PetrA: Herramienta Orientada a Objetos para la Modelación, Simulación y Verificación de Redes de Petri.

Sergio Chapa Vergara Amilcar Meneses Viveros Hugo García Monroy*
Sección de Computación, Depto. de Ingenieria Electrica
CINVESTAV

*Depto. Aplicación de Microcomputadoras
Instituto de Ciencias
Universidad Autónoma de Puebla.

Resumen

Cuando se desarrollan aplicaciones distribuidas existen dependencias críticas que pueden ser difíciles de encontrar entre los distintos módulos o componentes de la aplicación, lo cual provoca que la confiabilidad del software desarrollado no sea adecuada. Por esta razón requerimos de un mecanismo que nos ayude a proporcionar una buena confiabilidad a este tipo de sistemas, esta confiablilidad la obtenemos a través de distintas técnicas durante el desarrollo de software. Debido a que las redes de Petri son una herramienta muy versátil para este propósito, presentamos los avances del desarrollo de la aplicación PetrA, aplicación orientada a objetos que se utiliza como plataforma para trabajar con diferentes tipos de redes de Petri.

1 Introducción

El desarrollo de los sistemas distribuidos se ha incrementado en los últimos años, debido al grado de confiabilidad y popularidad que han alcanzado las redes. Muchos sistemas se han desarrollado en este tipo de plataformas y, del mismo modo, muchos sistemas que se habían desarrollado en computadoras *mainframe* se han emigrado a plataformas distribuidas. Esta migración se ha realizado a través de simuladores o de la reescritura del código del sistema[14]. Ejemplos de estos sistemas distribuidos son manejadores de bases de datos y simuladores de computadoras paralelas.

Sin embargo, el grado de complejidad que se utiliza en el desarrollo y mantenimiento de estos sistemas es mayor que el utilizado en las aplicaciones no distribuidas. En general se han desarrollado diferentes metodologías para que el diseño y mantenimiento de sistemas sea más fácil. La metodología que parece ser la más exitosa para este propósito es la orientada a objetos[7][9]. Esta facilