de la condición (transición).

Como el paso 4 se cumple (EVENTO no existe en la CCPN), se ejecuta el paso 5 (crear un lugar para representarlo), 6 (crear una transición t_1 para representar a CONDICION) y 7 (conectar un arco dirigido del EVENTO a CONDICION). Figura 4.2.

Como el paso 8 del algoritmo se cumple (ACCION no existe en CCPN), se ejecutan pasos 9 y 10 (crear un lugar para representar a ACCION y conectar un arco dirigido de la transición al lugar de la ACCION). Figura 4.3.

La figura 4.3 muestra la CCPN generada a partir de la base de reglas de entrada al algoritmo.

Ejemplo 2

A la base de reglas del ejemplo anterior se le agrega la regla: Cada vez que se agregue una cuenta nueva a la BD CUENTA, si el saldo de la cuenta es menor que $0.00, entonces el saldo se ajusta a $0.00. Esta nueva regla se representa, de acuerdo a la sintaxis propuesta, de la siguiente manera:

```
ON insert_CUENTA
IF saldo < 0.00
THEN update CUENTA set saldo=0.00 where saldo < 0.00;
```

Ahora el arreglo de reglas contiene dos elementos. En la primera iteración del algoritmo se genera una CCPN como la figura 4.3. Continuando con la aplicación del algoritmo, en la segunda iteración las variables EVENTO, CONDICION y ACCION almacenan los datos:

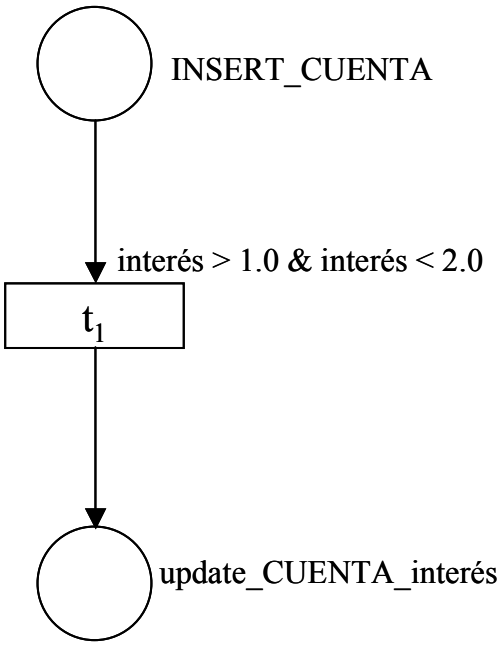

Figura 4.3: CCPN obtenida de la regla ECA correspondiente.

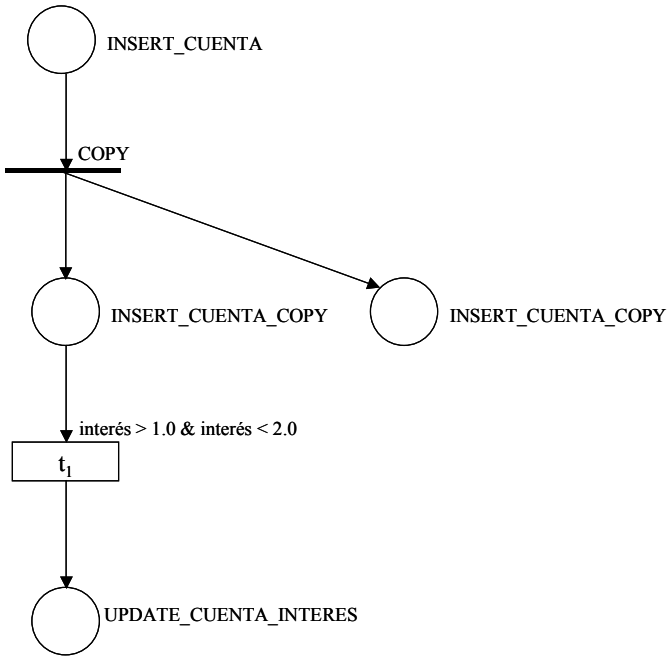

Figura 4.4: Incorporación de la trancición $t_c \in T_{copy}$ para hacer una réplica del token del lugar de entrada.

```
EVENTO ← ''insert_CUENTA''
CONDICION ← ''saldo < 0.00''
ACCION ← ''update_CUENTA_saldo''
```

Como el paso 4 no se cumple (no existe EVENTO), se ejecuta el paso 13. El paso 14 tampoco cumple (ya existe una transición COPY), por lo tanto, ejecutamos los pasos 24 al 31, con los cuales agregamos una transición COPY para duplicar el evento activador de las dos reglas y generamos una transición t_2 para representar la CONDICION de la segunda. Figura 4.4.

La condición del paso 33 no se cumple (no existe ACCION), entonces realizamos los pasos 36 y 37 para conectar la transición t_2 de la regla a la ACCION existente. Figura 4.5.

La figura 4.5 muestra la CCPN generada a partir de la base de reglas de entrada al algoritmo, compuesta por dos reglas.

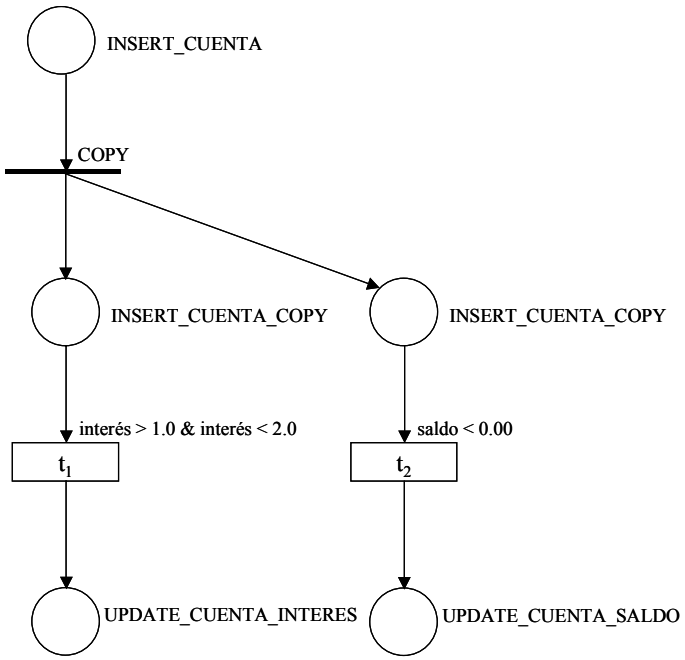

Figura 4.5: CCPN obtenida a partir de la definición de dos reglas con el mismo evento activador.

Ejemplo 3

Considerando ahora la tabla SALDOBAJO, con los campos número, fecha_inicio y fecha_término. Esta tabla almacena aquellas cuentas que tengas como saldo una cantidad menor de $500.00. A la base de reglas del ejemplo anterior se le agrega la regla: Cuando se actualiza el campo "saldo" de la tabla CUENTA, si el saldo es menor que $500.00 y no se ha almacenado como una cuenta con un saldo bajo, entonces se agrega a la tabla SALDOBAJO el número de la cuenta, la fecha en que el saldo se redujo y en el campo `fecha_término` se guarda un valor nulo.

```
ON update_CUENTA_saldo
IF saldo < 500.00 & not exists (select * from SALDOBAJO where
    num = new.num and end is null)
THEN insert into SALDOBAJO values (new.num, today(), null);
```

Ahora el arreglo de reglas contiene tres elementos. De acuerdo con la secuencia del ejemplo anterior, el algoritmo genera una CCPN como la figura 4.5. Continuando con la aplicación del algoritmo, en la tercera iteración las variables EVENTO, CONDICION y ACCION almacenan los datos:

```
EVENTO ← ''update_CUENTA_saldo''
CONDICION ← ''saldo < 500.00 & not exists (select * from SALDOBAJO
    where num = new.num and end is null)''
ACCION ← ''insert_CUENTA''
```

Como el paso 4 no se cumple, se verifica el paso 13, el cual sí cumple con la condición (EVENTO existe en la CCPN). Pero la condición del paso 14 no se cumple, ya que se trata de un nodo terminal, entonces se aplica el paso 38, 39 y 40 para crear la transición t3 de la CONDICION y conectar al EVENTO con ésta transición. Figura 4.6.

Se verifica el paso 41. Como la ACCION no existe, entonces se ejecutan los pasos 42 y 43, para crear el lugar correspondiente a ACCION y conectar a la CONDICION con la ACCION. Figura 4.7.

La CCPN de la figura 4.7 representa a la base de tres reglas en este ejemplo.

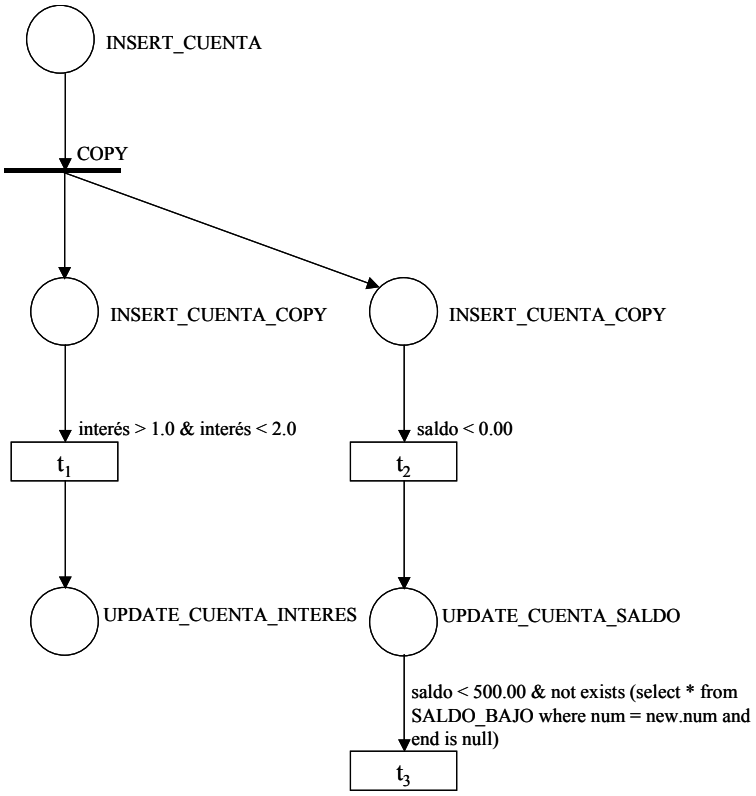

Figura 4.6: A la CCPNde la figura 4.5 se le agrega una regla más, donde el lugar de salida de t_2 es el evento activador de la nueva regla y t_3 es la parte condicional.

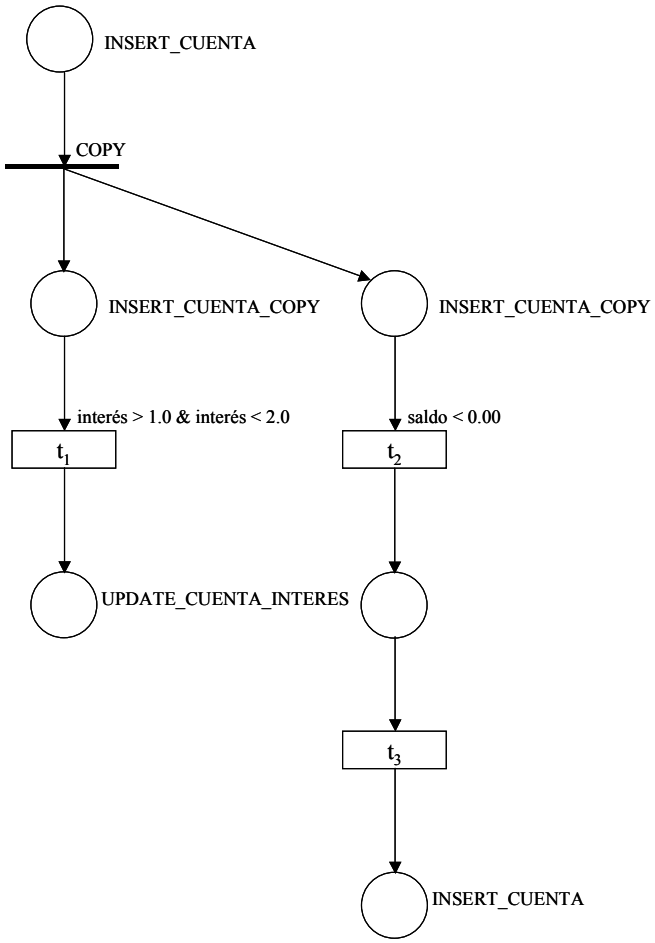

Figura 4.7: CCPN resultante donde se consideraron tres reglas ECA.

Ejemplo 4

Se presenta un último ejemplo, pero ahora es para mostrar la forma en que se realiza la ejecución del disparo de transiciones de la CCPN a partir de una marca inicial M_0. Se toma la CCPN generada en el ejemplo anterior para llevar a cabo la simulación. La iniciación de los lugares $p \in P$ es como sigue:

`I(INSERT_CUENTA) = 1‘(15462, ’Joselito Medina Marín’, -100.00, 0.50)`

`I(UPDATE_CUENTA_INTERES) = ∅`

`I(UPDATE_CUENTA_SALDO) = ∅`

`I(INSERT_CUENTA) = ∅`

Esta iniciación se muestra en la figura 4.8.

En esta marca inicial M_0, la única transición que está activada es $t_c \in T_{copy}$, al dispararla se elimina el token del lugar de entrada a t_c y se agrega un token idéntico en cada lugar de salida de t_c. Figura 4.9.

En la marca M_1 resultante, las transiciones t_1 y t_2 se activan. Pero, para que la transición se dispare, también se evalúa la condición. La condición de t_1 es "interés $>$ 1.0 & interés $<$ 2.0" y el token almacena los datos (15462, 'Joselito Medina Marín', -100.00, 0.50). La primera parte de la condición es falsa, ya que $0.5 > 1.0$, por lo tanto la transición t_1 no se dispara. En cambio, la transición t_2 tiene como condición "saldo $<$ 0.00" y la información del token es (15462, 'Joselito Medina Marín', -100.00, 0.50). La evaluación de la condición -100.00 $<$ 0.00 es verdadera, por lo tanto, la transición t_2 se dispara, generando el token ("saldo=0.00", "where saldo $<$ 0.00;") para colocarlo en su lugar de salida "UPDATE_CUENTA_SALDO". Figura 4.10.

El disparo de la transición t_2 da como resultado una marca M_2 (figura 4.10), en la cual la transición t_3 se activa. Para realizar el disparo de t_3 se verifica la condición "saldo $<$ 500.00 & not exists (select * from SALDOBAJO where num = new.num and end is null)" con la información del token "("saldo=0.00", "where saldo $<$ 0.00;")". La primera condición 0.00 $<$ 500.00 se cumple, la segunda también es verdadera, porque es una cuenta que se acaba de agregar, entonces no existe dentro de la tabla SALDOBAJO. Entonces la transición t_3 se dispara, suponiendo que la fecha actual es 26 de agosto de 2002, se genera el token "(15462, '26-08-2002', null)" para colocarlo en su lugar de salida "INSERT_SALDOBAJO". Figura 4.11.

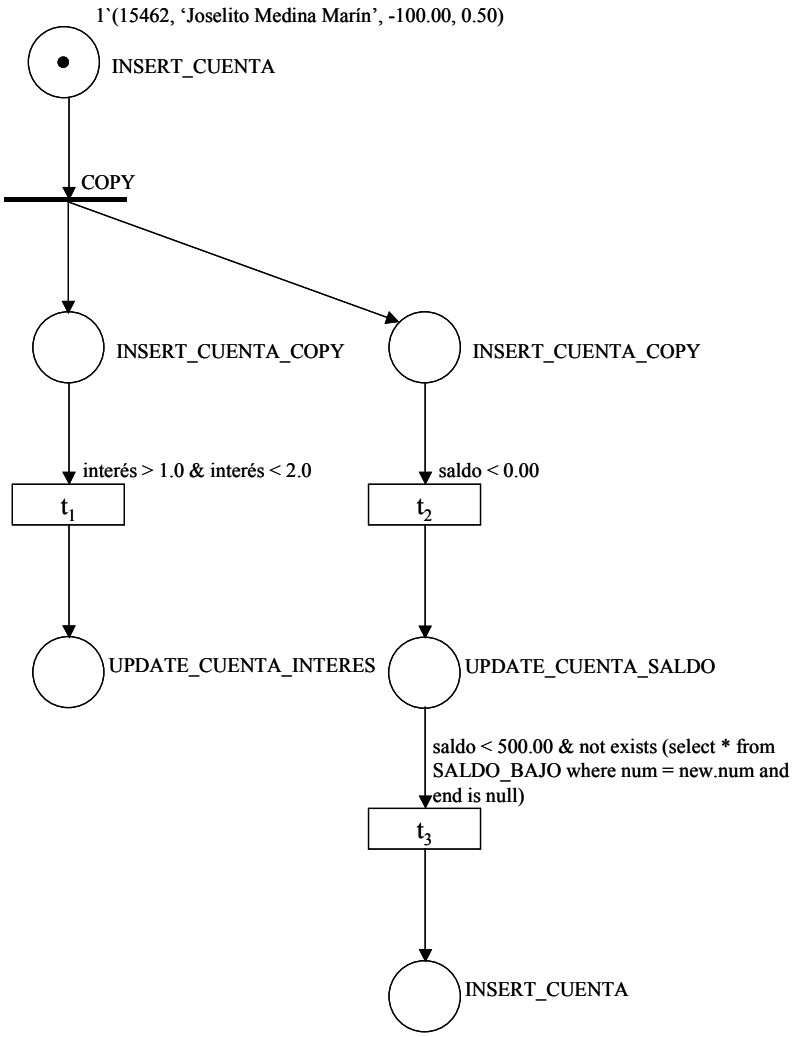

Figura 4.8: Marca inicial M_0 de la CCPN.

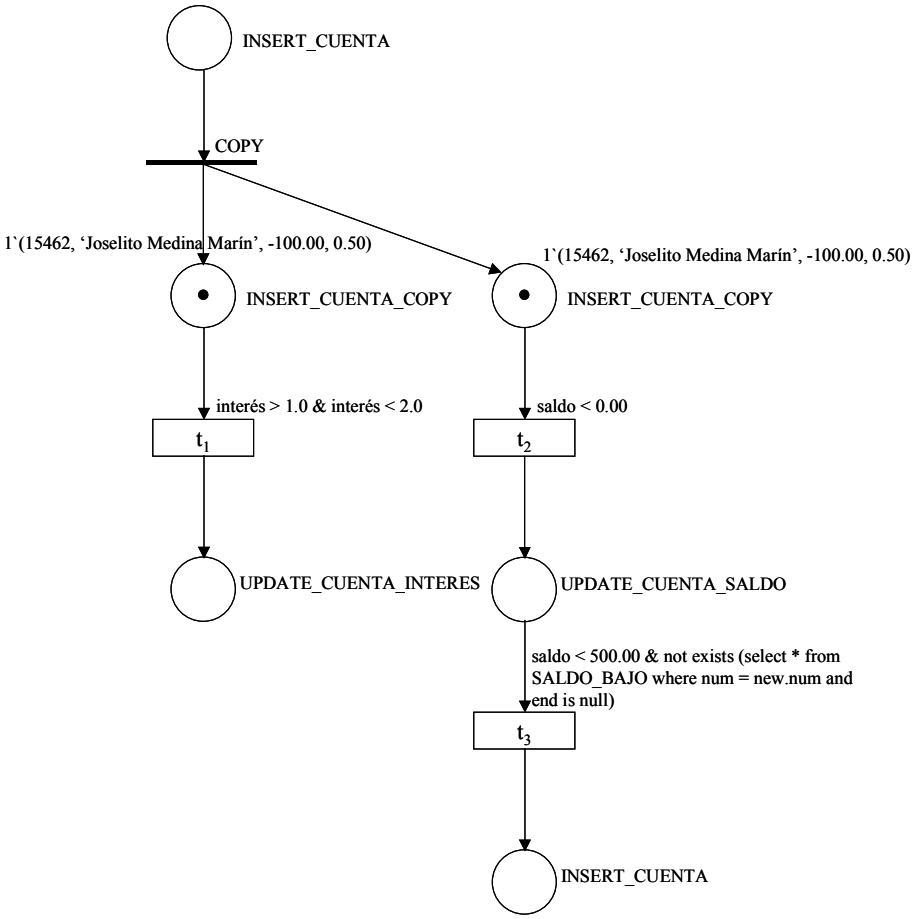

Figura 4.9: Marca M_1 obtenida después del disparo de la transición $t_c \in T_{copy}$.

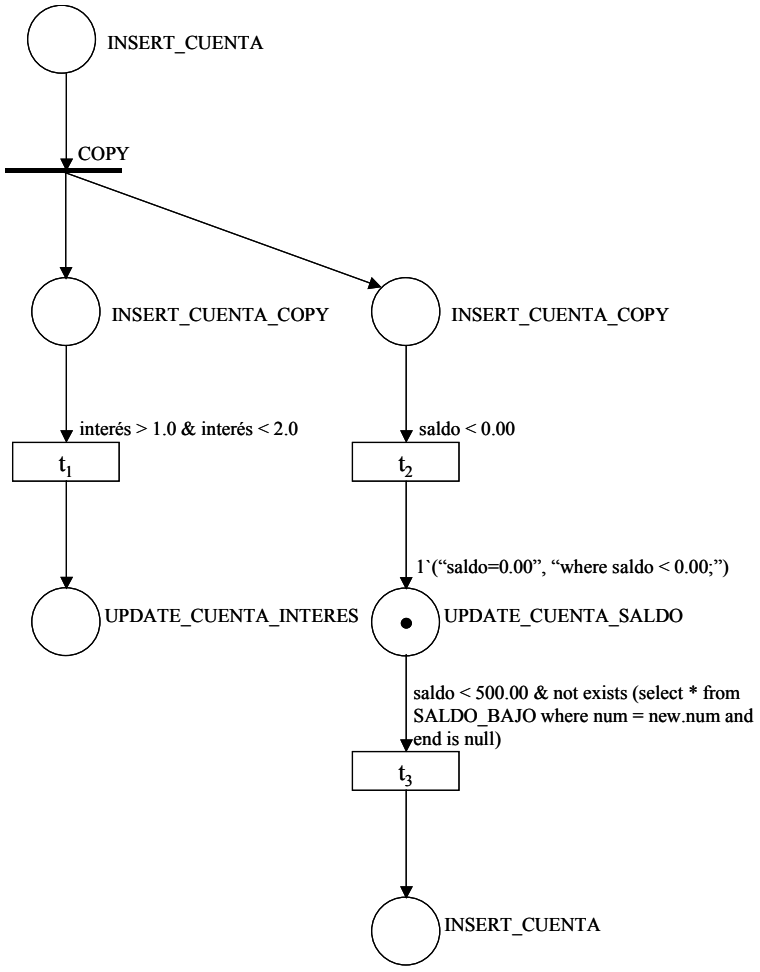

Figura 4.10: Marca M_2 obtenida a partir del disparo de la transición t_2.

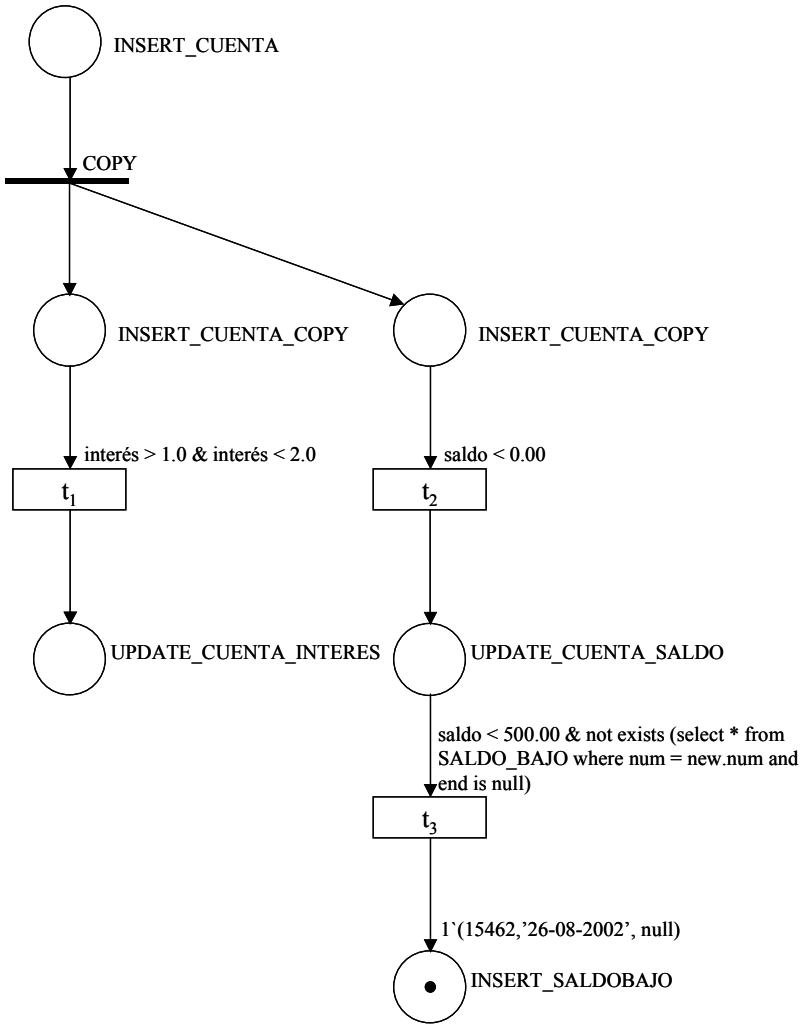

Figura 4.11: Marca final en la ejecución de la simulación.

4.7. Comentarios finales

La definición de reglas ECA, en los sistemas de BDA comerciales, se realiza con la sintaxis que ofrece cada sistema. Cuando se define un número pequeño de reglas se detecta fácilmente la presencia de inconsistencias, como no-terminación y confluencia. Sin embargo, cuando la base de reglas se compone por muchas reglas, es difícil detectar la relación existente entre ellas y en consecuencia, las inconsistencias que pudieran presentarse en la base de reglas.

Para detectar estos problemas en una base de reglas activas, se han desarrollado diferentes enfoques, [34] [35] [8] [36], para descubrirlos y resolverlos. Sin embargo, este tipo de análisis se realiza en un entorno totalmente independiente a la definición de las reglas. Esto conlleva a tener por un lado la definición de las reglas y por otro el análisis de los problemas de inconsistencia.

En esta investigación se propone un enfoque donde se aplican las características de modelación y simulación que ofrecen las PN. La CCPN propuesta es una extensión de PN que modela una base de reglas ECA y a la vez, utilizándola como una herramienta gráfica, muestra la relación que existe entre ellas. De esta manera, la CCPN permite de manera visual detectar la presencia de ciclos en el disparo de reglas ó la convergencia de dos reglas en la misma acción. Además, con la CCPN puede simularse el comportamiento de la base de reglas antes de declararla en una BD.

Proponemos también un algoritmo para generar una CCPN a partir de una base de reglas ECA, el cual comienza con la generación de la forma más sencilla de una CCPN de la primera regla, agregándosele las reglas subsecuentes de acuerdo a su relación con las reglas ya definidas. Finalmente, se presentan ejemplos de reglas ECA para mostrar la aplicación del algoritmo.

Capítulo 5

Plataforma de desarrollo: ECAPNSim

Se han implementado una gran variedad de editores de PN, los cuales pueden consultarse en [14]. Estos editores tienen la capacidad de modelar PN coloreadas, con tiempo, estocásticas, orientadas a objetos, entre otras. Algunos de los sistemas de [14] ofrecen un editor gráfico, animación del movimiento de los tokens cuando se disparan las transiciones, control de la velocidad de la simulación, análisis de la estructura de la PN, análisis de los espacios de estados que pueden alcanzarse desde un estado inicial y métodos de análisis para las PN. Sin embargo, ninguno de ellos modela bases de reglas ECA. Por lo cual se desarrolló una interfaz gráfica para modelar y simular CCPN, a la que hemos denominado ECAPNSim (ECA rules & Petri Net Simulator). ECAPNSim simula y ejecuta bases de reglas ECA, utilizando a la CCPN como un modelo para representar bases de reglas ECA.

En la sección 5.1 se plantea la manera en que se desarrolla el ECAPNSim, describiendo el ambiente bajo el que se programó y se muestra la arquitectura en que está dividido. En la sección 5.2 se describe el diagrama de las clases que intervienen en el sistema, mostrando las clases que se forman por herencia y las que se forman por composición de otras clases. Además, en esta sección se mencionan las características de cada clase, así como los métodos que utilizan. En la sección 5.3 se explica la manera de utilizar el ECAPNSim y cómo genera una CCPN a partir de una base de reglas ECA almacenada en un archivo de texto. Se muestra también las dos formas de trabajo del ECAPNSim: simulación y real. Finalmente, en la sección 5.4 se hacen unos comentarios sobre el capítulo.

5.1. Planteamiento

El desarrollo de bases de reglas activas es una actividad que necesita realizarse con mucho cuidado. Actualmente existen muy pocos sistemas, como [10], que realizan la depuración y análisis de la base de reglas. La mayoría de los sistemas comerciales de BDA ([3], [5], [6], 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 pensar en otra alternativa de solución.

A diferencia de otros simuladores, ECAPNSim ofrece dos modalidades de uso. En la primera modalidad, el usuario ejecuta la simulación del comportamiento de la base de reglas. En la segunda modalidad, la base de reglas analizadas previamente, dará comportamiento activo a una BD pasiva, utilizando la misma CCPN con que se ejecutó la simulación.

5.1.1. 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. Es posible controlar la velocidad con que se ejecuta la simulación La interfaz gráfica ofrece barras de herramientas e íconos de edición, simulación y operaciones sobre archivos.

ECAPNSim cuenta con un procedimiento de generación automática de una estructura CCPN a

partir de la definición de una base de reglas activas. Este procedimiento es una implementación del algoritmo presentado en la sección 4.5. ECAPNSim toma la declaración de la base de reglas de un archivo de texto, de acuerdo a la sintaxis descrita en la sección 4.1. Las reglas leídas se convierten en objetos y se almacenan en un arreglo, el cual es el dato de entrada al procedimiento de conversión.

El desarrollo de ECAPNSim se hizo bajo la plataforma MAC OS X Server, dado que la BD pasiva (Postgres) se encuentra instalada en este laboratorio.

5.1.2. Arquitectura de ECAPNSim

En 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 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 SQL (como se explicó en la sección 2 del capítulo 4) y les agrega la información contenida en el mensaje. La asignación de tokens se refiere a la colocación de tokens en los lugares de la CCPN correspondiente. La CCPN siempre se encuentra en espera de tokens, cuando recibe alguno verifica si la transición 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 (condición evaluada a verdadero) se analizan para preparar la instrucción SQL que se ejecutará en la BD. La descripción anterior se muestra en la figura 5.1.

5.2. 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 exportar reglas

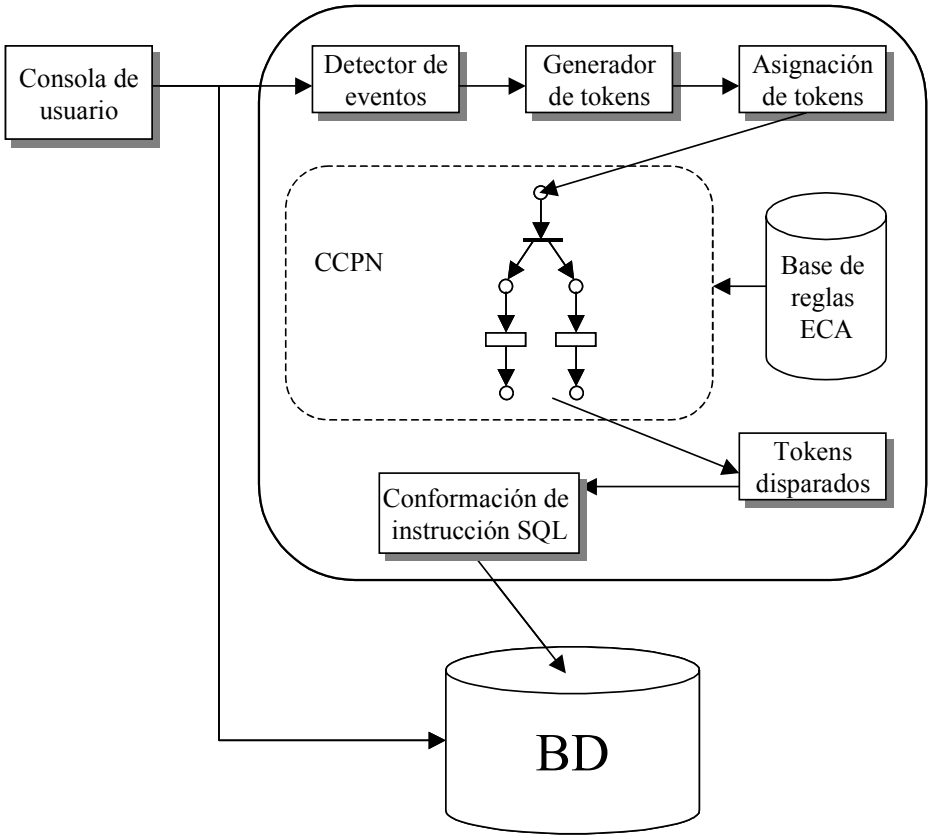

Figura 5.1: Arquitectura del ECAPNSim.

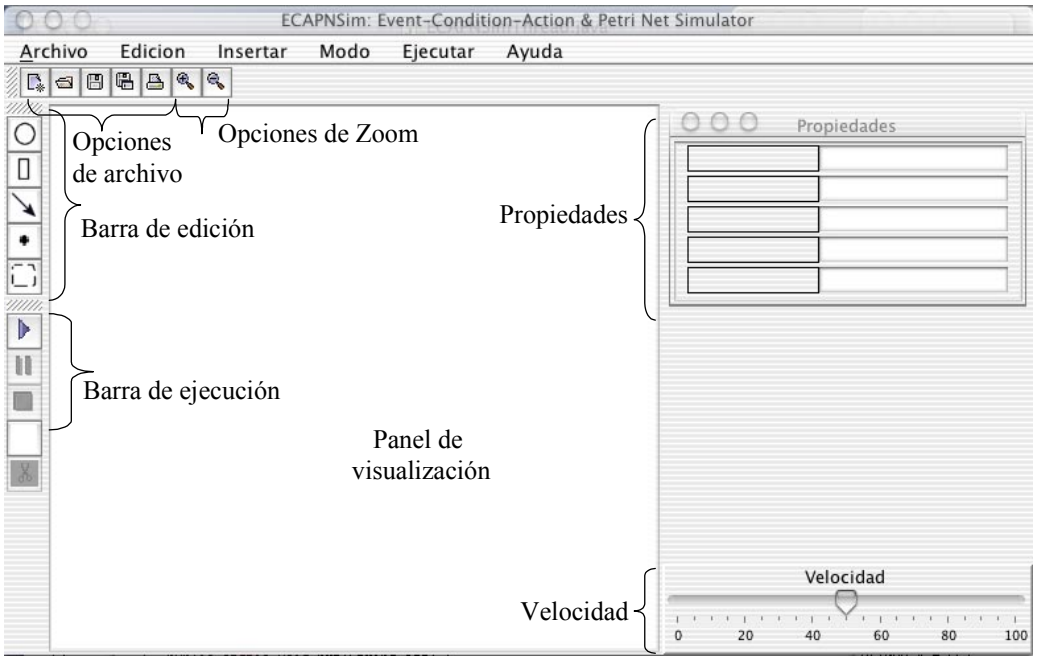

Figura 5.2: Pantalla de visualización de ECAPNSim.

ECA desde archivo y generar la CCPN correspondiente. Las características de la interfaz gráfica se muestran en la figura 5.2.

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

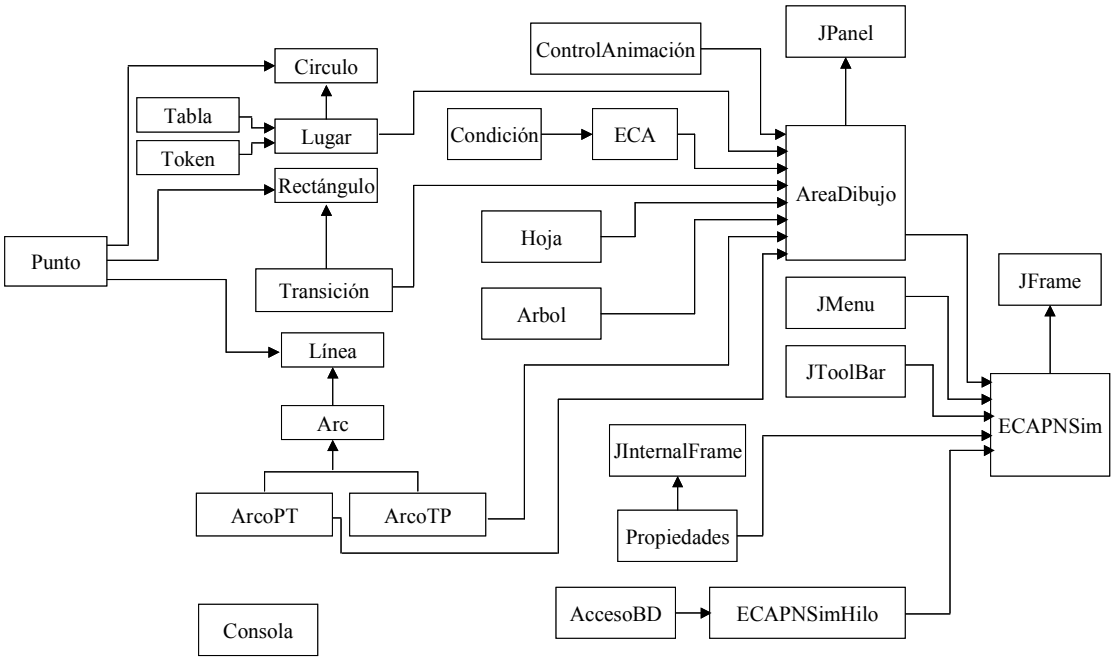

Figura 5.3: Diagrama de clases del ECAPNSim.

correspondientes a la acción de las reglas disparadas.

5.2.1. Diagrama de clases

En POO se utilizan los conceptos de herencia y composición [37]. 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 5.3.

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,* sobreescribiendo los métodos donde es necesario diferenciar entre un arco de lugar de entrada y uno de lugar de salida.

5.2.2. La clase *ECAPNSim*

Esta es la clase principal del sistema, la cual se llama igual que el sistema propio: ECAPNSim. Hereda los atributos y métodos de la clase "JFrame" de Java. Proporciona la interfaz gráfica donde el usuario interactúa con el sistema para el diseño de la base de reglas activas. En esta clase se hace la definición del área de dibujo. Las barras de herramientas para edición, manejo de archivos y ejecución de la simulación se definen en esta clase, así como el frame para la visualización de las propiedades de los elementos de la CCPN y el "slider" para controlar la velocidad de la simulación. Los métodos implementados en esta clase son los siguientes:

public String getCurrentPath(): Con éste método obtenemos el directorio actual, el cual es útil para mostrar el contenido del directorio cuando se desea abrir o guardar la CCPN generada.

public void abrirECA(): Este método presenta al usuario una ventana para seleccionar el archivo que contiene la definición de las reglas ECA. Utiliza un filtro para mostrar solamente a este tipo de archivos.

public void abrirEPS(): Este método permite al usuario seleccionar el archivo que haya almacenado una CCPN. Utiliza un filtro para mostrar solamente los archivos de CCPN generadas.

public void guardarEPS(): Muestra la ruta actual y permite especificar la ruta donde se guardará el archivo de la CCPN.

public int preguntar(): Se utiliza para lanzar una caja de diálogo y preguntar si el usuario desea guardar los cambios realizados a la CCPN actual.

5.2.3. La clase *AreaDibujo*

Esta clase es el motor del ECAPNSim. Los métodos que realizan la simulación del comportamiento de la base de reglas activa, así como los vectores que almacenan a los elementos de la CCPN se definen en esta clase. Extiende de la clase de Java "JPanel", para definir el área donde será dibujada la CCPN. Implementa la detección de eventos de mouse sobre el panel, para poder llevar a cabo operaciones de edición sobre la CCPN. El algoritmo para convertir una base de reglas en una estructura CCPN se implementa en esta clase.

Se definen Timers para controlar el desplazamiento de los tokens por la red y el valor del porcentaje que se asigna para el zoom. Los métodos implementados son los siguientes:

public void mousePressed(): Detecta cuando el botón derecho del "mouse" se presiona dentro del área de dibujo. Dependiendo el elemento que haya sido escogido previamente, se inserta o se selecciona un elemento en la posición (x,y) del área de dibujo donde se detectó el evento del "mouse". Si un elemento se selecciona con el "mouse", automáticamente se despliegan sus propiedades en la ventana respectiva.

public void mouseClicked(): Detecta cuando se ha dado doble click al botón derecho del "mouse". Cuando éste evento ocurre sobre un lugar, se despliega la información que contienen los tokens que se encuentren en ese lugar.

public void mouseDragged(): Se utiliza cuando un elemento de la CCPN es arrastrado hacia otra posición del área de dibujo.

public void paintComponent(): Método heredado de la clase "JPanel", el cual se encarga de dibujar los elementos de la CCPN sobre el área de dibujo. Cuando se ejecuta la simulación, éste método se invoca varias veces para dar la apariencia de movimiento en la animación del desplazamiento de los tokens.

public void abrir(): Utilizado para abrir un archivo que se haya almacenado previamente una CCPN.

public void guardar(): Utilizado para almacenar una CCPN generada.

public int importar(): Implementa el algoritmo de conversión de la base de reglas ECA a una CCPN. Toma como parámetro de entrada el archivo de texto donde se define la base de reglas. Un proceso interesante en éste método es la distribución que proporciona a la CCPN en el área de dibujo. Cuando se analiza el arreglo de reglas, el algoritmo de conversión solamente

conecta los lugares que estén relacionados. La asignación de coordenadas (x,y) se hace al terminar de conectar los lugares y transiciones, debido a que un mismo evento activador puede aparecer en las primeras y hasta en las últimas reglas. Si se asignan las coordenadas a los elementos cuando se analiza cada regla ECA, es posible que el evento de la última regla ya se haya definido, haciendo necesaria la reestructuración de la CCPN. Para evitar esto, primero se genera la CCPN sin asignarle coordenadas a los elementos, al finalizar el proceso de conversión ya se tiene conocimiento del total de elementos de la CCPN y por lo tanto, se puede realizar una distribución uniforme de la CCPN sobre el área de dibujo.

public void creaTabla(): Se encarga de la generación de objetos tipo "Tabla", donde se define la estructura de las tablas que conforman la BD con la cual se está trabajando.

public static void addTabla(): Agregar una tabla a cada lugar de la CCPN.

public boolean isHead(): Determina si un lugar dado es solamente un evento de entrada de una regla ECA y no la acción.

public void agregaHoja(): Método utilizado en la distribución de la CCPN sobre el área de dibujo. Se invoca cada vez que se agrega un nuevo elemento a la CCPN.

public void suprime(): Método invocado cuando se elimina un elemento de la CCPN.

public void play(): Inicia la simulación de la CCPN.

public void stopE(): Finaliza la simulación de la CCPN.

public void activarTrans(): Activa las transiciones que cumplen con la regla de disparo de una transición. Este método se invoca durante la simulación de la CCPN.

public void desactivarTrans(): Desactiva las transiciones que fueron disparadas. Este método se invoca durante la simulación de la CCPN.

public boolean anyTransActive(): Verifica si alguna de las transiciones está activada.

public void actualizaTokens(): Realiza las operaciones de eliminar tokens del lugar de entrada y agregar tokens al lugar de salida cuando se dispara la transición.

public void limpiar(): Elimina todos los elementos de CCPN que se encuentren en el área de dibujo.

public void modZoom(): Método para acercar ó alejar en el área de dibujo la imagen de la CCPN.

void scroll(): Método para agregar los elevadores de desplazamiento cuando se generan CCPN

que no se visualizan completamente en el área de dibujo.

public double getPorc(): Devuelve el porcentaje con el que se está visualizando la CCPN.

public void pres(): Identificar el tipo elemento que se seleccionó con el "mouse".

public void setDelay(): Asignar el valor de la velocidad en la simulación.

public void modo(): Establece la modalidad de operación de ECAPNSim, ya sea en modo Simulación o modo Real.

5.2.4. La clase *Propiedades*

Se utiliza para mostrar las propiedades de los elementos de la CCPN. El método implementado en esta clase es:

public void actualizaFrame(): Método se invoca cuando se selecciona un elemento de la CCPN. El método obtiene los datos del elemento y los presenta en la pantalla de propiedades.

5.2.5. La clase *ECAPNSimHilo*

Esta clase se encarga de definir el socket de comunicación y aceptar la conexión que solicita la consola de usuario. ECAPNSim funciona como servidor al aceptar múltiples hilos de conexión. Pueden ejecutarse más de una consola concurrentemente. El método de esta clase es:

public void run(): Abre los canales de comunicación entre la consola de usuario y el ECAPNSim. Es el detector de eventos, dado que recibe los mensajes que el usuario envía a la BD.

5.2.6. La clase *AccesoBD*

La comunicación entre ECAPNSim y la BD se establece mediante esta clase. A través de ella se envían las instrucciones SQL a la BD. Los métodos que se implementan en esta clase son:

public void inicializar(): Con este método se inicia la comunicación entre el ECAPNSim y la BD.

public void close(): Método que se invoca cuando se da por terminada la conexión entre el ECAPNSim y la BD.

public ResultSet executeQuery(): Este método realiza una consulta dentro de la BD. Se invoca cuando se utiliza el comando "select".

public int executeUpdate(): Con éste método se realizan modificaciones al estado que guarda la BD. Se invoca cuando se utilizan los comandos "insert", "delete" ó "update".

5.2.7. La clase *ControlAnimación*

Se utiliza para controlar la animación del desplazamiento de los tokens. La posición del punto que se va a graficar y el factor de desplazamiento del punto se definen en esta clase. Los métodos que se implementan en esta clase son:

public Punto getPto(): Obtiene la posición (x,y) del punto a graficar.

public void setPto(): Asigna la posición (x,y) del punto a graficar.

public double getAvX(): Obtiene el factor de avance del token sobre el eje X.

public double getAvY(): Obtiene el factor de avance del token sobre el eje Y.

public void setAvX(): Asigna el factor de avance del token sobre el eje X.

public void setAvY(): Asigna el factor de avance del token sobre el eje Y.

public void incrementa(): Incrementa los valores de X e Y de acuerdo al factor de avance.

5.2.8. La clase *ECA*

Esta clase genera objetos para almacenar las características de las reglas ECA. Esta clase se utiliza posteriormente en la clase "AreaDibujo" para generar la estructura CCPN, de acuerdo al algoritmo definido de conversión de reglas ECA en CCPN. El método `nombreAccion()` también se utiliza por la clase ECAPNSimHilo, para identificar el lugar donde se asignará el token correspondiente a la instrucción SQL que se recibe por medio de la consola. La parte condicional de la regla se almacena como un vector, para poder soportar condiciones compuestas, con la limitación de que solamente se aceptan condiciones unidas por el condicional "Y". Los métodos declarados en esta clase son:

public void divideCondComp(): Divide y obtiene las diferentes condiciones de la parte condicional.

public void divideCondPrim(): Divide y obtiene los diferentes valores cada condición.

public static String nombreAccion(): Devuelve el nombre del lugar de acuerdo a la instrucción SQL que recibe como parámetro. Este método es útil para determinar en qué lugar se colocará un token generado a partir de la instrucción SQL.

public String getEvent(): Regresa el evento a detectar.

public String getENomTabla(): Regresa el nombre de la tabla donde ocurrió el evento.

public String getANomTabla(): Regresa el nombre de la tabla donde se ejecutara la acción.

public String getCondition(): Regresa la condición a evaluar.

public Vector getVectorCond(): Regresa el vector de las condiciones a evaluar.

public String getAction(): Regresa la acción a ejecutar.

public void setAction(): Asigna la acción a ejecutar.

public String getActionSql(): Regresa la instrucción SQL de la acción a ejecutar.

public void divideCondComp(): Divide y obtiene las diferentes condiciones de la parte condicional.

public void divideCondPrim(): Divide y obtiene los diferentes valores cada condición.

public static String nombreAccion(): Devuelve el nombre del lugar de acuerdo a la instrucción SQL que recibe como parámetro. Este método es útil para determinar en qué lugar se colocará un token, generado a partir de la instrucción SQL.

public String getEvent(): Regresa el evento a detectar.

public String getENomTabla(): Regresa el nombre de la tabla donde ocurrió el evento.

public String getANomTabla(): Regresa el nombre de la tabla donde se ejecutara la acción.

public String getCondition(): Regresa la condición a evaluar.

public Vector getVectorCond(): Regresa el vector de las condiciones a evaluar.

public String getAction(): Regresa la acción a ejecutar.

public void setAction(): Asigna la acción a ejecutar.

public String getActionSql(): Regresa la instrucción SQL de la acción a ejecutar.

5.2.9. La clase *Condición*

Clase que se utiliza para almacenar expresiones lógicas. Las expresiones lógicas se dividen la variable, el operador de comparación y el valor contra el que se compara la variable. Los métodos que se implementan en esta clase son:

public int convierte(): Convierte la cadena de tipo de dato a su respectivo valor numérico.

public String getVariable(): Regresa la variable de la condición.

public int getCondicion(): Regresa el operador con el que se evalúa la condición.

public String getValor(): Regresa el valor de la condición.

public boolean evalua(): Evalúa la condición, regresando verdadero en caso que se cumpla la condición y falso en caso contrario.

5.2.10. La clase *Hoja*

Esta clase se implementó para asignarle a cada elemento de la CCPN una posición en el árbol generado por la concatenación de varias reglas ECA. La concatenación de modelos CCPN para reglas ECA depende de las relaciones que existan entre las reglas, por lo que es difícil conocer desde un inicio la posición en el Panel de cada elemento de CCPN. Esta clase se auxilia de los siguientes métodos:

public int getIndicePoT(): Regresa el índice de la hoja.

public int getTipo(): Regresa el tipo de la hoja, ya que puede ser un lugar ó una transición. public int getNivel(): Regresa el nivel donde se encuentra cada hoja con respecto a las demás.

public int getNumArbol(): Regresa el número del árbol al que pertenece la hoja.

public void setIndicePoT(): Asigna el índice a la hoja.

public void setTipo(): Asigna el tipo de la hoja (lugar ó transición).

public void setNivel(): Asigna el nivel donde se encuentra la hoja.

public void setNumArbol(): Asigna el número del árbol al que pertenece la hoja.

5.2.11. La clase *Arbol*

Esta clase se auxilia de la clase *Hoja* para formar una estructura completa con todas aquellas reglas que tengan alguna relación. Conociendo el número total de elementos que tiene la red en profundidad y a lo ancho, podremos calcular la posición de cada uno de ellos en un área proporcional a la cantidad de elementos que tenga el árbol. Esta clase define los siguientes métodos:

public Punto getPtoI(): Regresa el punto inicial (xi,yi) del área donde se dibuja la CCPN.

public Punto getPtoF(): Regresa el punto final (xf,yf) del área donde se dibuja la CCPN.

public Vector getVector(): Regresa el vector de elementos que existen por cada nivel del árbol.

public void setPtoI(): Establece el punto inicial del área donde será dibujada la CCPN.

public void setPtoF(): Establece el punto final del área donde será dibujada la CCPN.

public void setVector(): Asigna el vector de elementos que existen por cada nivel del árbol.

public int maximo(): Regresa el valor máximo de elementos que hay por nivel en el árbol.

5.2.12. La clase *Punto*

Esta clase construye objetos donde se almacena una coordenada (x,y). Para manipular a estos objetos se definieron los siguientes métodos:

public int valX(): Devuelve el valor X del objeto.

public int valY(): Devuelve el valor Y del objeto.

public void nvoX(): Asigna un nuevo valor a X.

public void nvoY(): Asigna un nuevo valor a Y.

public boolean iguales(): Verifica si el punto enviado como parámetro, coincide con el objeto *Punto*.

public double distancia(): Calcula la distancia entre dos objetos de la clase *Punto*.

public String toString(): Devuelve el valor del punto como "String".

5.2.13. La clase *Círculo*

Clase que se utiliza para declarar objetos de tipo Círculo, almacenando el punto origen (x,y) y el valor del radio. Esta clase se auxilia de los siguientes métodos:

public Punto getOrigen(): Devuelve el punto en que se encuentra el origen (x,y).

public int getRadio(): Devuelve el valor del radio del Circulo.

public void mover(): Desplaza el circulo en el eje cartesiano hacia un nuevo origen (x,y).

public double circunferencia(): Calcula el valor de la circunferencia del circulo.

public double area(): Calcula el valor del área del circulo.

public boolean pertenece(): Verifica si un punto dado p se encuentra dentro del área del circulo.

public String toString(): Devuelve el valor del Circulo como "String".

5.2.14. La clase *Lugar*

Esta clase se implementó para definir objetos de tipo lugar. Se definen variables para almacenar el nombre, cantidad de tokens, a que tabla representa y el tipo de evento que representa (insert,

update, delete, ó select). Los métodos que utilizan estos objetos son:

public void asignaTipo(): Asigna al objeto el tipo de comando que está representando ("insert", "delete", "update").

public void actNombre(): Actualiza el nombre del lugar.

public String getNombre(): Obtiene el nombre del lugar.

public void incTokens(): Incrementa la cantidad de tokens colocados en el lugar.

public void restarTokens(): Decrementa la cantidad de tokens colocados en el lugar.

public int getTokens(): Devuelve el numero de tokens que posee el lugar.

public Token getTokens(): Regresa el contenido del token.

public Tabla getTabla(): Obtiene el nombre de la Tabla que corresponde al lugar.

public int getTipo(): Obtiene el tipo de lugar ("insert", "delete", "update").

5.2.15. La clase *Tabla*

La definición de las tablas que forman la BD es importante para llevar a cabo una simulación adecuada. Esta clase proporciona al usuario la capacidad para definirlas, considerando los tipos de datos que se utilizan en SQL. Esta clase implementa los siguientes métodos:

public String[] getACampos(): Devuelve el arreglo de campos que componen la estructura de la tabla.

public String getCampo(): Devuelve el nombre del campo en un determinado índice.

public String[] getATipos(): Devuelve el arreglo de los tipos de dato con los que pueden definirse los campos de la tabla.

public void setVal(): Asigna un tipo de dato a los campos de la tabla.

public int size(): Devuelve el total de campos de la tabla.

public void setName(): Actualiza el nombre de la tabla.

public String getName(): Devuelve el nombre de la tabla.

5.2.16. La clase *Token*

En esta clase se almacena la información concerniente al evento que fue detectado, para posteriormente colocarlo en el lugar correspondiente. La clase Token es una extensión de la clase de Java

"HashMap", para aprovechar las facilidades que ofrece esta clase de almacenar objetos con su respectiva llave. El método generaToken( ) definido en esta clase, es utilizado por la clase AreaDibujo, para generar el token correspondiente a la acción de una regla, cuando la condición se cumple. Otra clase que utiliza a éste método es ECAPNSimHilo, que genera un token con la instrucción SQL enviada a través de la consola hacia el ECAPNSim. A continuación se describen los métodos de esta clase:

public String getVal (): Devuelve el valor almacenado en un campo del token.

public void setVal(): Asigna un valor a un campo del token.

public static Token generaToken(): Genera un token de acuerdo a la instrucción SQL que se haya recibido como parámetro. Este método convierte en tokens las instrucciones SQL obtenidas por el detector de evento, para colocarlos dentro de la CCPN.

public int getTipoPlace(): Devuelve el tipo de lugar al que pertenece el token.

5.2.17. La clase *Rectángulo*

Clase que define objetos de tipo rectángulo. En esta clase se almacena una coordenada origen del rectángulo, así como el valor de la base y de la altura. Los métodos que se consideran en esta clase son:

public Punto getOrigen(): Devuelve el punto en que se encuentra el origen (x,y) del rectángulo.

public Punto getCentro(): Devuelve el punto (x,y) en que se encuentra el centro del rectangulo.

public int getH(): Devuelve el valor de la altura del rectángulo.

public int getW(): Devuelve el valor de la base del rectángulo.

public void setH(): Asigna un valor a la altura del rectángulo.

public void setW(): Asigna un valor a la base del rectángulo.

public boolean pertenece(): Verifica si un punto dado se encuentra dentro del área del rectángulo. Este método es útil para determinar si un elemento lugar fue seleccionado con el "mouse".

public void mover(): Mueve el rectángulo a una nueva coordenada dentro del área de dibujo.

5.2.18. La clase *Transición*

La clase `Transición` implementa la funcionalidad del elemento transición de PN. Además, para evaluar las reglas ECA, se anexó la parte condicional de la regla ECA. Cuenta con un contador de tokens, para aquellos tokens que no cumplen con las condiciones de la regla. El contador de tokens es mostrado en el área de dibujo junto con la transición. Una transición realiza diferentes actividades, por lo que es necesario declarar los siguientes métodos:

public void actNombre(): Actualiza el nombre de la transición.

public String getNombre(): Obtiene el nombre de la transición.

public void actEstado(): Actualiza el estado de activación de la transición.

public void activar(): Activa una transición.

public boolean esActiva(): Verifica si la transición esta activada.

public boolean dispara(): Verifica si la transición puede dispararse.

public String getCondition(): Devuelve la condición de la transición.

public void setCondition(): Actualiza la condición de la transición.

public String getActionSql(): Regresa la instrucción SQL de la acción a ejecutar.

public boolean evaluaToken(): Evalúa el token de acuerdo a la condición de la transición.

public Token getToken(): Devuelve el token a evaluar.

public int getBadTokens(): Devuelve el numero de tokens rechazados.

public void incBadTokens(): Incrementa el numero de tokens rechazados.

5.2.19. La clase *Línea*

En esta clase se almacena el punto inicial y el punto final de una línea. Los métodos que ofrece esta clase son:

public Punto getPt1(): Regresa el primer punto de la línea.

public Punto getPt2(): Regresa el segundo punto de la línea.

public void setPt1(): Asigna el primer punto de la línea.

public void setPt2(): Asigna el segundo punto de la línea.

public void mover(): Mueve un extremo de la línea hacia un nuevo punto. Este método se utiliza cuando se desplaza un elemento (lugar o transición) y ya tiene colocado un arco, ya que el arco se mueve solamente en un extremo.

public double distancia(): Calcula la distancia que existe entre los dos puntos que definen a la línea, en otras palabras, calcula la longitud de la línea.

public double pendiente(): Calcula la pendiente que tiene la línea en el eje cartesiano.

public double distPunto(): Calcula la distancia que existe entre la línea y un punto dado.

public boolean pertenece(): Verifica si un punto dado se encuentra sobre la línea.

public Punto puntoMedio(): Calcula el punto medio de la línea.

5.2.20. Las clases *Arco, ArcoPT y ArcoTP*

En la clase *Arco* se define la funcionalidad básica de un arco de PN. *ArcoPT* y *ArcoTP* son clases que heredan los atributos y métodos de la clase *Arco*, sobrescribiendo los métodos que definen el punto inicial y final del arco. Los métodos que utilizan estas clases son:

public void actNombre(): Actualiza el nombre del arco.

public String getNombre(): Obtiene el nombre del arco.

public void actTokens(): Actualiza el numero de tokens que representa el peso del arco.

public int getTokens(): Devuelve el peso del arco.

public void calcX(): Calcula los valores para el arreglo de puntos xCoord[] utilizado para dibujar una flecha en el punto destino.

public void calcY(): Calcula los valores para el arreglo de puntos yCoord[] utilizado para dibujar una flecha en el punto destino

public void valBeta(): Calcula el ángulo que se forma entre los dos puntos del arco.

public double getBeta(): Devuelve el ángulo que se forma entre los dos puntos del arco.

public int[] getPointsX(): Devuelve el vector de puntos en X para dibujar la flecha.

public int[] getPointsY(): Devuelve el vector de puntos en Y para dibujar la flecha.

public int getPlace(): Obtiene el índice del lugar que conecta.

public void setPlace(): Modifica el índice del lugar que conecta.

public int getTrans(): Obtiene el índice de la transición que conecta.

public void setTrans(): Modifica el índice de la transición que conecta.

public void calcLD(): Calcula los puntos de la línea que será dibujada.

public void setLD(): Asigna los valores de la línea que será dibujada.

public Linea getLD(): Obtiene los valores de la línea que será dibujada.

5.2.21. La clase *Consola*

La definición de la consola se realiza en esta clase. Establece comunicación con la clase ECAPNSim para enviarle las instrucciones SQL que serán procesadas. El método que utiliza para establecer la comunicación es:

public void contacto(): Establece comunicación con el ECAPNSim, para el envío de las instrucciones SQL.

5.3. Uso de ECAPNSim

ECAPNSim permite al usuario modelar bases de reglas activas, ejecutar la simulación de su comportamiento y conectarse con una BD pasiva para ejecutar el disparo de las reglas en ella. Antes de utilizar ECAPNSim se necesita la definición de la base de reglas ECA en un archivo de texto, de acuerdo a la sintaxis de la sección 1 del capítulo 4. Por convención y para reconocer a los archivos de reglas para el ECAPNSim, éstos se guardan con la extensión "eca". Además, se debe conocer la estructura de las tablas que conforman la BD a la cual se le pretende implementar dinamismo.

5.3.1. Creación de una CCPN

Considerando el ejemplo 3 de la sección 6 del capítulo 4, se capturan las siguientes reglas en un archivo de texto:

```
insert_CUENTA
IF interés > 1.0 & interés < 2.0
THEN update CUENTA set interés = 2.0 where interés > 1.0 and interés < 2.0;

ON insert_CUENTA
IF saldo < 0.00
THEN update CUENTA set saldo=0.00 where saldo < 0.00;
```

A éste archivo lo guardamos como ejemplo1.eca. La estructura de la tabla CUENTA es:

```
CUENTA(número, nombre, saldo, interés)
```

Teniendo definido el archivo de la base de reglas ECA y la estructura de la tabla CUENTA, iniciamos ECAPNSim con la ejecución de la siguiente línea en la consola:

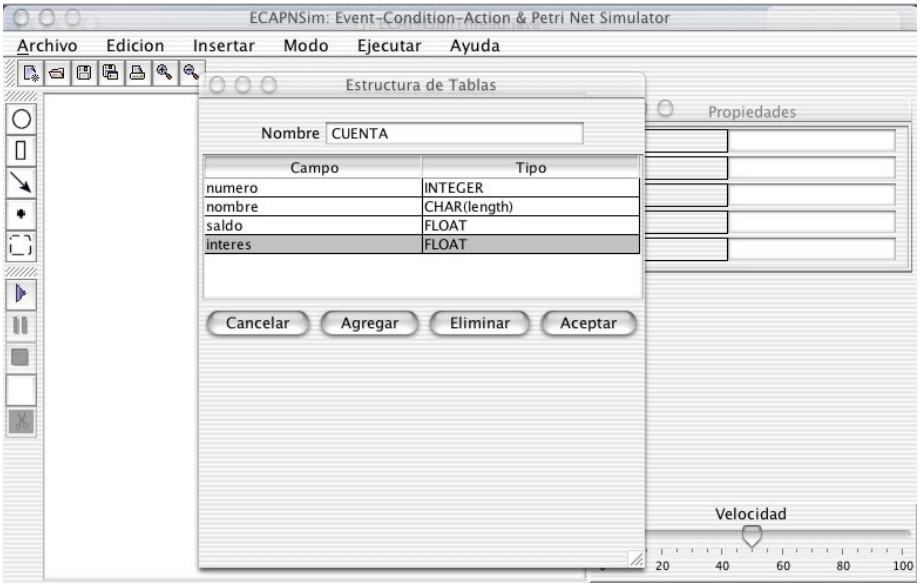

Figura 5.4: Declaración de la estructura de tablas.

```
$java ECAPNSim
```

Se obtiene la interfaz mostrada en la figura 5.2. Se procede a la definición de la estructura de la tabla CUENTA, seleccionamos el menú Archivo-Tabla y en el cuadro de diálogo que aparece se declara la estructura de CUENTA. Figura 5.4.

Ya definida la estructura de la tabla, se llama al procedimiento de conversión de la base de reglas ECA en una CCPN. Este procedimiento se localiza en el menú Archivo - Importar AR. Seleccionamos el archivo ejemplo2.eca, donde se declararon las reglas ECA. Figura 5.5.

El algoritmo de conversión genera una CCPN como la que se muestra en la figura 5.6. La CCPN obtenida puede almacenarse en un archivo, utilizando la extensión "pnj" para identificar los archivos generados con ECAPNSim.

Se presiona el botón PLAY, ubicado en la barra de ejecución, para esperar las instrucciones enviadas a través de la consola de usuario, la cual se invoca con el comando:

```
$java Consola
```

Se obtiene una interfaz como en la figura 5.7. Esta interfaz presenta un campo de edición de instrucciones SQL y un área donde se muestran los resultados obtenidos por la consulta a la BD.

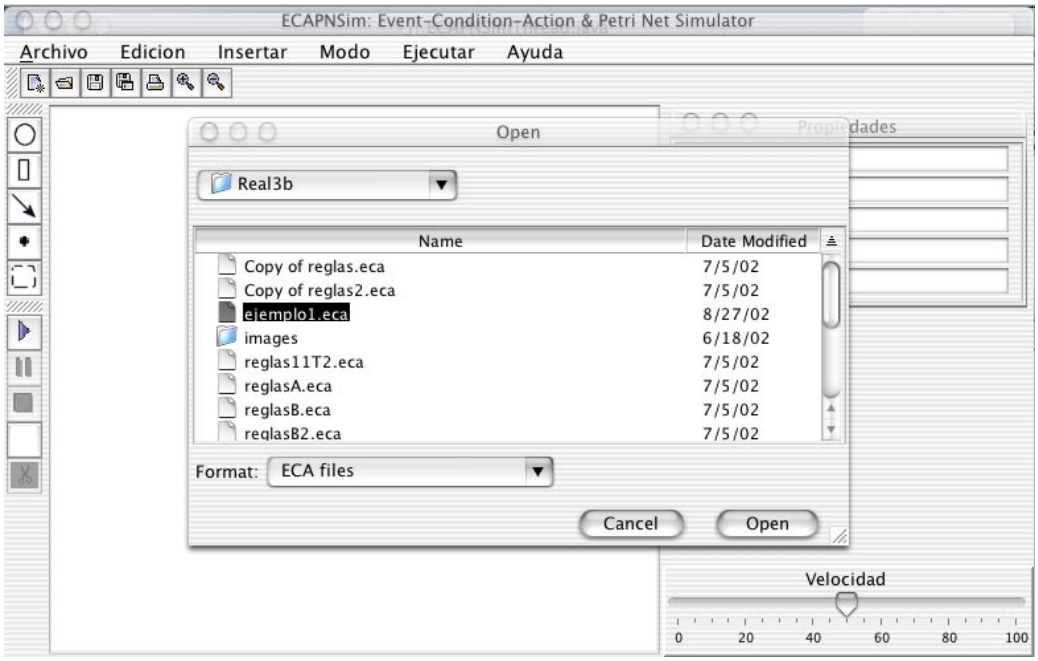

Figura 5.5: Archivo que contiene la definición de las reglas ECA.

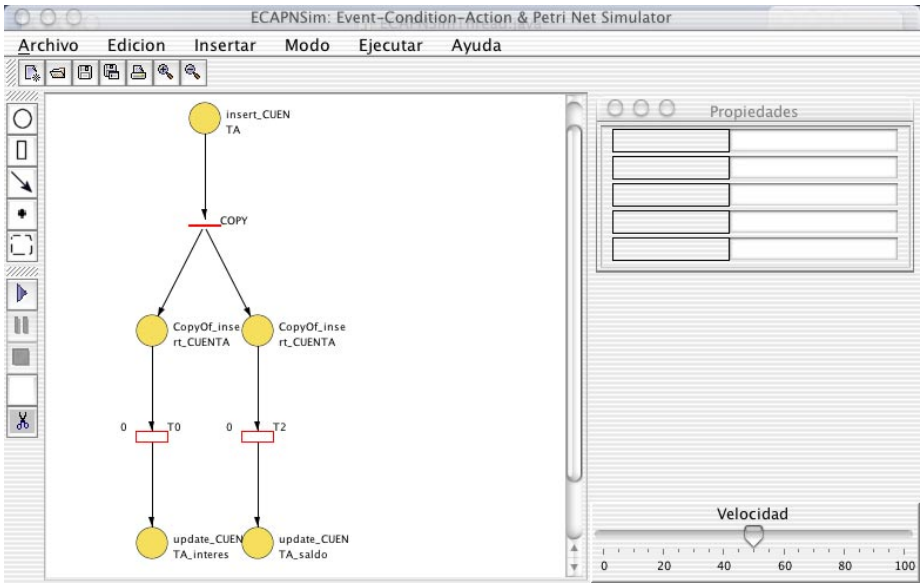

Figura 5.6: CCPN obtenida por el ECAPNSim a partir de las reglas de ejemplo1.eca.

En la consola de usuario se escriben las instrucciones SQL de igual manera como si se estuviese trabajando en una consola de la BD.

Para simular el comportamiento de la base de reglas ECA en la CCPN, se agrega el registro (125, 'Alan Medina', -100.00, 0.5) a la tabla CUENTA, escribiendo en la consola de usuario la instrucción correspondiente. Figura 5.8.

En la CCPN de ECAPNSim se agrega el token correspondiente a la instrucción enviada por la consola de usuario. Figura 5.9.

Como el mismo evento activa a dos reglas diferentes, entonces el token se duplica para evaluar ambas reglas. Figura 5.10.

Cada token se utiliza en la evaluación de la condición de la regla, almacenada en cada transición. Figura 5.11.

Al evaluar la condición de las transiciones, solamente una de ellas cumple y se dispara la transición, generando el token correspondiente a la acción de la regla. Figura 5.12.

La marca final de la CCPN se muestra en la figura 5.13.

De esta manera se simula la base de reglas ECA representada como una CCPN. Si el usuario

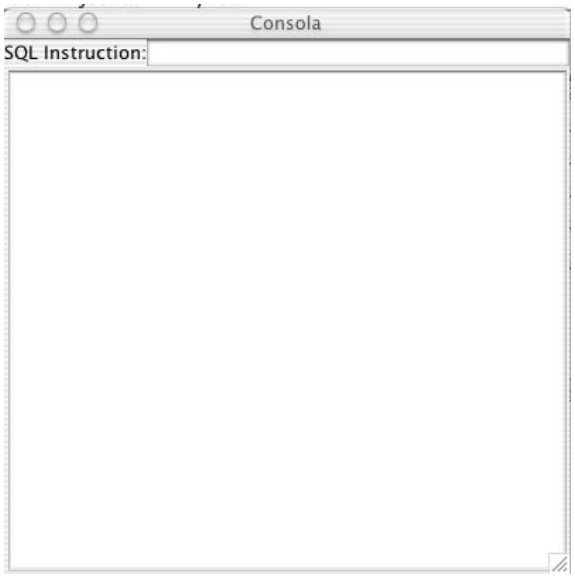

Figura 5.7: Consola de usuario.

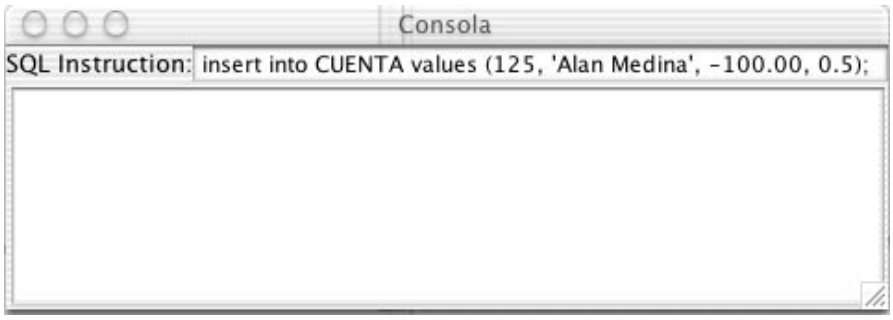

Figura 5.8: Captura de instrucciones SQL en la consola de usuario.

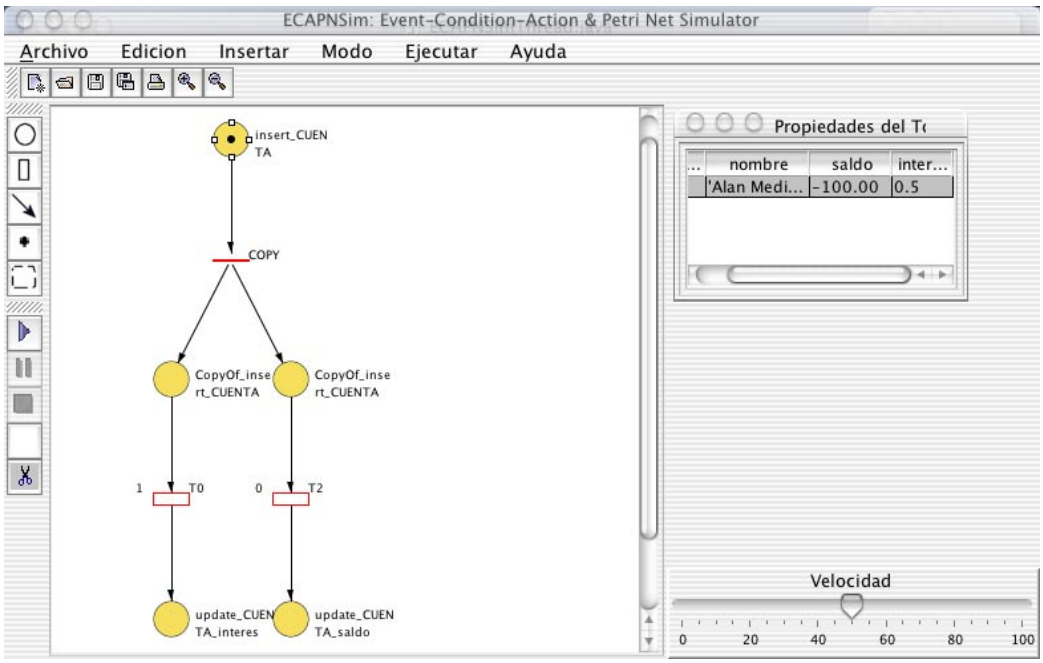

Figura 5.9: Token agregado en la CCPN.

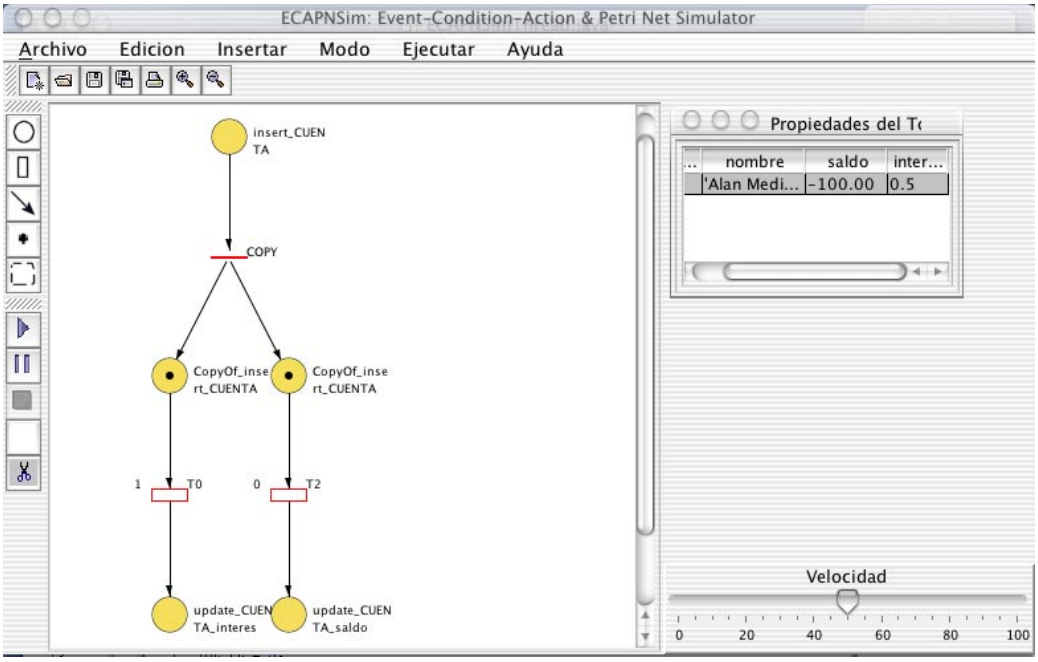

Figura 5.10: Duplicación del token.

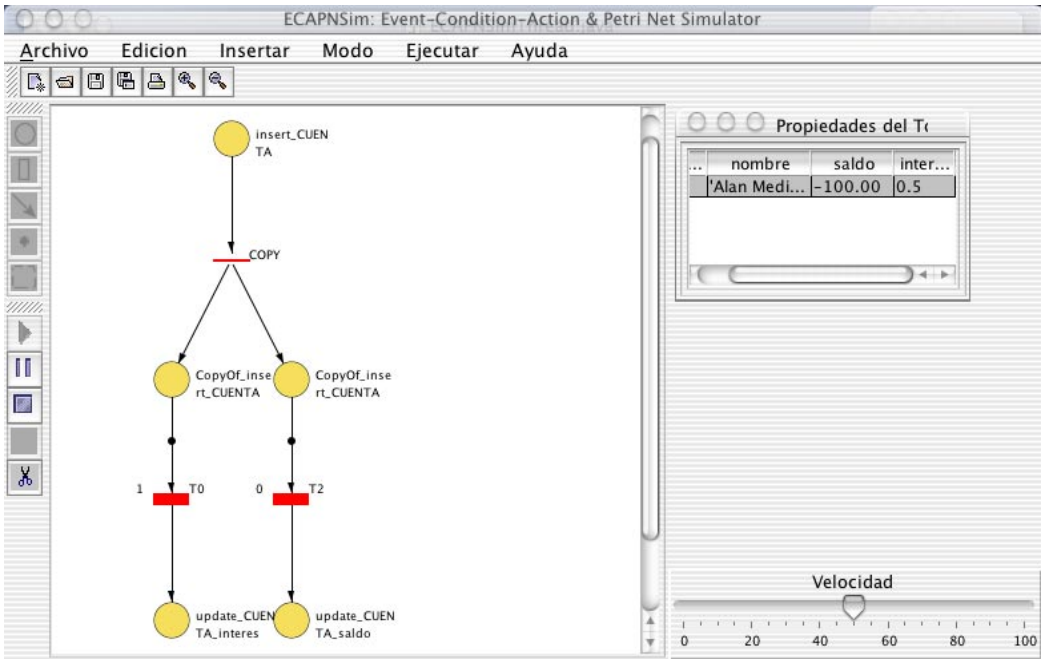

Figura 5.11: Evaluación de tokens.

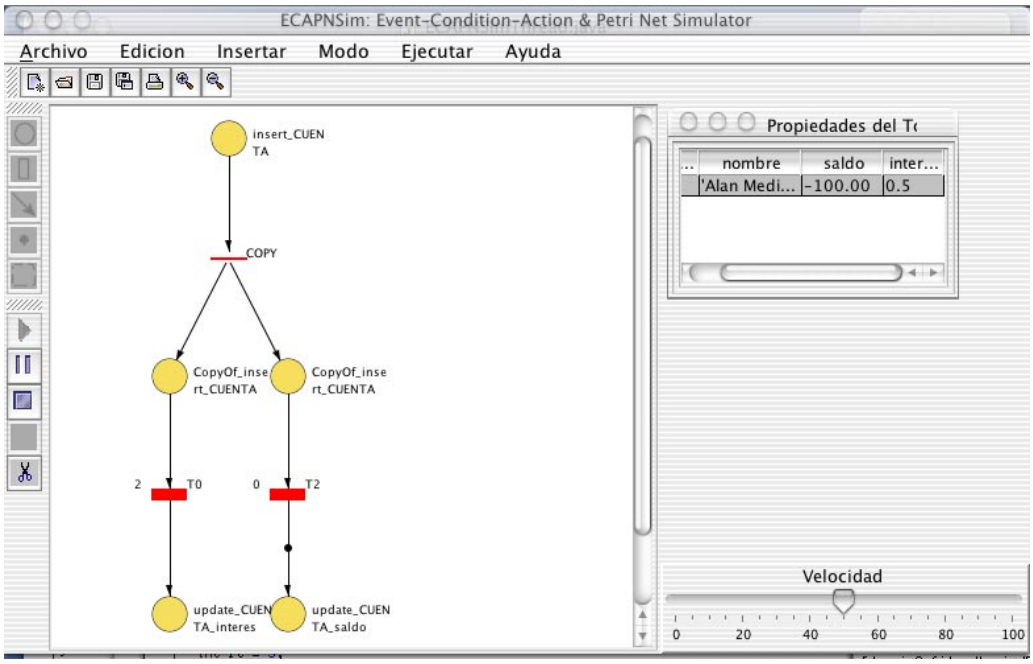

Figura 5.12: Disparo de una de las reglas.

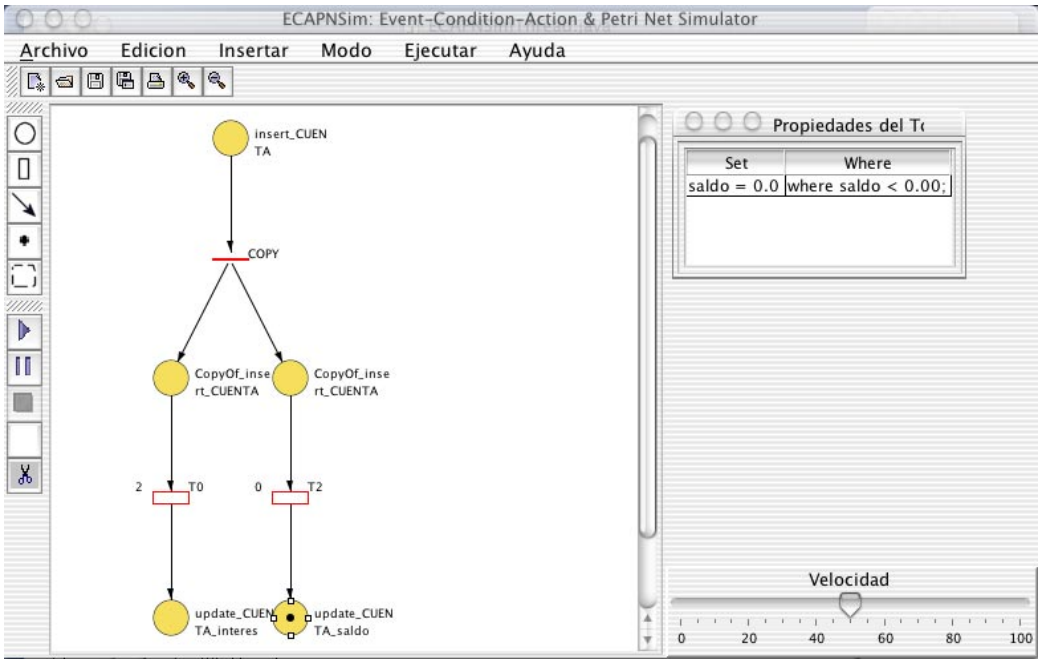

Figura 5.13: Marca final de la CCPN.

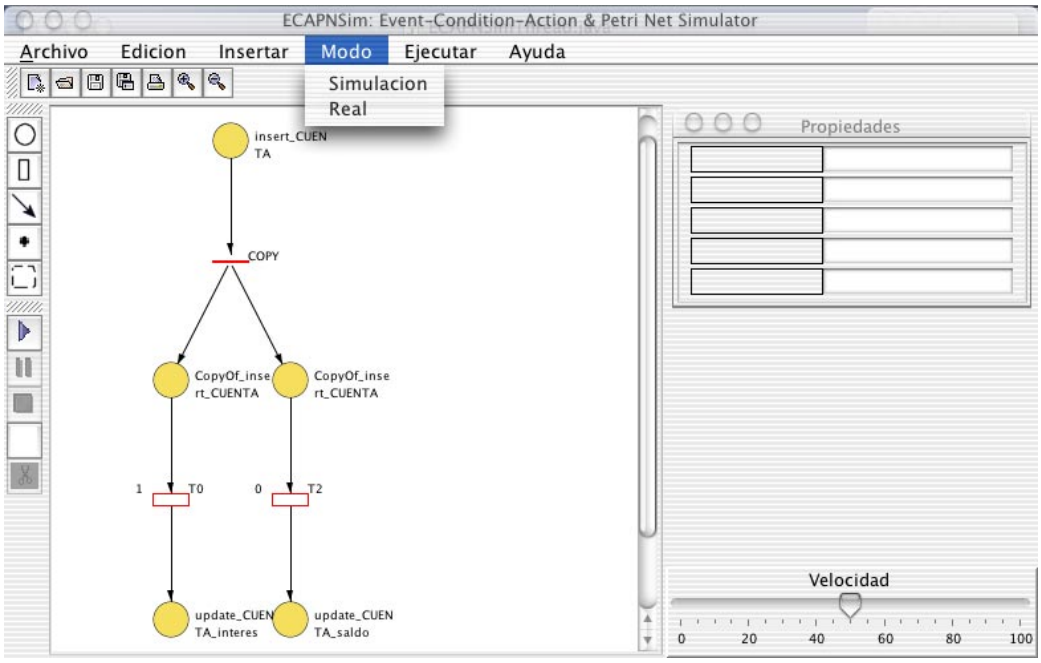

Figura 5.14: Activación del modo "Real".

desea implementar esta base de reglas en una BD, primero se activa la modalidad "Real" ubicada en el menú "Modo". Figura 5.14.

La ejecución de la misma instrucción en la consola de la figura 5.8, la información almacenada en la BD es como se muestra en la figura 5.15, donde el campo SALDO no almacenó el valor de -100.00. El disparo de la regla asignó el valor de 0.00 a éste campo.

5.4. Comentarios finales

El desarrollo de simuladores es importante para analizar el comportamiento de sistemas antes de implementarlos en un ambiente real. Las PN se utilizan para modelar y simular sistemas manejados por eventos. Existen editores de PN para diferentes tipos de sistemas [14], sin embargo, no hay herramientas que simulen el comportamiento de bases de reglas ECA, de ahí la importancia del desarrollo de la interfaz ECAPNSim.

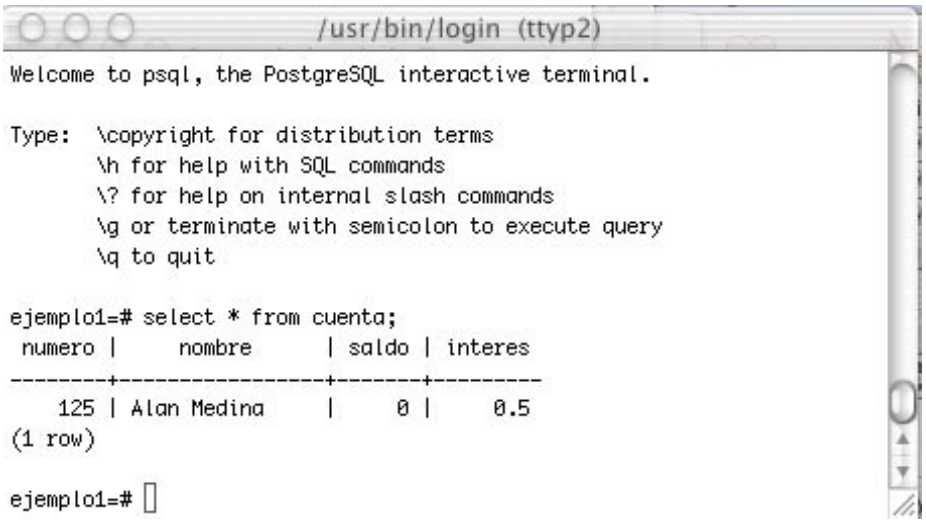

Figura 5.15: Resultado de la BD en el disparo de la regla.

En éste capítulo se presenta una descripción de la implementación de ECAPNSim, mostrando la aplicación de la POO en la representación de los elementos de la CCPN como objetos.

ECAPNSim está desarrollado en la plataforma MAC OS X Server, sin embargo, su implementación completamente en Java, permite hacerlo portable hacia otros sistemas operativos. Esto se considerará para futuras extensiones del ECAPNSim.

El ambiente de trabajo de ECAPNSim ofrece una interfaz amigable, con barras de menús e íconos de edición de CCPN, ejecución de la simulación y manejo de archivos. Cualquier usuario familiarizado con interfaces gráficas será capaz de manejar el ECAPNSim fácilmente.

Para el análisis de las reglas ECA se han desarrollado diversos enfoques [36] [40] [41] y, por otro lado, las BDA que ofrecen la declaración de reglas activas [3] [5] [8]. ECAPNSim ofrece éstas dos características de una base de reglas ECA a la vez. Con la simulación de la CCPN se analiza el comportamiento de las reglas y con la conexión de ECAPNSim con la BD se le proporciona dinamismo a la BD.

Capítulo 6

Casos de estudio

En este capítulo presentamos ejemplos de bases de reglas activas modeladas como CCPN y simuladas en el ECAPNSim. Las reglas fueron tomadas de [42], [4] y [11]. El capítulo se divide en tres secciones: la sección 6.1 muestra la simulación de las bases de reglas en ECAPNSim, la sección 6.2 muestra la conexión de ECAPNSim con una BD definida en Postgres (tomada como BD pasiva) y en la sección 6.3 se hacen unos comentarios sobre los casos de estudio.

6.1. Simulación de base de reglas ECA

En el modo de "Simulación", el ECAPNSim no interactúa con la BD, solamente muestra el desempeño de las reglas bajo la entrada de ciertas instrucciones SQL, las cuales modifican el estado de la BD. A continuación se presenta la simulación de bases de reglas ECA.

Ejemplo 1

La base de reglas de este ejemplo se tomó de [42] y su descripción es la siguiente:

Regla 1: Cuando el salario de un empleado se modifica, si el valor actualizado es menor de $1,000.00, entonces el salario se incrementa en un 10 %.

Regla 2: Cuando el salario de un empleado se modifica, el presupuesto del departamento al que pertenece se incrementa con $1,000.00 más.

Regla 3: Cuando el salario de un empleado se modifica, si el salario es mayor de $10,000.00, entonces el empleado se asigna al departamento "Departamento Rico".

Regla 4: Cuando el presupuesto de un departamento de modifica, si el presupuesto es mayor de $1,000.00, entonces el departamento pertenece a la clase "Presupuesto no vacío".

La traducción de éstas cuatro reglas a la sintaxis de reglas ECA propuesto es:

```
ON update_EMPLEADO_salario
IF salario < 1000.00
THEN update EMPLEADO set salario = salario*1.1 where
    id = update.id;

ON update_EMPLEADO_salario
IF TRUE
THEN update DEPARTAMENTO set presupuesto = presupuesto+1000
    where nombre = update.departamento;

ON update_EMPLEADO_salario
IF salario > 10000.00
THEN update EMPLEADO set departamento = 'Departamento Rico'
    where id = update.id;

ON update_DEPARTAMENTO_presupuesto
IF presupuesto > 1000
THEN update DEPARTAMENTO set clase = 'Presupuesto No Vacio'
    where nombre = update.nombre;
```

Almacenamos esta definición en el archivo tesis1.eca

Las tablas que forman parte de la BD son:

```
EMPLEADO(id, nombre, salario, departamento)
DEPARTAMENTO(nombre, presupuesto, clase)
```

La conversión de estas reglas a una CCPN se muestra en la figura 6.1, donde se observa que las

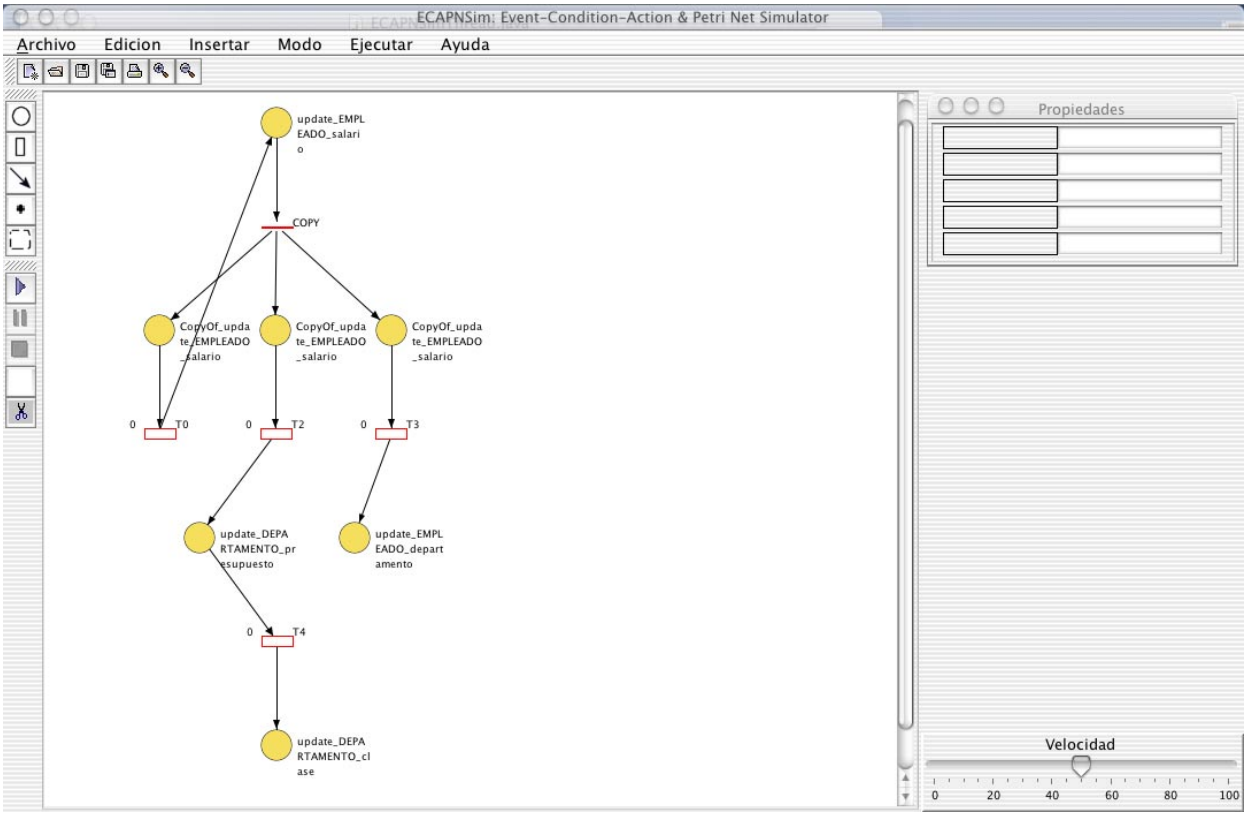

Figura 6.1: CCPN obtenida de la base de reglas ECA.

reglas 1, 2 y 3 tienen el mismo evento activador, la acción de la regla 1 es el mismo evento que la activa y la acción de la regla 2 es el evento de la regla 4.

Para probar el comportamiento de la CCPN envíamos por medio de la consola la instrucción de asignarle $2,000.00 al presupuesto del departamento de Recursos Humanos. Figura 6.2.

El token generado a partir de ésta instrucción se coloca en el lugar correspondiente y activa la transición de la regla. Figura 6.3.

La transición evalúa la condición "presupuesto $>$ 1000" con la información contenida en el token. Como el token es de tipo "UPDATE" su estructura está conformada por los campos "SET" y "WHERE", con valores "presupuesto = 2000" y "where nombre = 'Recursos Humanos"', respectívamente. Se observa que la evaluación de la condición obtiene un valor verdadero, lo cual propicia

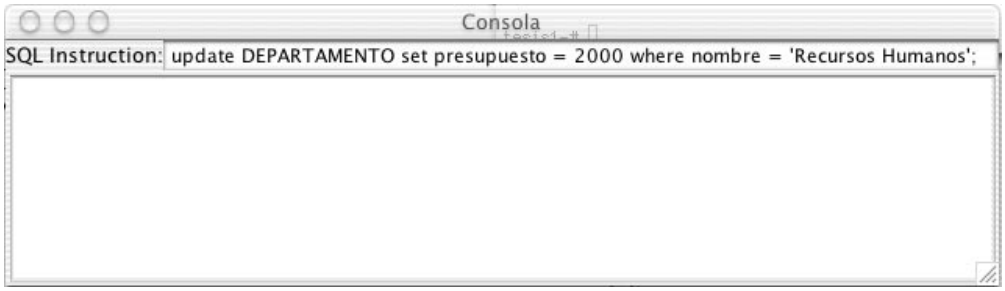

Figura 6.2: Instrucción SQL enviada por medio de la consola de usuario.

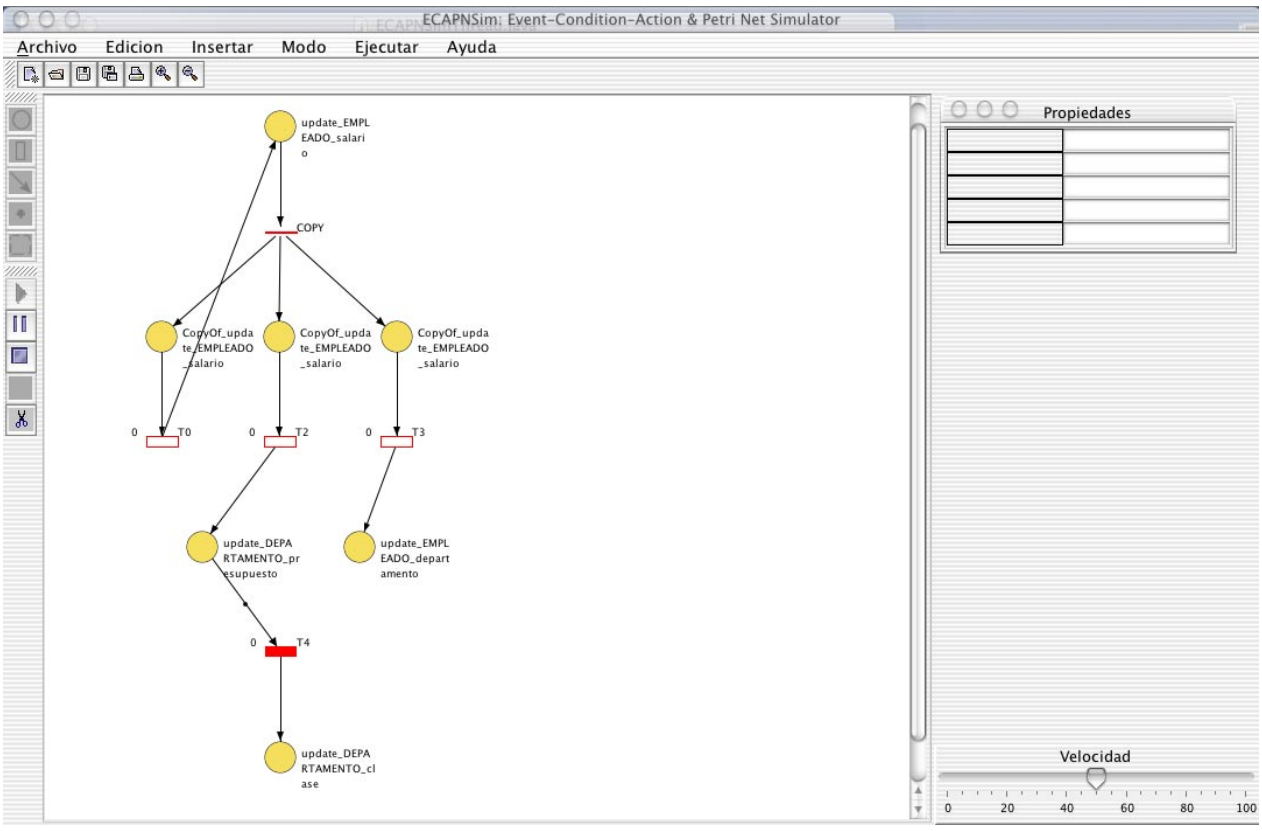

Figura 6.3: Activación de la transición con la presencia del token.

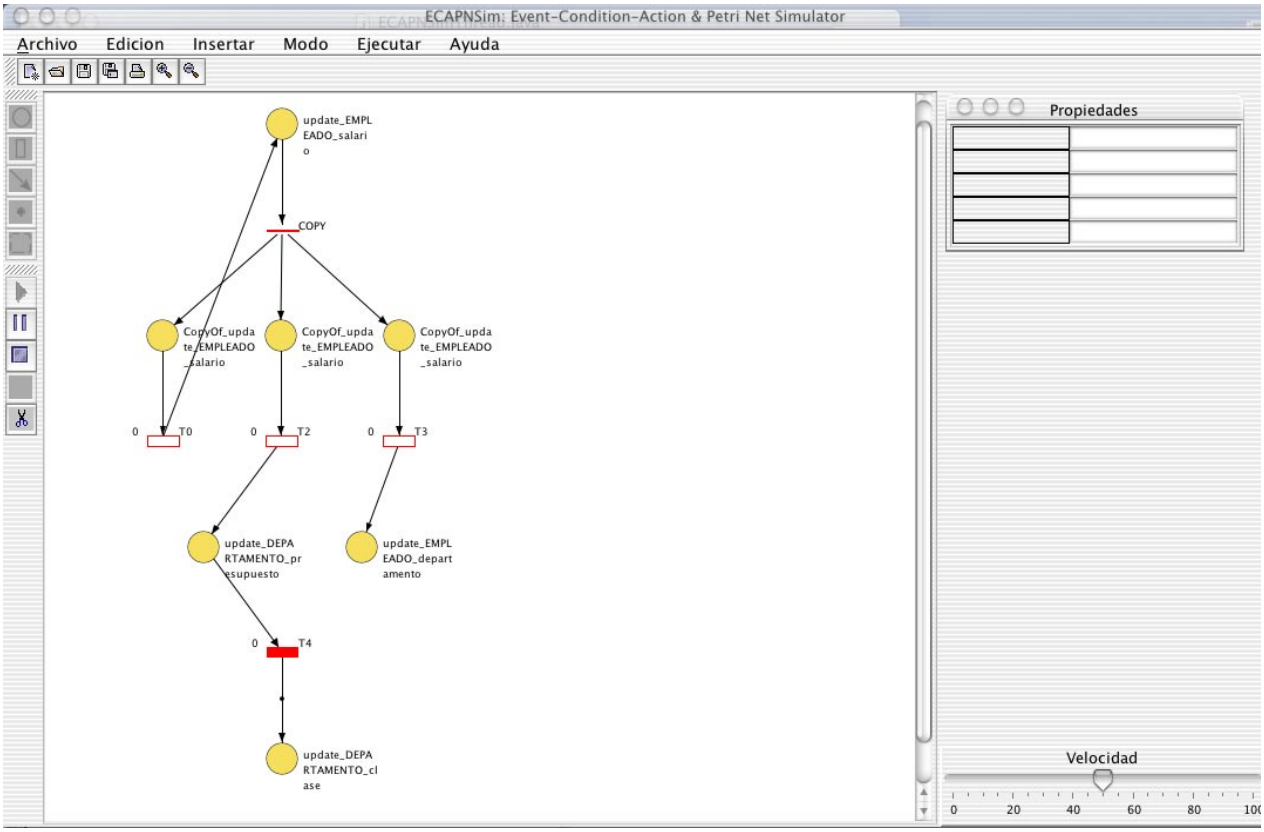

Figura 6.4: Disparo de la transición.

el disparo de la regla enviando el token con información de la acción al lugar de salida. Figura 6.4.

El disparo de la transición produce una marca final en la CCPN como se muestra en la figura 6.5.

Ejemplo 2

La base de reglas de este ejemplo se tomó de [4] y su descripción es la siguiente:

Regla 1: Cuando el salario de un empleado se modifica, si la persona a la que se le modificó el salario se llama Mary, entonces el salario de John se modifica asignándole el nuevo salario de Mary más $1,000.00.

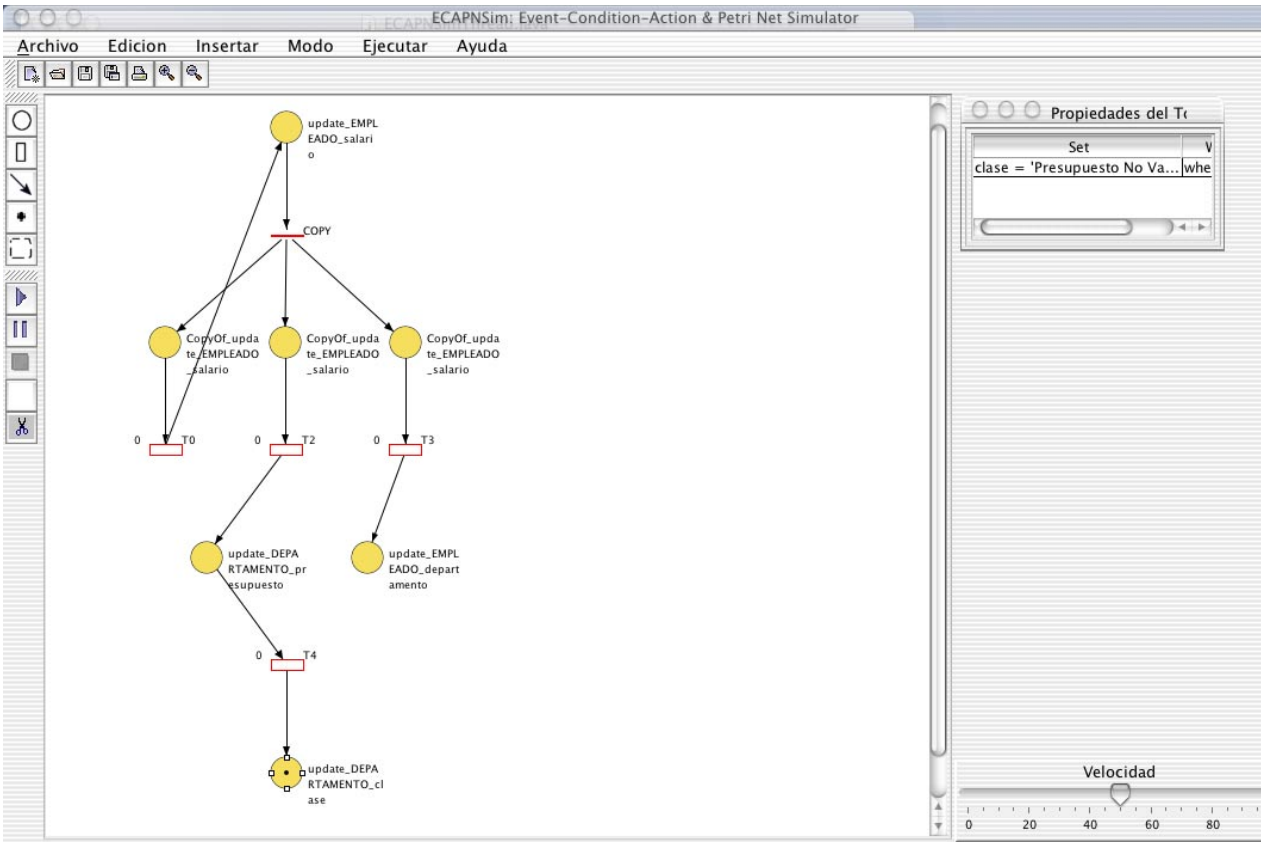

Figura 6.5: Marca final de la CCPN.

Regla 2: Cuando el salario de un empleado se modifica, si la persona a la que se le modificó el salario se llama John, entonces el salario de Tom se modifica asignándole el nuevo salario de John más $1,000.00.

Regla 3: Cuando el salario de un empleado se modifica, si la persona a la que se le modificó el salario se llama Tom, entonces el salario de Joe se modifica asignándole el nuevo salario de Tom más $1,000.00.

La traducción de éstas tres reglas a la sintaxis de reglas ECA propuesto es:

```
ON update_EMPLEADO_salario
IF nombre = 'Mary'
THEN update EMPLEADO set salario = update.salario+1000.00 where nombre = 'John';

ON update_EMPLEADO_salario
IF nombre = 'John'
THEN update EMPLEADO set salario = update.salario+1000.00 where nombre = 'Tom';

ON update_EMPLEADO_salario
IF nombre = 'Tom'
THEN update EMPLEADO set salario = update.salario+1000.00 where nombre = 'Joe';
```

Almacenamos esta definición en el archivo tesis2.eca

La tabla que forma parte de la BD es:

```
EMPLEADO(id, nombre, salario)
```

La conversión de estas reglas a una CCPN se muestra en la figura 6.6, donde se observa que las tres reglas tienen el mismo evento activador y la acción de cada una de ellas vuelve a ser el evento activador de las tres.

Para verificar el comportamiento de ésta CCPN, envíamos, a través de la consola de usuario, la asignación de $5,000.00 al salario de la empleada Mary traducido en una instrucción SQL. Figura 6.7.

El token generado a partir de ésta instrucción se coloca en el lugar correspondiente ("UPDATE_ EMPLEADO _SALARIO"). Como se trata del mismo evento activador para tres reglas distintas, se

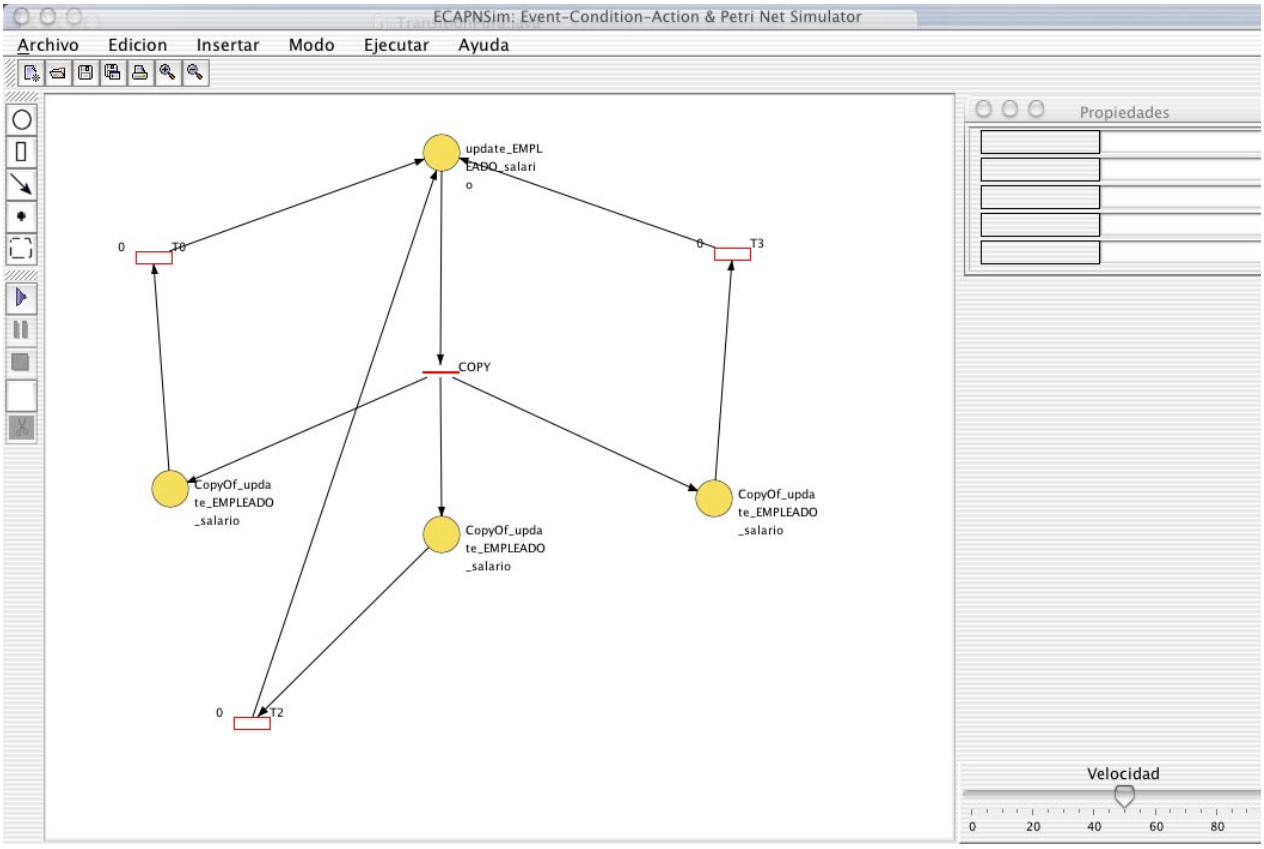

Figura 6.6: CCPN obtenida de la base de reglas.

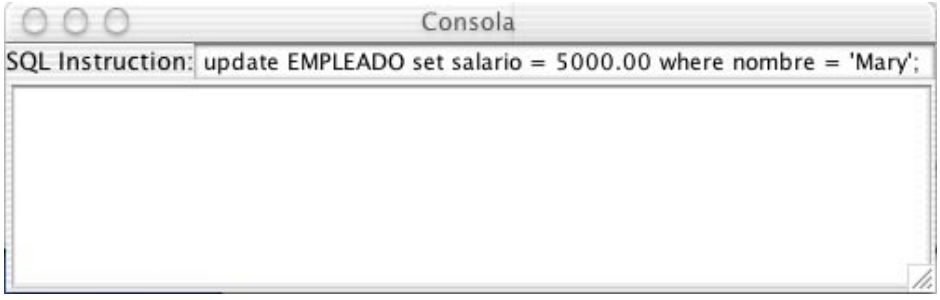

Figura 6.7: Instrucción SQL para modificar el salario de la empleada de nombre Mary.

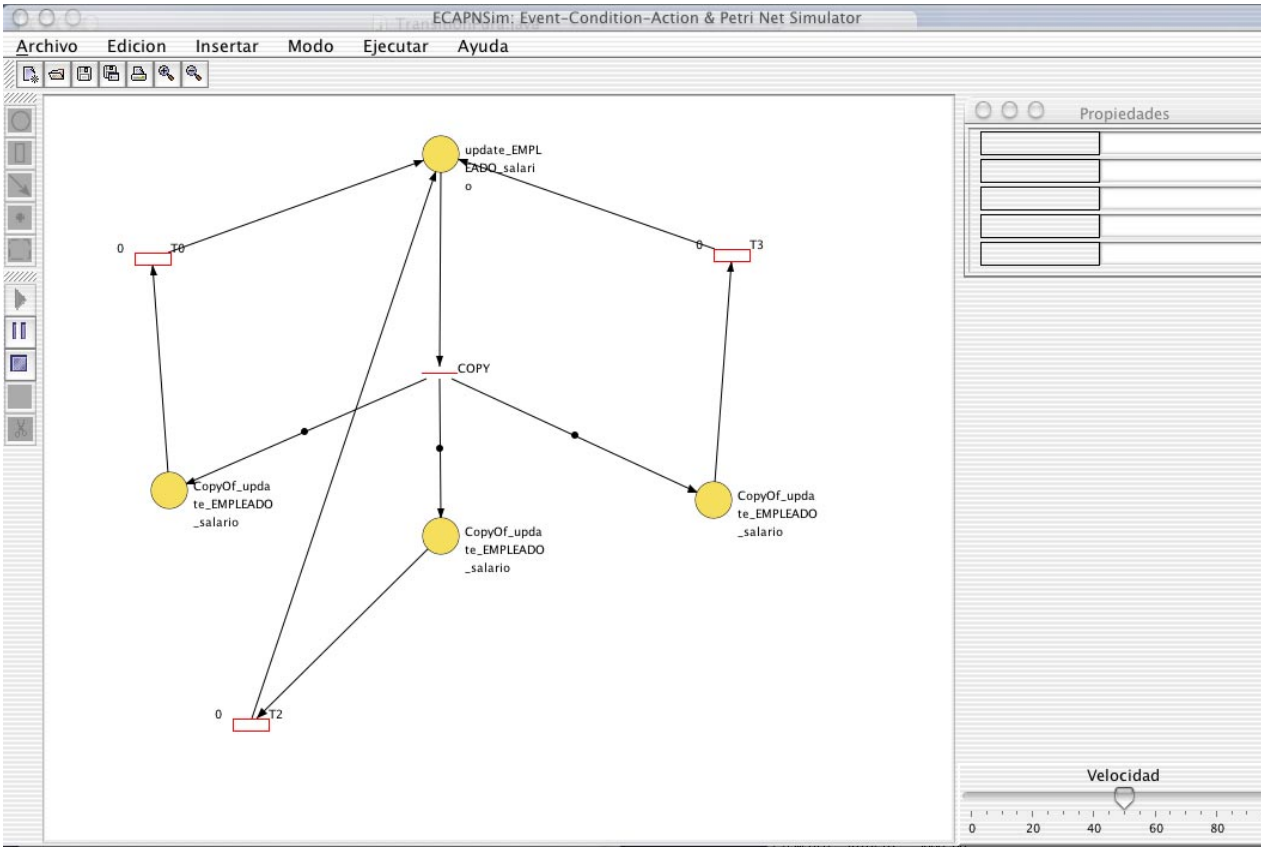

Figura 6.8: Generación de tokens idénticos al del evento original.

generan tres tokens idénticos y se colocan en lugares que representan a réplicas del evento original. Para esto, es necesario el uso de la transición tipo T_{copy}. Figura 6.8.

Los lugares de salida de la transición tipo T_{copy} almacenan ahora un token, lo que produce la activación de sus respectivas transiciones (figura 6.9). Sin embargo, solamente la transición T_0, la cual representa la condición "IF nombre = 'Mary"', se dispara (figura 6.10).

El disparo de la transición T_0 coloca un token de la acción en el lugar "UPDATE_ EMPLEADO _SALARIO", el cual contiene los datos "salario=6000.00" y "where nombre = 'John';". Se genera nuevamente la activación y disparo de la transición tipo T_{copy}, se activan las transiciones de las reglas y se dispara la transición T_2, correspondiente a la regla donde se verifica que el nombre sea

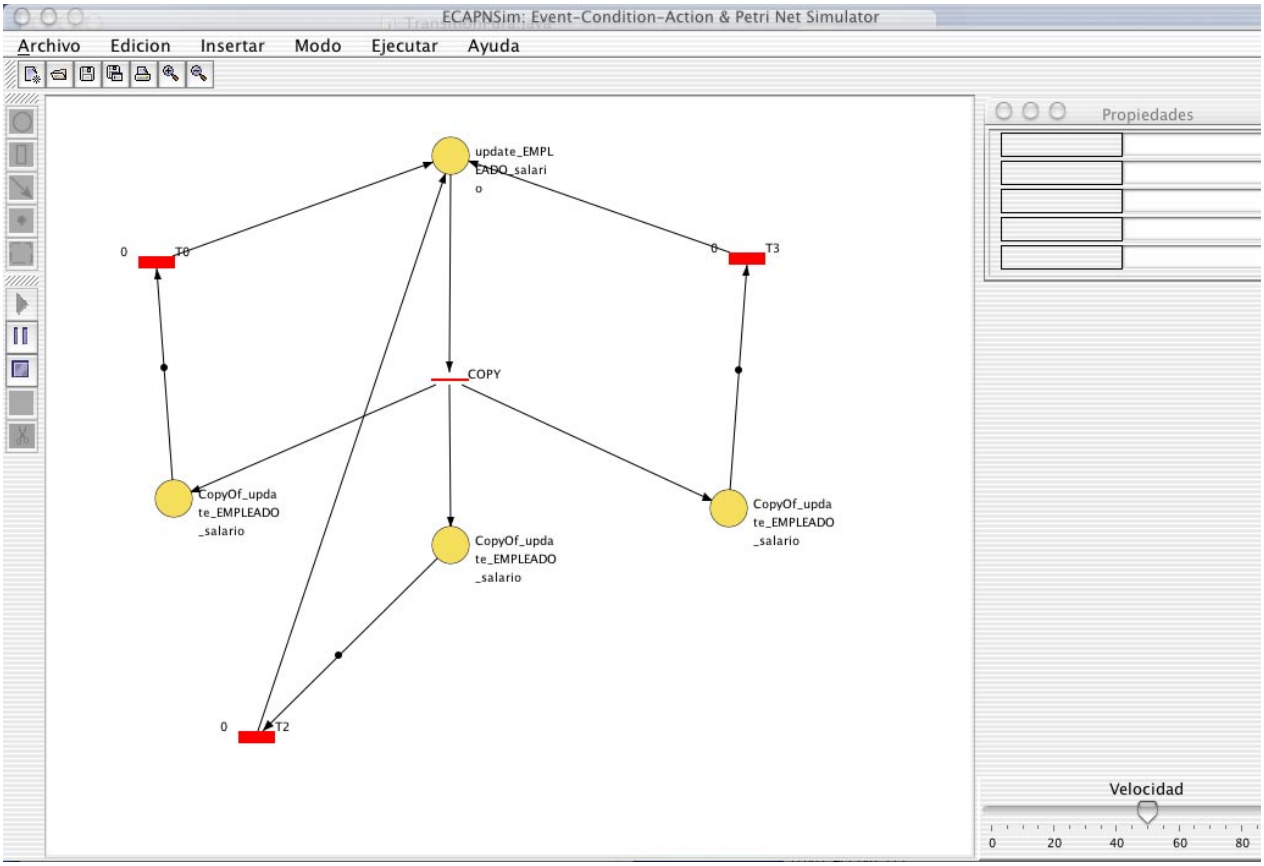

Figura 6.9: Activación de las transiciones de tipo T_{regla}.

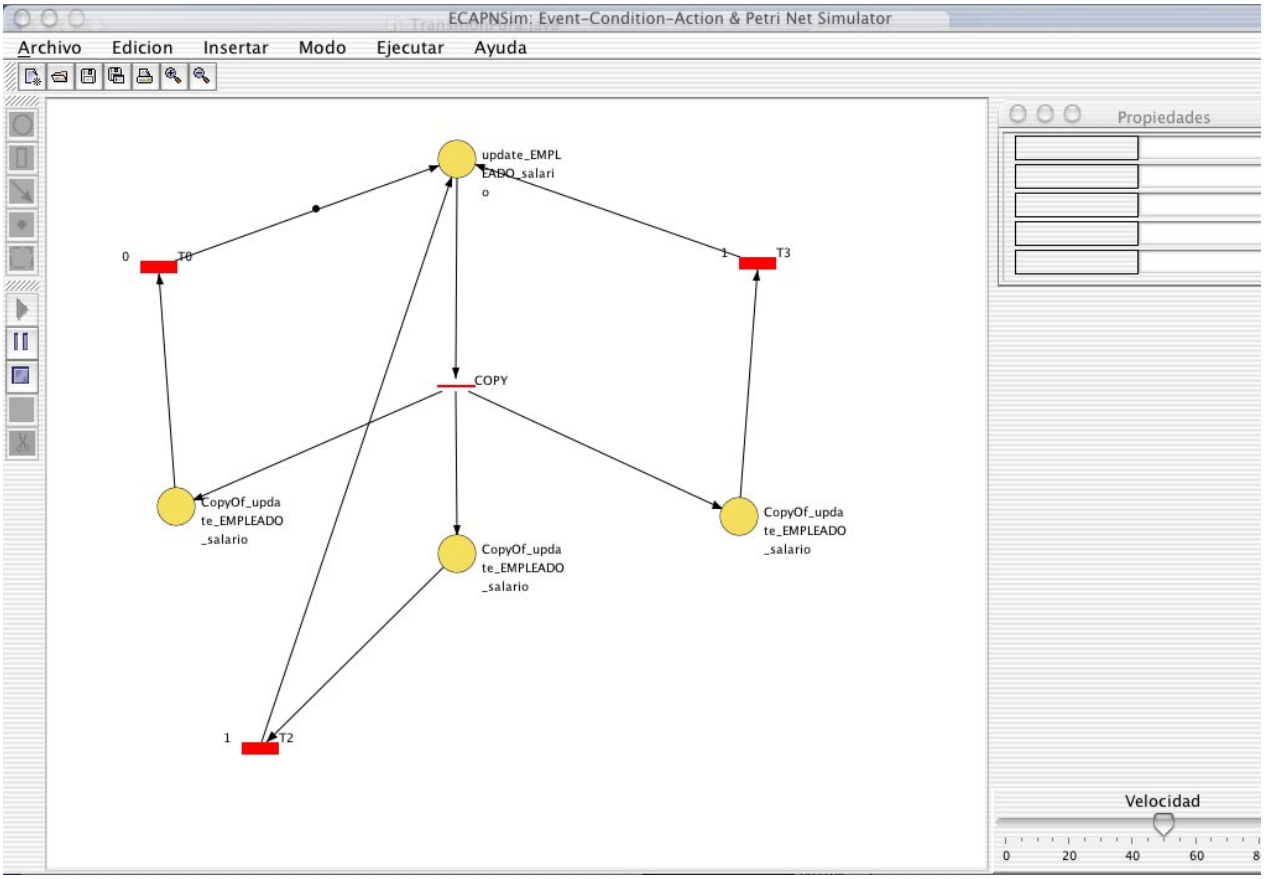

Figura 6.10: Disparo de la transición T_0.

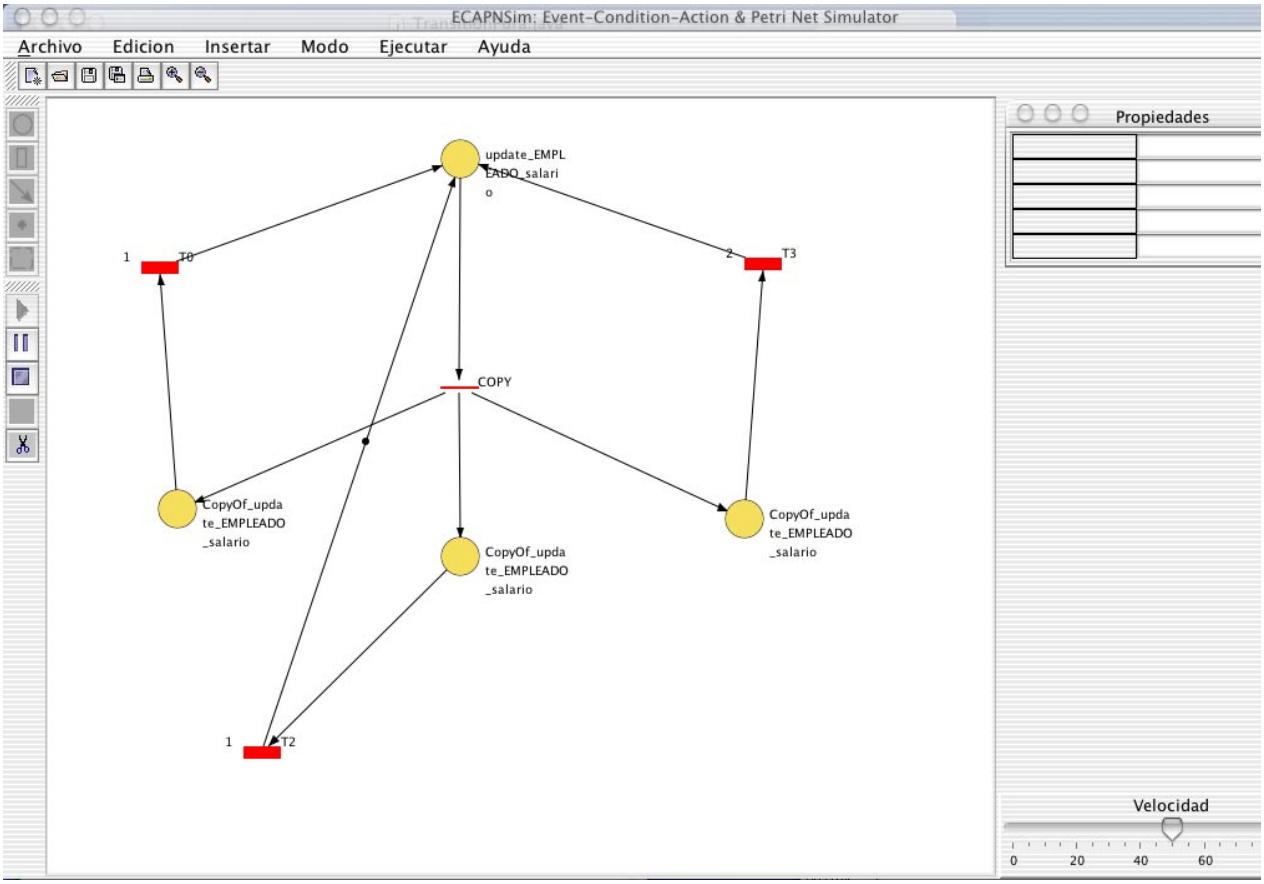

Figura 6.11: Disparo de la transición T_2.

"John". Figura 6.11.

Se genera el token de la acción con los datos "salario = 7000" y "where nombre='Tom';", el cual se pone en el lugar "UPDATE_EMPLEADO_SALARIO". Se vuelve a activar y disparar la transición T_{copy}, se activan las transiciones T_{regla} y ahora se dispara la transición T_3, correspondiente a la condición de la regla 3 (figura 6.12).

Se genera el token de la acción con los datos "salario = 8000" y "where nombre='Joe';", el cual se pone en el lugar "UPDATE_EMPLEADO_SALARIO". Se vuelve a activar y disparar la transición T_{copy}, se activan las transiciones T_{regla}, pero como el nombre del empleado en el token

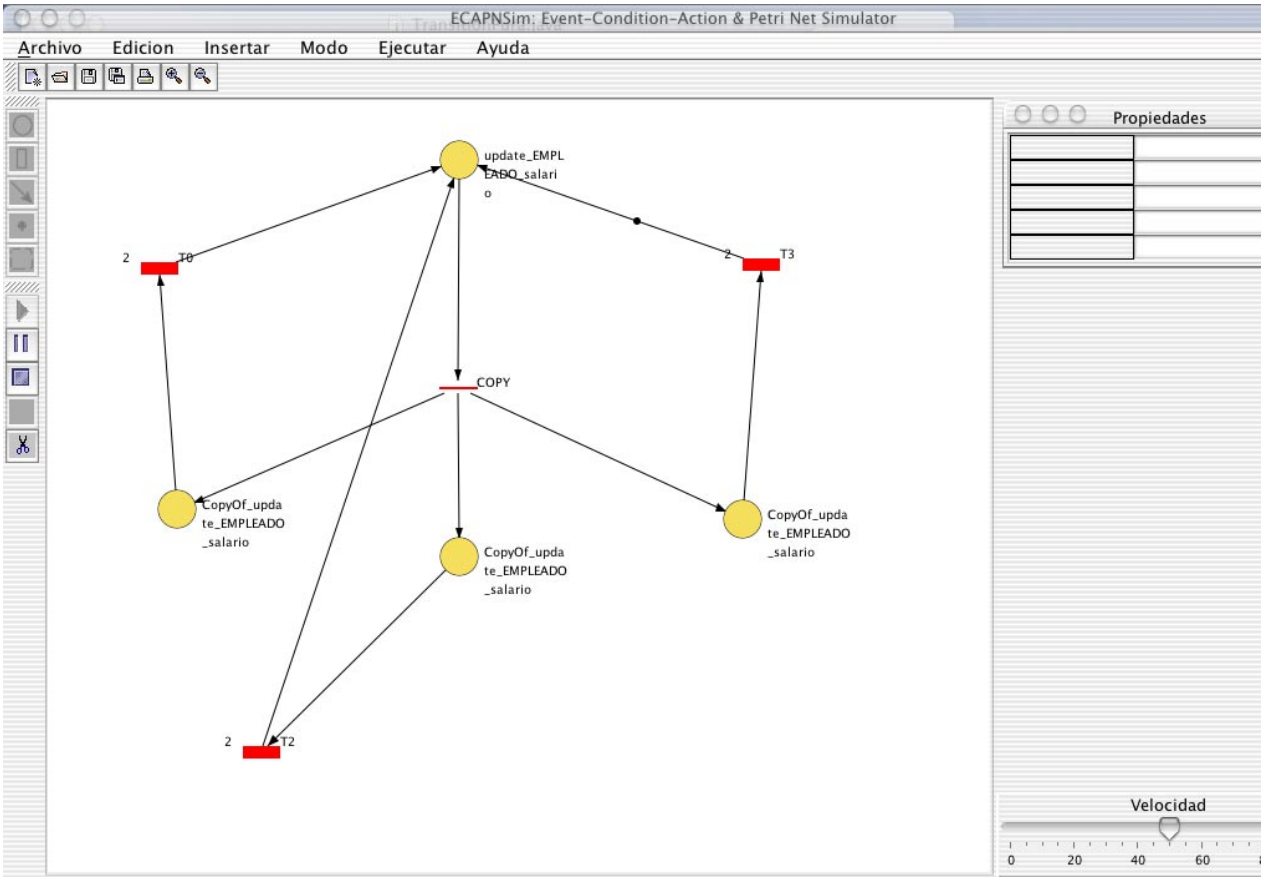

Figura 6.12: Disparo de la transición T_3.

es 'Joe', ninguna de ellas se cumple y por lo tanto, no se disparan.

Ejemplo 3

La base de reglas de este ejemplo se tomó de [11] y su 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 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 %.

La traducción de éstas cuatro reglas a la sintaxis de reglas ECA propuesto es:

```
ON update_PRIMA_cantidad
IF cantidad > 100
THEN update EMPLEADO set rango = update.rango+1 where emp_id = update.emp_id;

ON update_EMPLEADO_rango
IF rango > 5
THEN update PRIMA set cantidad = old.cantidad+rango*10
        where emp_id = update.emp_id;

ON insert_VENTAS
IF numero > 50
THEN update EMPLEADO set rango = old.rango+1 where emp_id = insert.emp_id;

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

Almacenamos esta definición en el archivo tesis3.eca
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 conversión de estas reglas a una CCPN se muestra en la figura 6.13, donde se observa que las tres reglas tienen el mismo evento activador y la acción de cada una de ellas vuelve a ser el evento activador de las tres.

Para probar el comportamiento de la CCPN envíamos por medio de la consola la instrucción de actualizar la prima al empleado con número de identificación = 3. Figura 6.14.

El token se coloca en el lugar "UPDATE_PRIMA_CANTIDAD" correspondiente a este evento (figura 6.15. Se activa la transición T_{10} (figura 6.16).

Como se cumple la condición de la regla 1, almacenada en la transición T_{10}, se dispara y se envía un token al lugar de salida "UPDATE_EMPLEADO_RANGO" (figura 6.17) con datos "rango = 4" y "wehere.emp_id = 3;"

Se activa la trancición tipo T_{copy}, generando dos tokens idénticos para evaluar a dos reglas respectívamente. Figuras 6.18 y 6.19.

Se activan las transiciones T_{11} y T_{14}, correspondientes a las reglas 2 y 3 (figura 6.20), pero como la condición de éstas transiciones no cumple con los datos del token, entonces no se dispara ninguna (figura 6.21).

Ahora, insertamos un registro a la tabla VENTAS, donde se agregará información de que las ventas del empleado 1 para el mes de septiembre fue de 55: (1, 9, 55). Se coloca el token con esta información en el lugar "INSERT_VENTAS", se activa la transición, se verifica la condición de la regla 3 y como las ventas registradas (55) es mayor que 50, entonces se genera el token correspondiente a la acción de la regla con la siguiente información ("salario=1510.00", "where emp_id = 1;"). Las figuras 6.22, 6.23, 6.24 muestran el proceso descrito.

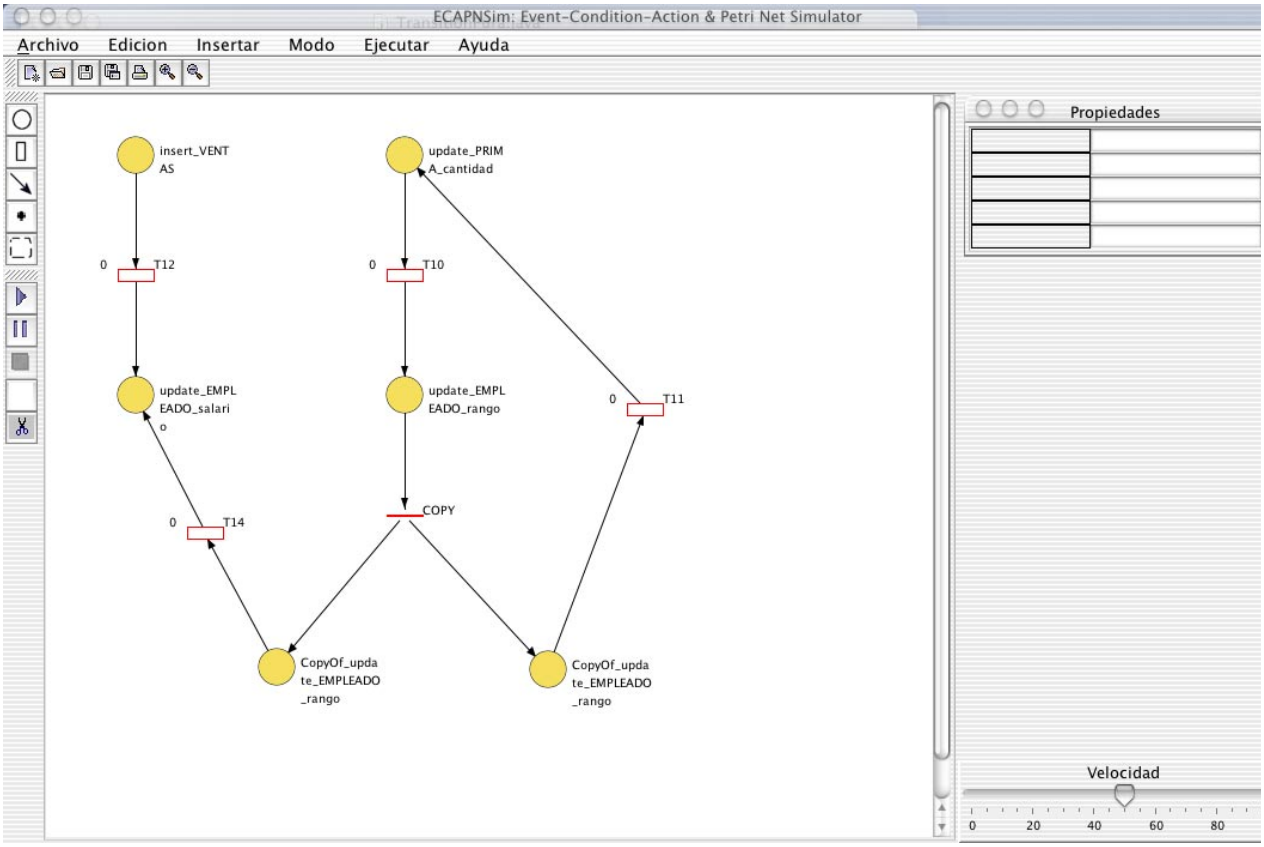

Figura 6.13: CCPN generada a partir de la base de reglas.

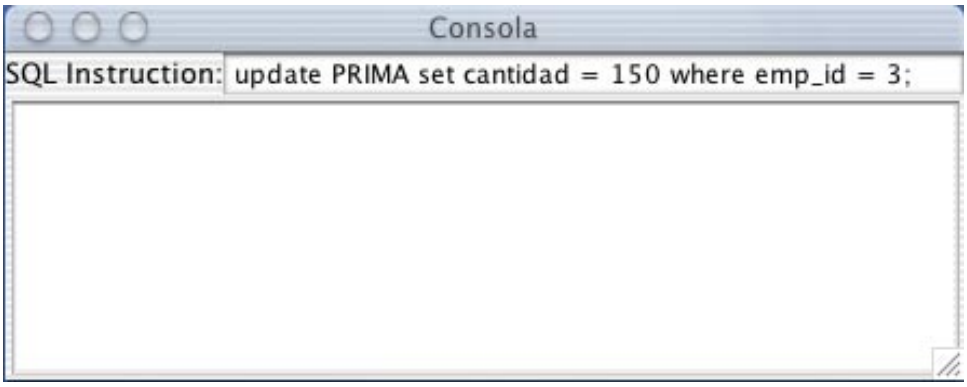

Figura 6.14: Instrucción para modificar la prima del empleado con identificación 3.

6.2. Conexión de ECAPNSim con Postgres

En el modo "Real" el ECAPNSim sí interactúa con la BD para ejecutar la acción de las reglas. En esta sección se retoman los ejemplos de la sección anterior pero ahora se ejecuta la acción en la BD.

Ejemplo 1

Considerando que la tabla DEPARTAMENTO contenga solamente el registro del departamento de Recursos Humanos, con un presupuesto de $350.00 y catalogado dentro de la clase "Austero". Figura 6.26.

El disparo de la regla mostrado en el ejemplo 1 de la sección anterior produce la modificación de la tabla, actualizando el valor del campo "presupuesto" asignándole $2,000.00 (objetivo de la instrucción de la consola) y modifica a la vez el campo "clase" asignándole la cadena "Presupuesto No Vacío" (acción de la regla). Figura 6.27.

Ejemplo 2

Considerando que la tabla EMPLEADO contenga solamente los registros presentados en la figura 6.28.

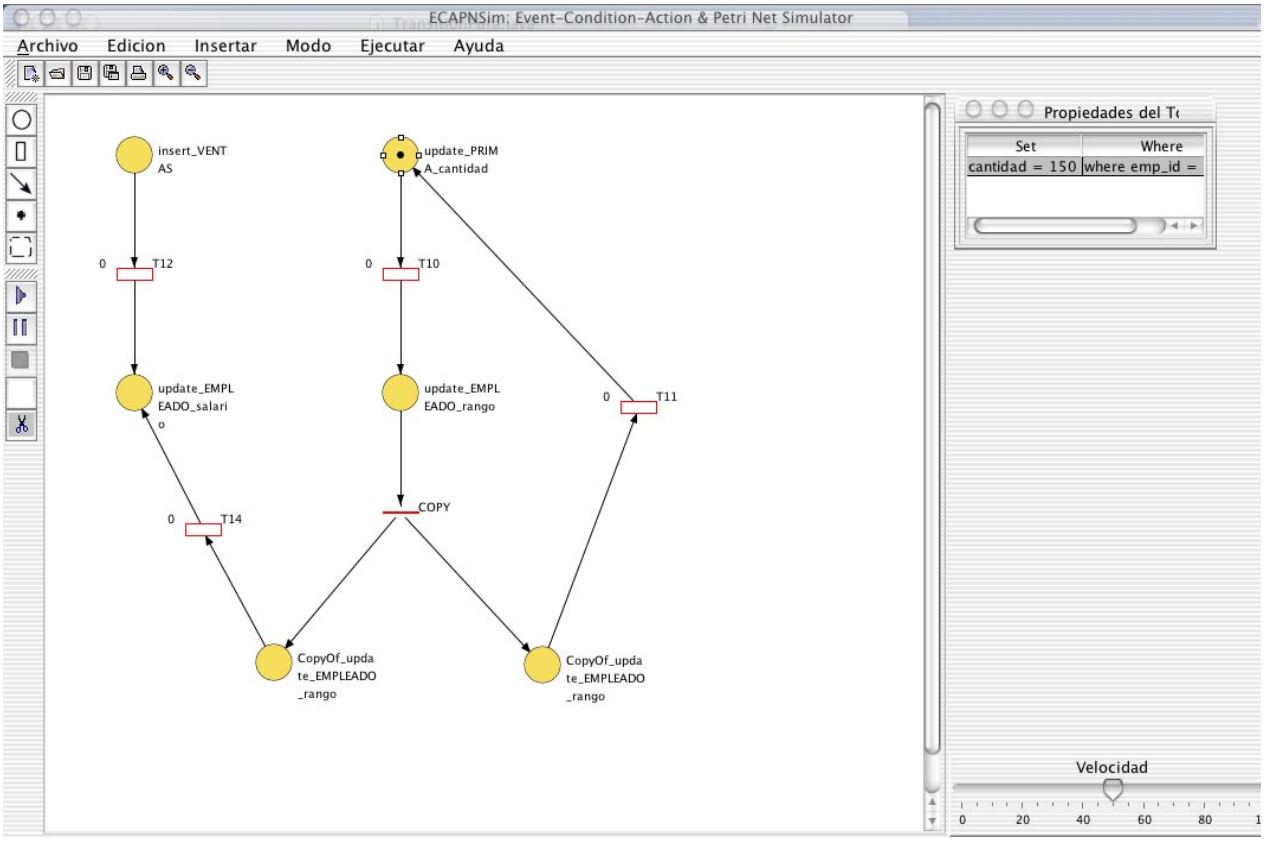

Figura 6.15: Colocación del token de la actualización en el lugar "UPDATE_PRIMA_CANTIDAD".

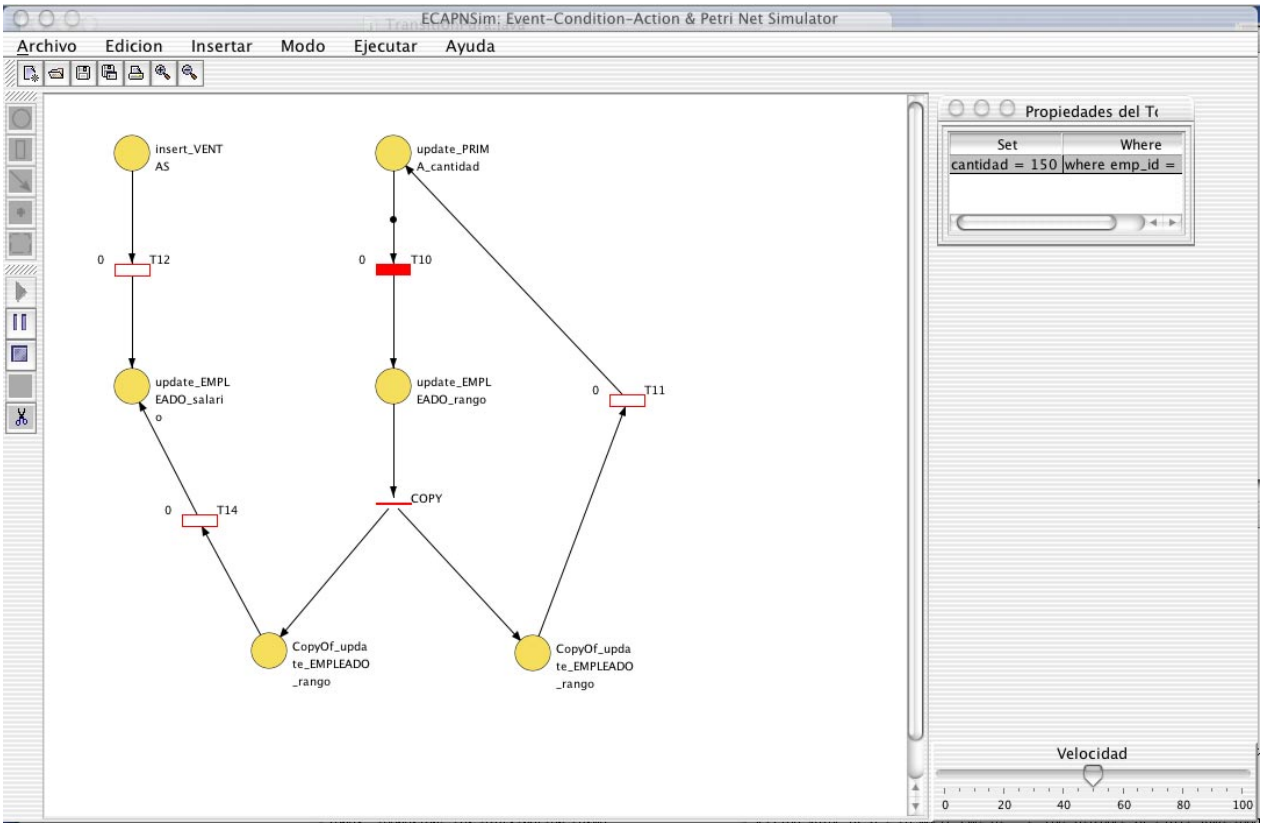

Figura 6.16: Activación de la transición T_{10} de la regla 1.

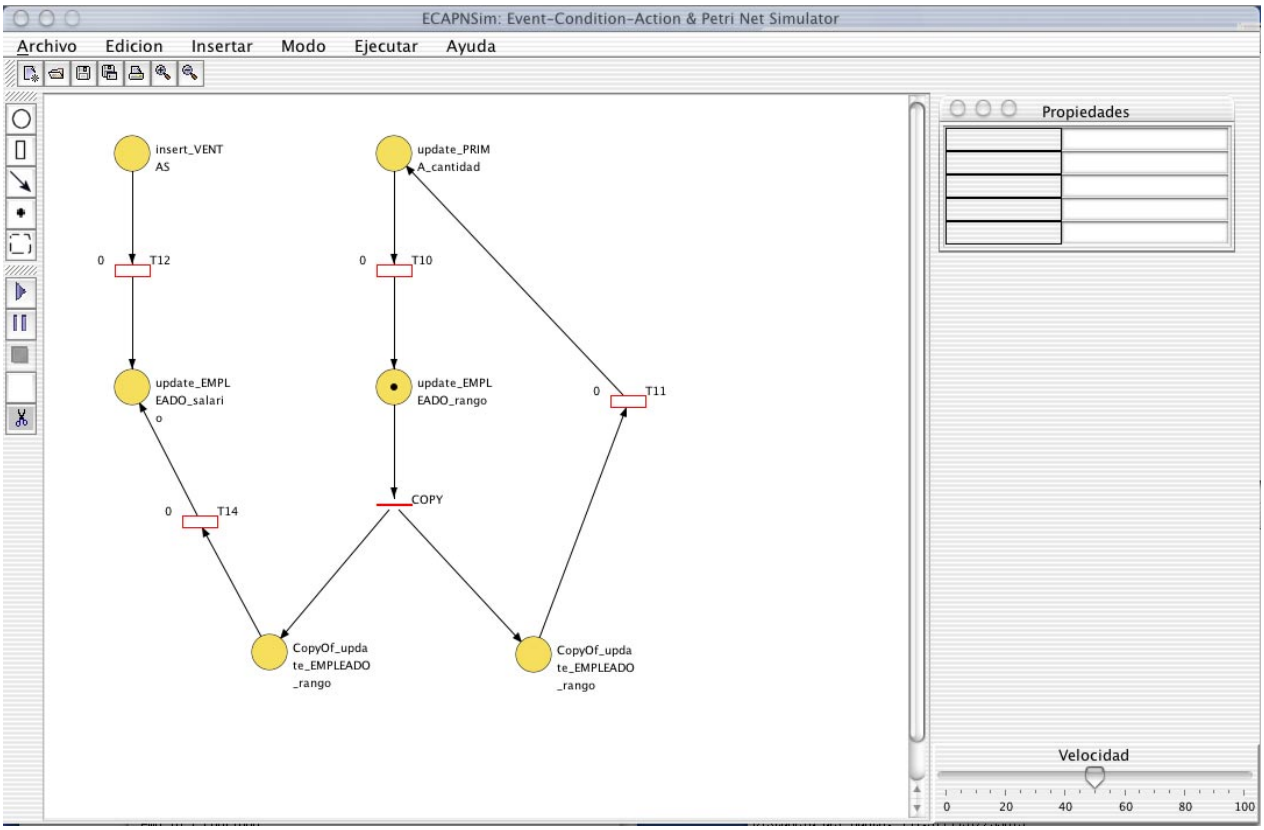

Figura 6.17: Se coloca un token en el lugar "UPDATE_EMPLEADO_RANGO", acción de la regla 1.

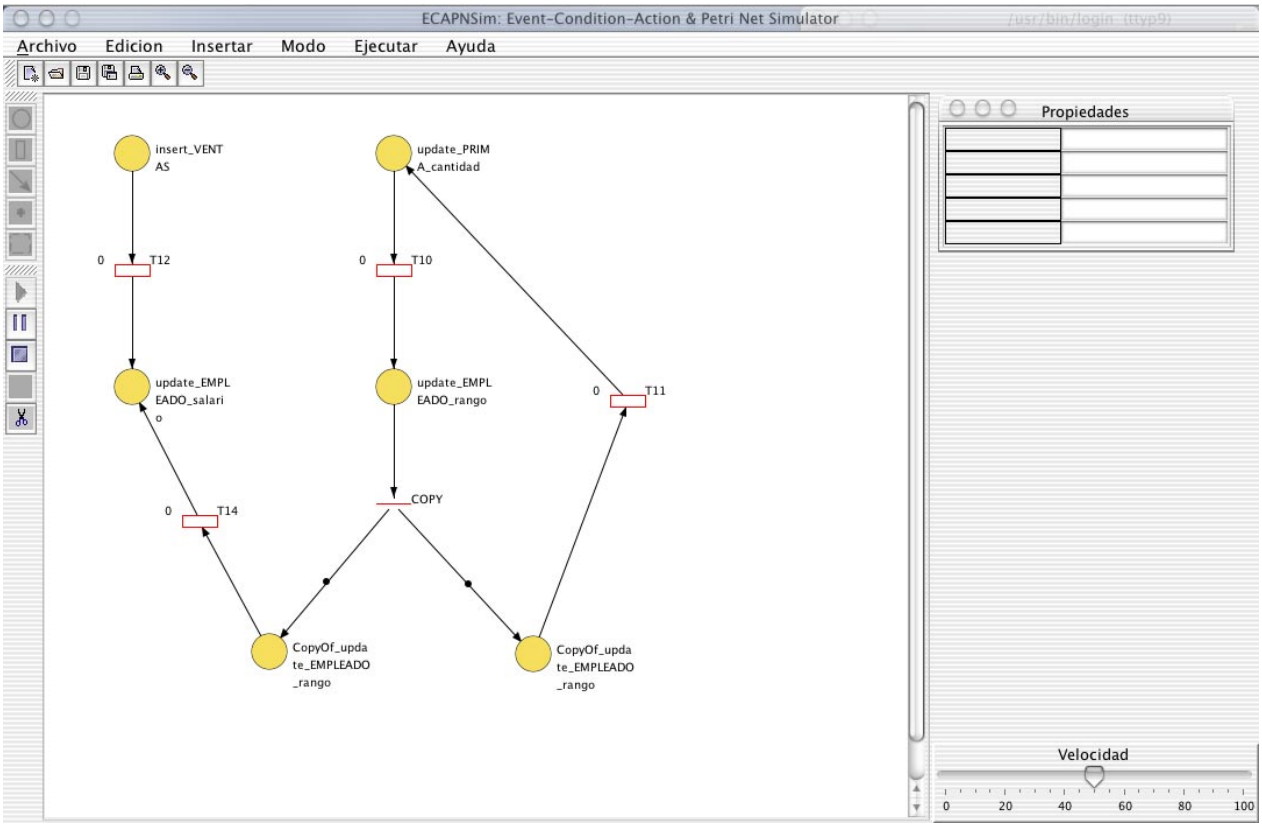

Figura 6.18: Duplicación del token para evaluar a dos reglas.

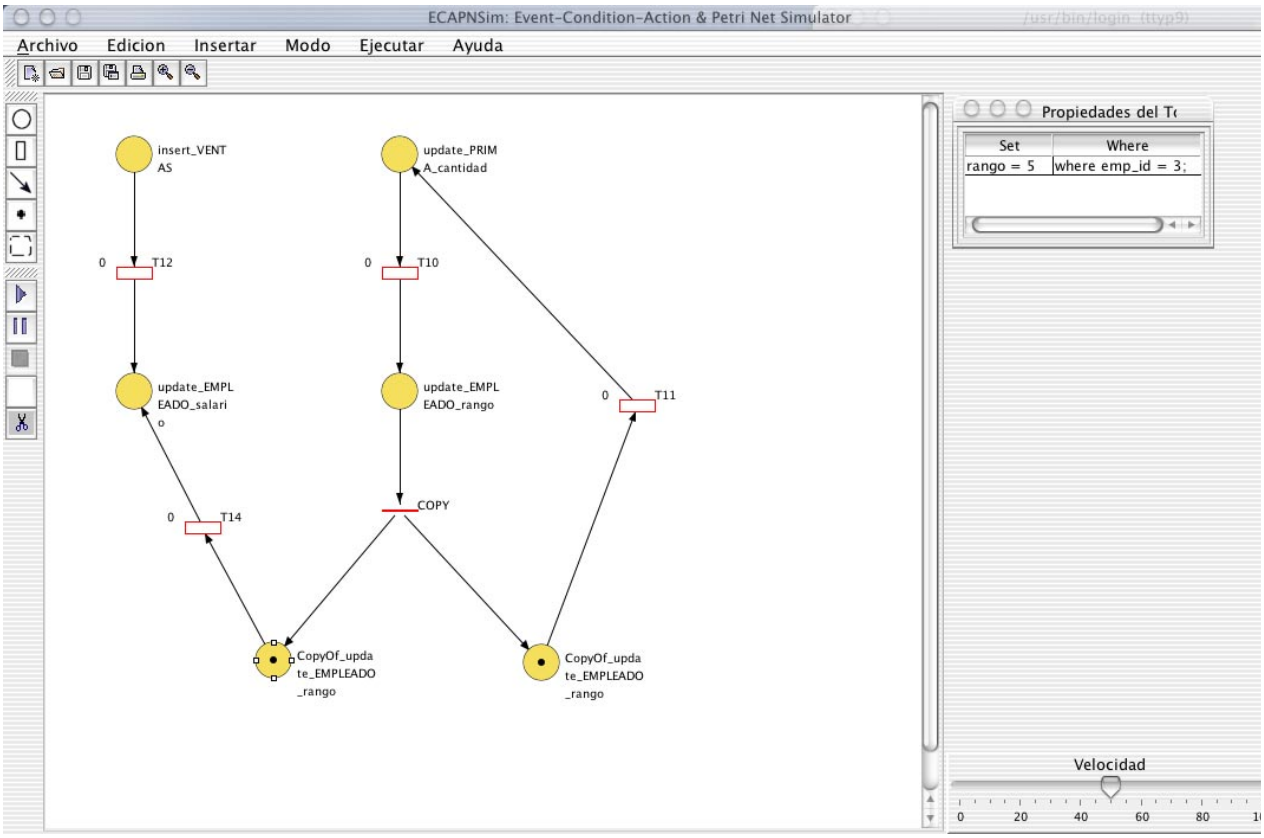

Figura 6.19: Tokens duplicados en dos lugares de la CCPN.

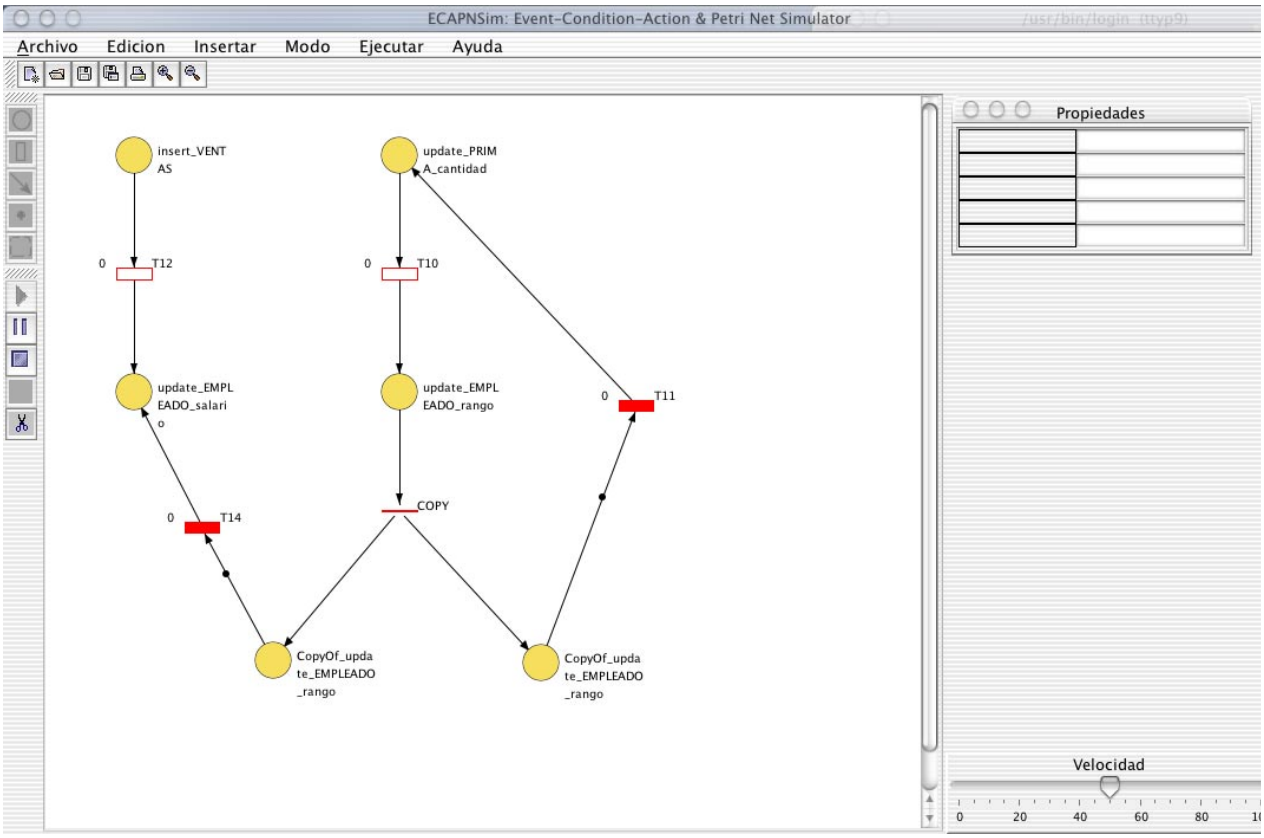

Figura 6.20: Activación de T_{11} y T_{14}.

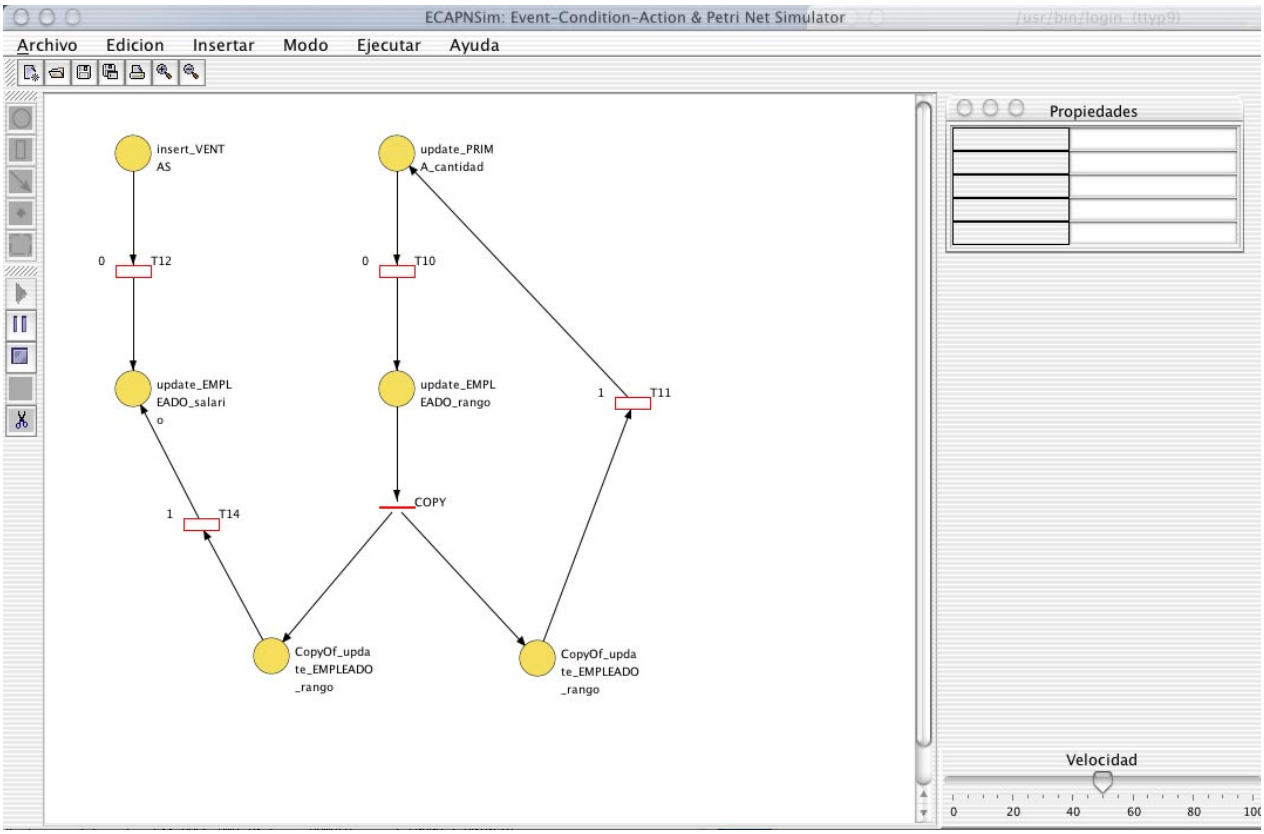

Figura 6.21: Las transiciones T_{11} y T_{14} no cumplen con la condición, por lo tanto no se disparan.

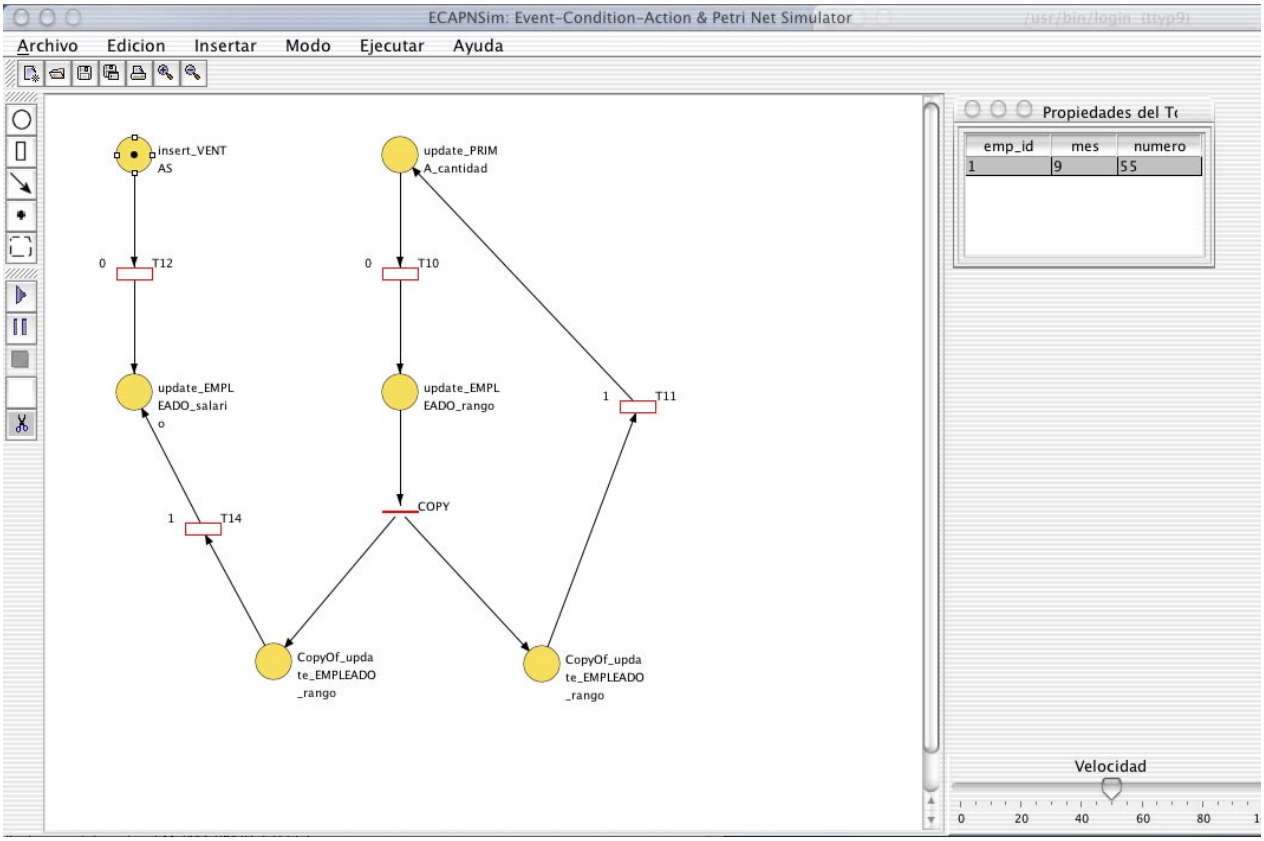

Figura 6.22: Se agrega el token en el lugar "INSERT_VENTAS".

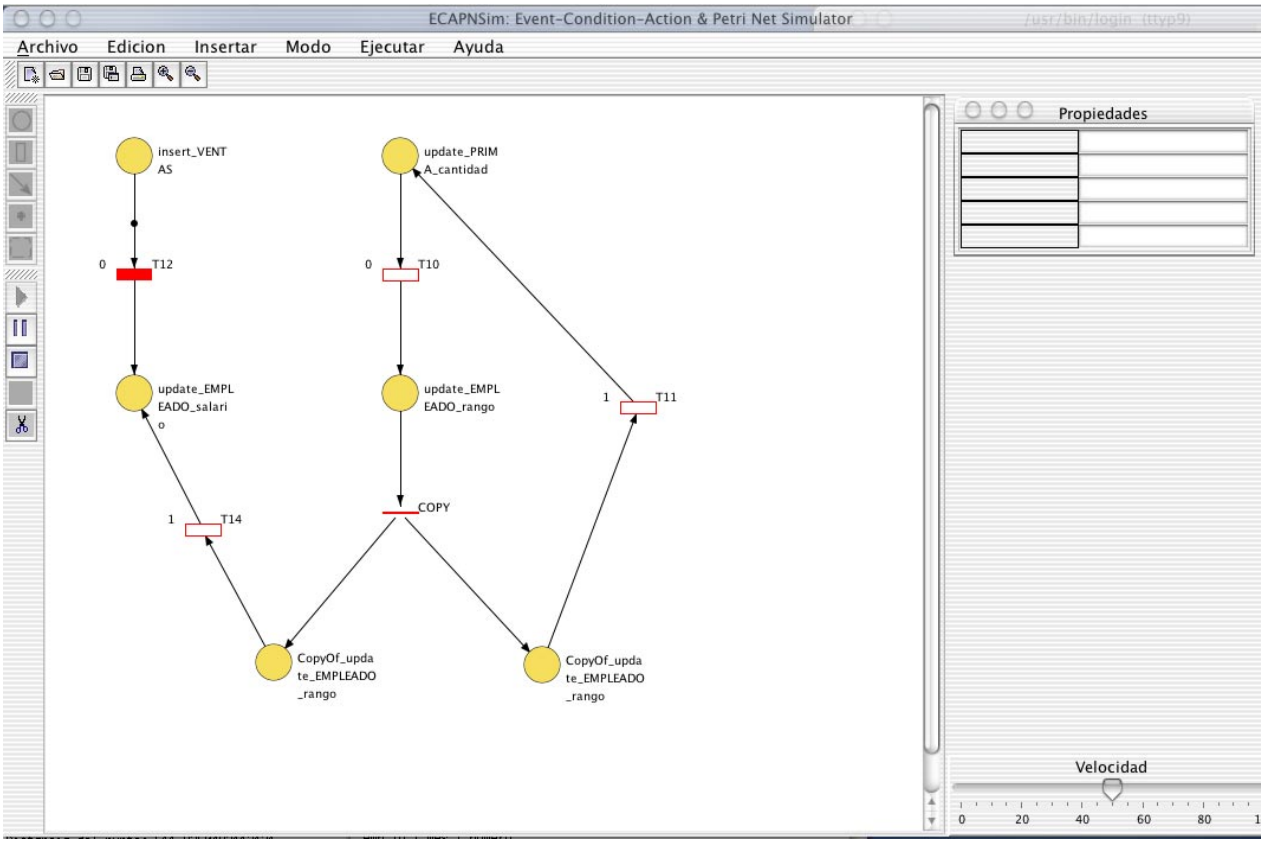

Figura 6.23: Activa la transición T_{12} de la regla 3.

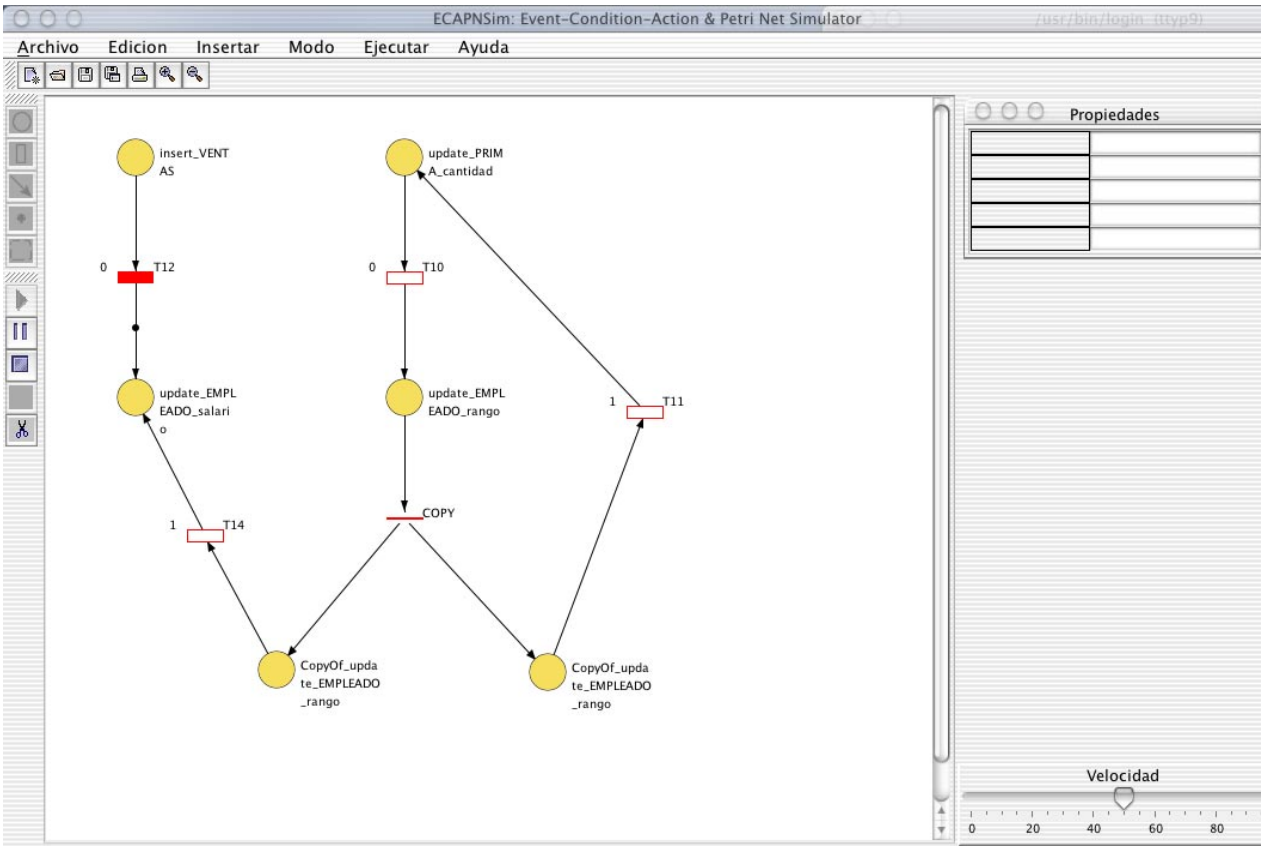

Figura 6.24: La transición T_{12} se dispara.

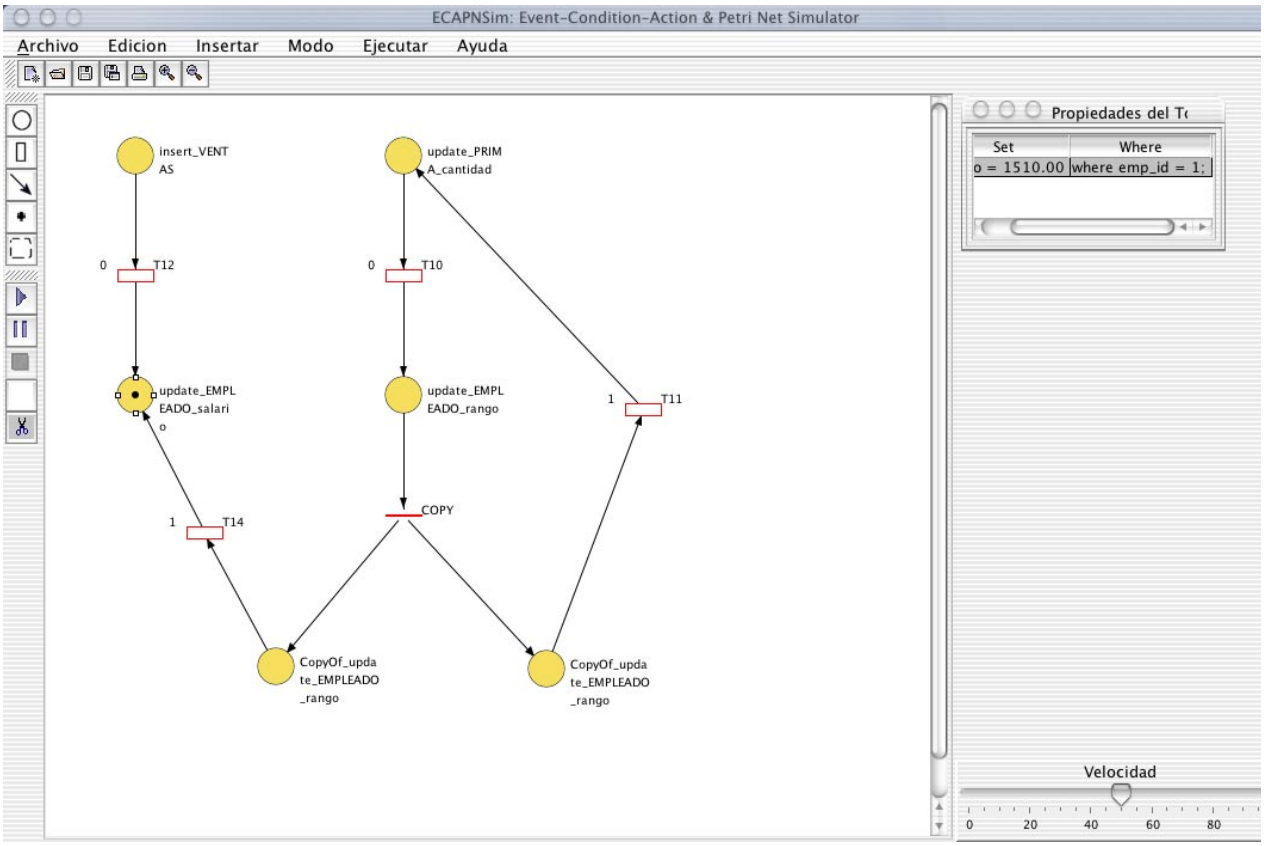

Figura 6.25: Se coloca el token correspondiente a la acción en el lugar "UPDATE_EMPLEADO_SALARIO".

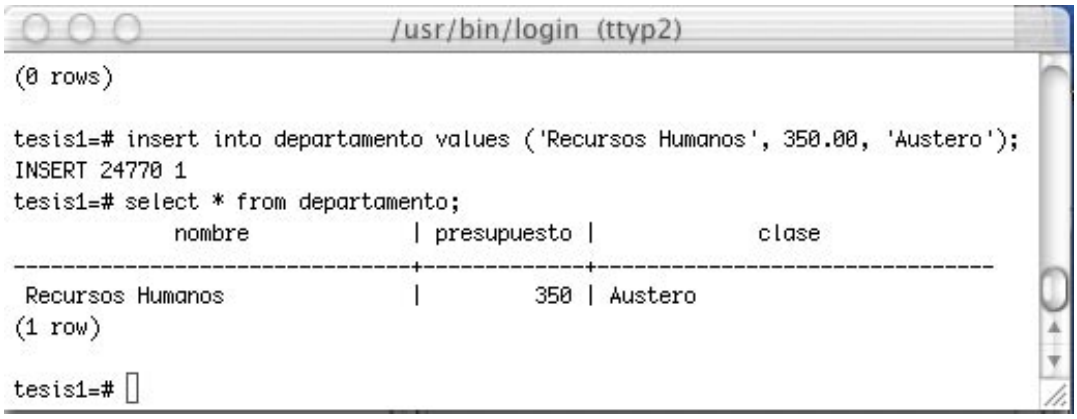

Figura 6.26: Estado de la tabla DEPARTAMENTO antes del disparo de la regla.

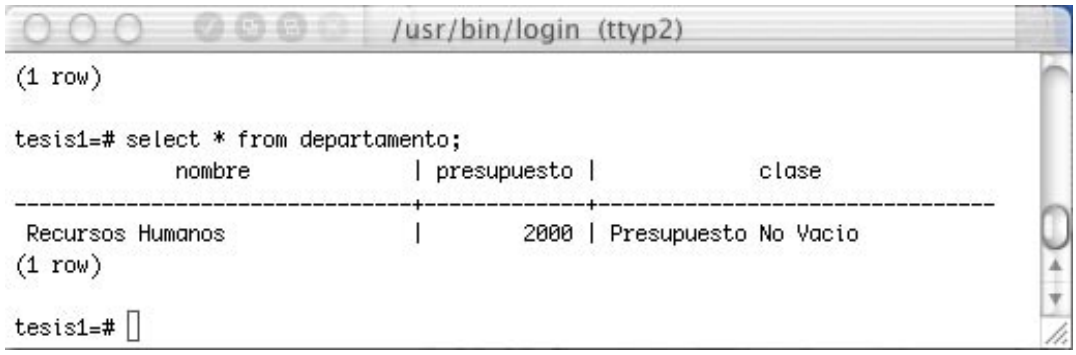

Figura 6.27: Estado final de la tabla DEPARTAMENTO después del disparo de la regla.

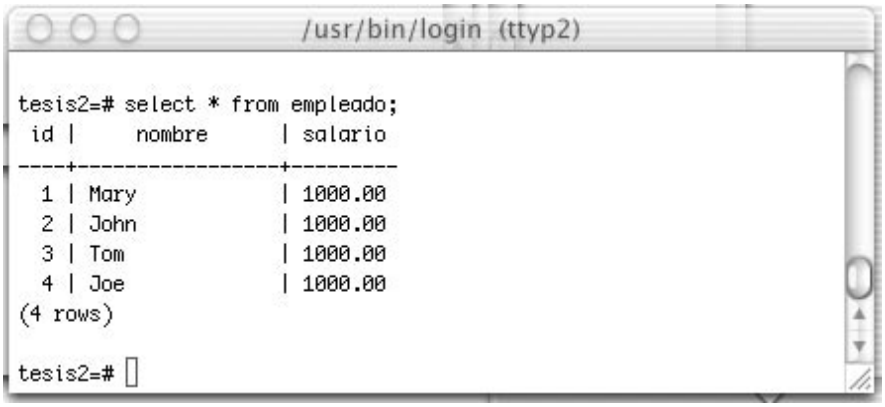

Figura 6.28: Registros almacenados en la tabla EMPLEADO.

La actualización del salario a $5,000.00 para el empleado de nombre 'Mary' (como se realizó en el ejemplo 2 de la sección anterior) modifica el contenido de las tablas como se muestra en la figura 6.29.

El disparo de la regla 1 provoca que el salario del empleado 'John', ahora es el salario de 'Mary' más $1,000.00. Figura 6.30.

El disparo de la regla 2 modifica el salario del empleado 'Tom', ahora es el salario de 'John' más $1,000.00. Figura 6.31.

El disparo de la regla 3, ahora modifica el salario del empleado 'Joe', asignándole $8,000.00 resultado de la suma del salario de Tom más $1,000.00. Figura 6.32.

Ejemplo 3

Considerando que las tabla EMPLEADO, PRIMA y VENTAS almacenan la información mostrada en las figuras 6.33, 6.34 y 6.35.

Cuando se ejecuta la instrucción de la consola de usuario mostrada en la figura 6.14 del ejemplo 3 de la sección anterior, el registro 3 de la tabla PRIMA actualiza el valor del campo "cantidad" a 150. Figura 6.36.

El disparo de la transición de la figura 6.17 produce que el rango del empleado 3 aumente un nivel (figura 6.37).

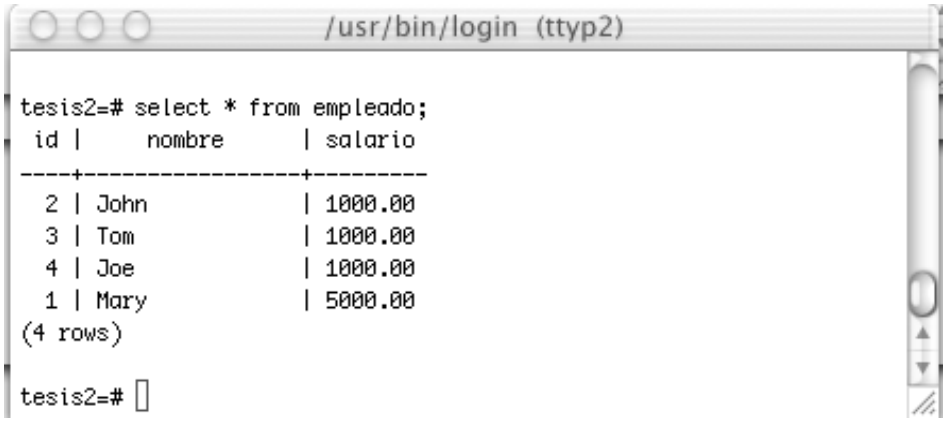

Figura 6.29: Modificación a la tabla EMPLEADO, donde a la empleada 'Mary' se le asignan $5,000.00 de salario.

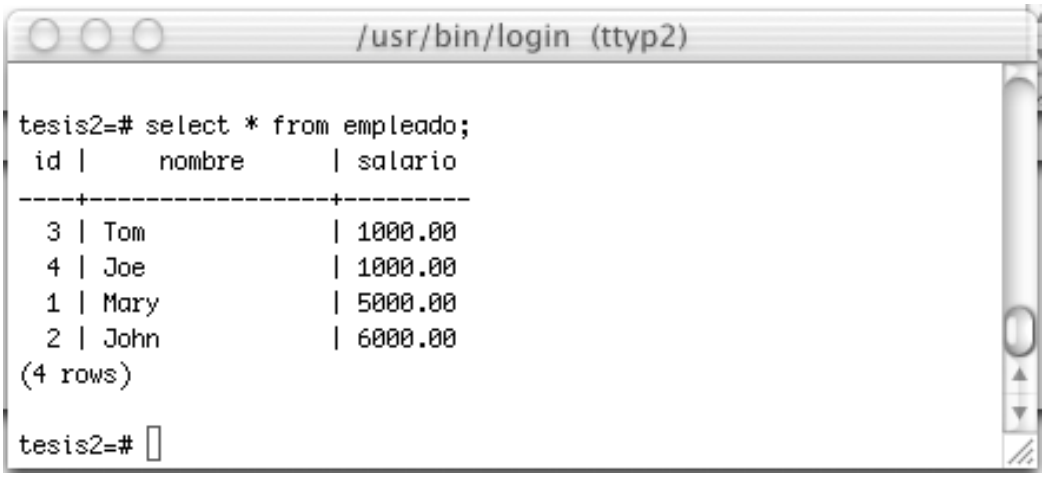

Figura 6.30: Modificación a la tabla EMPLEADO, donde al empleado 'John' se le asignan $6,000.00 de salario.

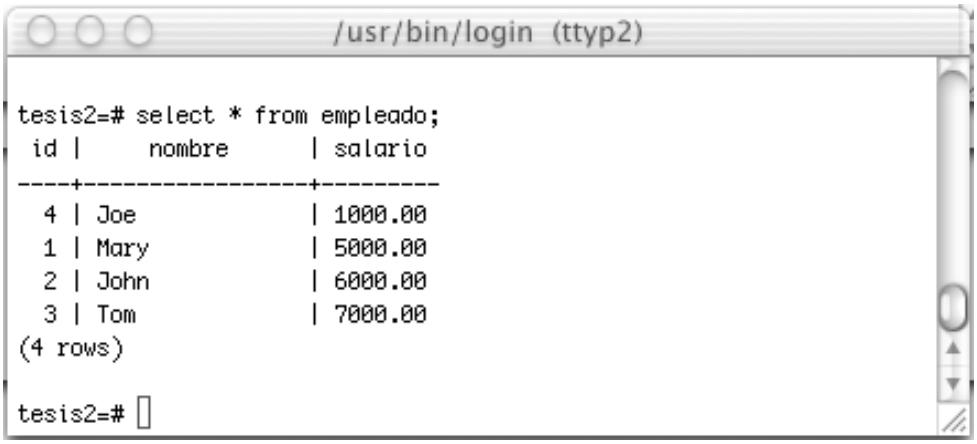

Figura 6.31: Modificación a la tabla EMPLEADO, donde al empleado 'Tom' se le asignan $7,000.00 de salario.

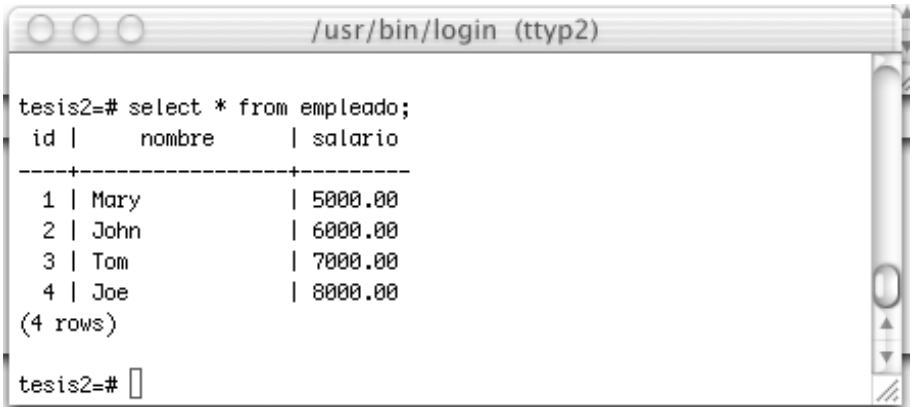

Figura 6.32: Modificación a la tabla EMPLEADO, donde al empleado 'Joe' se le asignan $8,000.00 de salario.

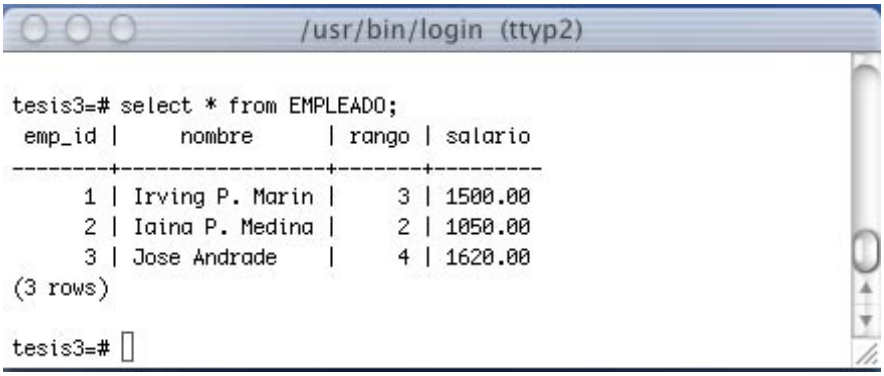

Figura 6.33: Información de la tabla EMPLEADO.

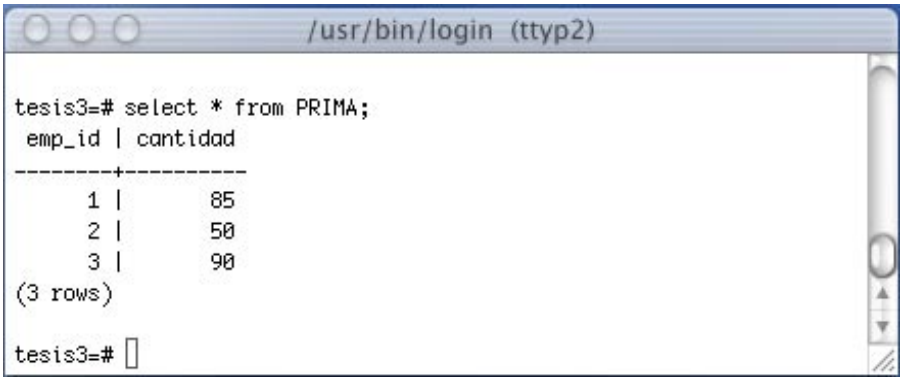

Figura 6.34: Información de la tabla PRIMA.

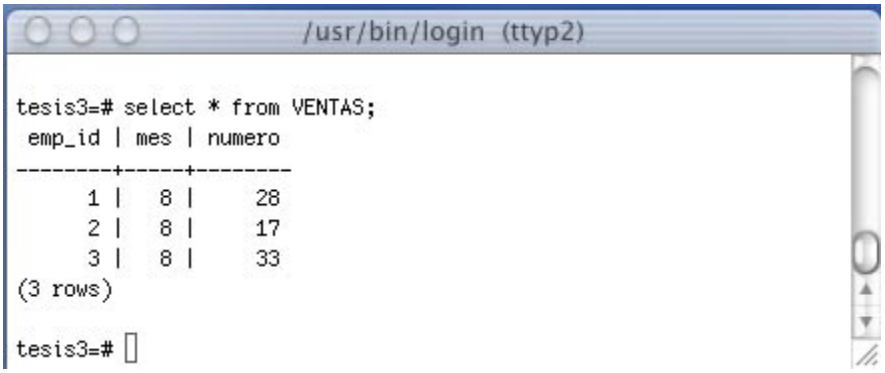

Figura 6.35: Información de la tabla VENTAS.

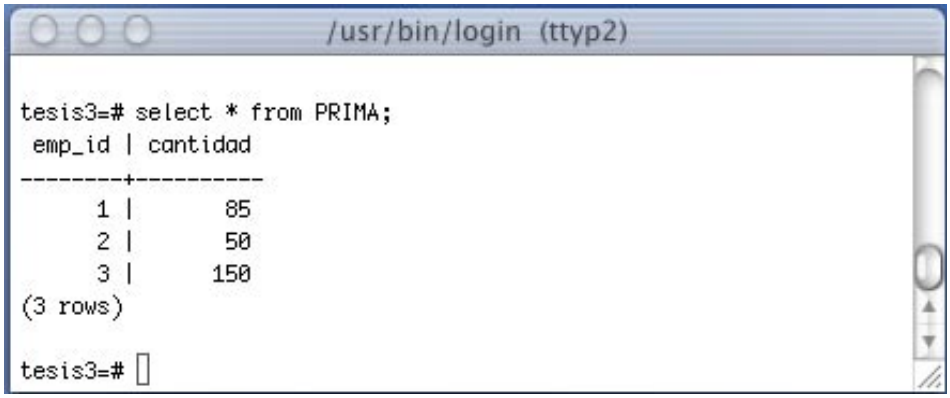

Figura 6.36: Modificación del registro 3 de la tabla PRIMA.

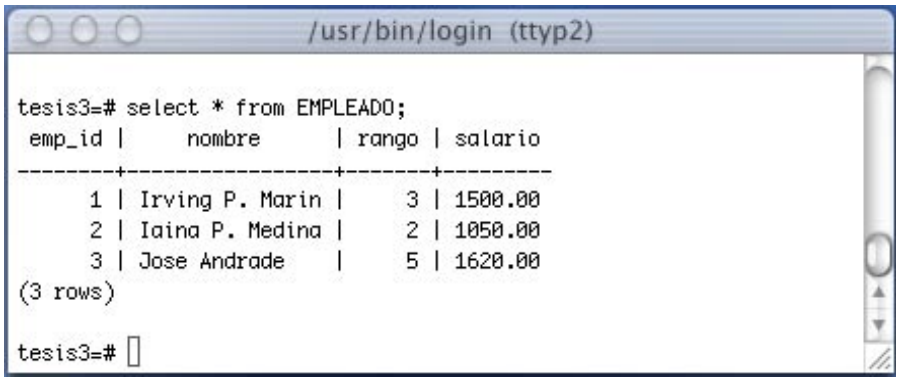

Figura 6.37: El rango del empleado 3 se incrementa en uno.

Continuando con la secuencia de instrucciones del ejemplo 3 de la sección anterior, se agrega un registro a la tabla VENTAS con la siguiente información (1, 9, 55), el resultado se muestra en la figura 6.38.

Este evento provoca el disparo de la regla 3, la cual se dispara al tener un número de ventas mayor que 50, por lo que el salario de éste empleado se actualiza aumentándole $10.00 a su salario actual. Estos cambios se ven reflejados en la figura 6.39.

6.3. Comentarios finales

Para validar un sistema, como el ECAPNSim, es necesario generar un sinnúmero de pruebas de diferentes tipos y así probar su factibilidad. En este reporte de tesis sólo se presentan tres ejemplos de bases de reglas ECA, pero se consideran suficientes para mostrar la aplicación de ECAPNSim en la simulación de reglas ECA y su implementación en ambientes reales.

Las reglas utilizadas en estas pruebas se tomaron de la literatura ([42], [4] y [11]) para evitar el uso de reglas ECA, propuestas por nosotros, que se ajustaran al sistema.

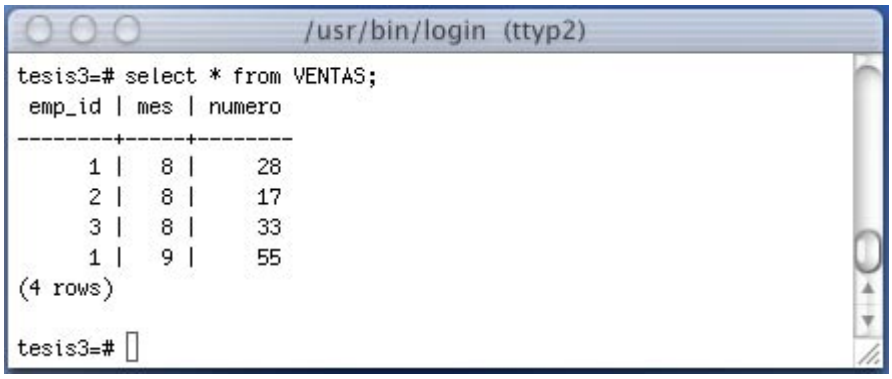

Figura 6.38: Se agrega el registro (1, 9, 55) a la tabla VENTAS.

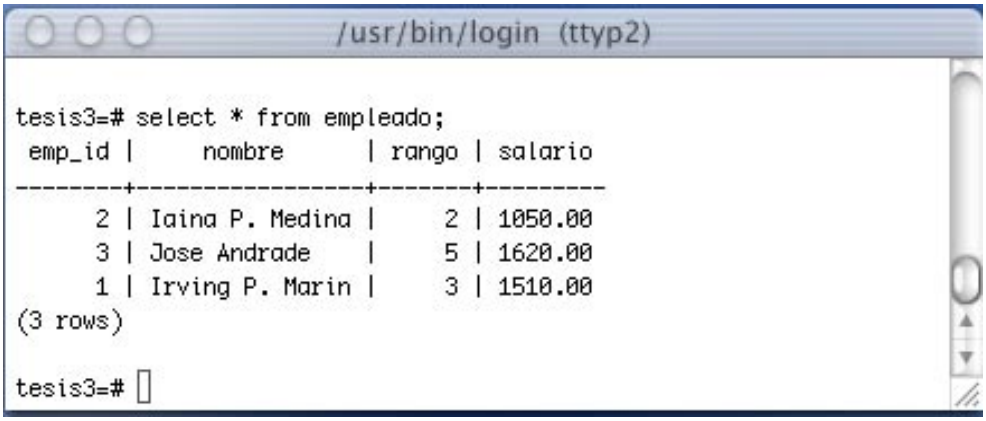

Figura 6.39: Se incrementan $10.00 al salario del empleado 1.

Capítulo 7

Conclusiones

En este capítulo se mencionan los resultados obtenidos durante el desarrollo de este trabajo de tesis, correspondientes a la CCPN y la implementación del ECAPNSim. Además, se describen las actividades a seguir para mejorar estas propuestas. En la sección 7.1 presentamos las aportaciones de la investigación y en la sección 7.2 se enlistan las actividades por realizar para perfeccionar la CCPN y el ECAPNSim.

7.1. Resultados obtenidos

No existe una definición estándar para las reglas de una BD activa, cada ADBMS implementa su propio lenguaje de definición de reglas. En estos lenguajes, es muy difícil observar los efectos que ocasiona el disparo de una regla. Existen mecanismos para analizar el comportamiento de una base de reglas activa, sin embargo, estos procedimientos se tienen que realizar en otro medio, separada de la definición de las reglas. En este trabajo de tesis se está proponiendo un modelo que combina la definición de reglas ECA con la facilidad de realizar la simulación de las mismas, aprovechando las propiedades que da la teoría de PN para la simulación de sistemas manejados por eventos.

El modelo original de PN no es suficiente para cubrir las características propias de una regla ECA, siendo necesaria la modificación de los elementos de la PN para almacenar información que nos permitiera manipular adecuadamente a las reglas ECA. El modelo propuesto es una PN extendida a la que denominamos Red de Petri Coloreada Condicional (CCPN, Conditional Colored

Petri Net), la cual contiene características del modelo original, así como de otras extensiones de PN.

Para probar la factibilidad del modelo CCPN, se diseñó el ECAPNSim, el cual es una herramienta gráfica para el diseño de PN, contando además con la propiedad de convertir una base de reglas activas en su correspondiente estructura de CCPN. El ECAPNSim puede trabajar en dos modalidades: i) Llevar a cabo la simulación recibiendo la información de los tokens por medio de una consola que funciona como cliente; y ii) Conectar al ECAPNSim con una BD, ejecutando las acciones de las reglas y las instrucciones enviadas por la consola sobre la BD.

Sin embargo, la limitante que tiene ECAPNSim es que en la modalidad de funcionamiento "Real" no se aceptan todo tipo de reglas ECA. Solamente se aceptan aquellas reglas donde el evento contiene toda la información necesaria para verificar la condición y, en su caso, ejecutar la acción.

No existe en la literatura un modelo que ofrezca un medio de análisis y aplicación de reglas ECA, lo cual, hace a la CCPN y al ECAPNSim únicos en su género.

7.2. Trabajo futuro

Como trabajo futuro pretendemos lo siguiente:

- Mejorar ECAPNSim para soportar toda clase de reglas ECA.
- Realizar análisis de terminación y confluencia en el modelo CCPN, aprovechando las propiedades de PN.
- Considerar la definición de diferentes tipos de eventos compuestos para las reglas ECA.
- Implementar el uso de los diferentes conectivos lógicos en la definición de la parte condicional de la regla ECA.
- Montar ECAPNSim sobre una BD real y proporcionarle un comportamiento activo para cualquier definición de reglas ECA.

Bibliografía

[1] Date C.J., "*An Introduction to Database Systems"*, Addison-Wesley, Sixth Edition, System Programming Series, Reading, (MA) 1995.

[2] Paton N.W., Diaz O., "Active Database Systems", *ACM Computing Surveys*, Vol. 31, No. 1,1999, pp. 64-103

[3] 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.

[4] Chang Y.I, and Chen F.L., "RBE: A Rule-by-example Active Database System", *Software - Practice and Experience*, vol. 27, 365-394, April 1997.

[5] Widom J., "*The Starburst Active Database Rule System*", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 4, August 1996.

[6] www.ca.postgresql.org

[7] www-db.research.bell-labs.com/project/ode/index.html

[8] 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.

[9] Y.W. Stanley, Jawardi R., Cherukuri P.,Li Q., and Nartey R., "OSAM* KBMS/P: A Aparallel, Active, Object-Oriented Knowledge Base Server", *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, No. 1, January/February 1998, pp. 55-75.

[10] 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

[11] 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

[12] 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

[13] 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

[14] www.daimi.au.dk/PetriNets/tools/quick.html

[15] Meneses Viveros A., "PetrA: Herramienta para la modelación y simulación de redes de Petri", Tesis de Maestría presentada en la sección de Computación dle Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del IPN, México. Febrero de 2002.

[16] Silberschatz A., Korth H.F., Sudarshan S., "*Database System Concepts*", Third Edition, McGraw-Hill, 1999.

[17] Dittrich K., Gatziu S., Geppert A., "*The Active Database Management System Manifesto: A Rulebase of ADBMS Features*". A Join Report by the ACT-NET Consortium.

[18] Peterson J.L., *"Petri Net Theory and The Modeling of Systems"*, Prentice-Hall, Inc., 1981. ISBN 0-13-661983-5.

[19] Murata T., *"Petri Nets: Properties, analysis, and applications"*, Proceedings of the IEEE, 77(4):541-580, 1989.

[20] 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.

[21] Wang J., *"Timed Petri Nets. Theory and Applications"*, Kluwer Academic Publishers, 1998.

[22] Champagat R., Esteban P., Pingaud H., ValetteR., "*Petri net based modeling of hybrid systems*", Computers in Industry 1998, No. 36, pp. 139-146.

[23] Chen H. and Hanisch H., "Analysis of hybrid systems on hybrid net condition/event system model", *Disrete Event Systems: Theory and Applications*, vol.11, 163-185, 2001

[24] 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.

[25] 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

[26] 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.

[27] 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.

[28] 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. Jair.

[29] 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

[30] 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.

[31] 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.

[32] 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.

[33] Kulkarni K., Mattos N. y Cochrane R., *"Active database features in SQL3"*. In Active Database Systems, N. Paton, Ed. Springer Verlag, Berlin, Germany, 1998.

[34] 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.

[35] 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.

[36] 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.

[37] Eckel B., Thinking in Java, 2nd. Edition, Prentice Hall, June 2000.

[38] 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

[39] Medina Marín J. y Li X., "Red de Petri Coloreada Condicional y su Aplicación en Sistemas de Bases de Datos Activas", *Octava Conferencia de Ingeniería Eléctrica 2002 (CIE'02)*, México, D.F., México, Septiembre 4, 5 y 6, 2002.

[40] 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.

[41] Zimmer D., Unland R., Meckenstock A., *"Rule Termination Analysis based on Petri Nets"*, Cadlab Report 3, Cadlab, Fustenallee 11, 33102 Paderborn, Germany, February 1996.

[42] Fraternali P., Teniente E., and Urpí T., "Validating Active Rules by Planning", *the 3rd International workshop on Rules in Database (RIDS'97) (or LNCS 1312)*, Skövde, Sweden, June, pp. 181-196, 1997.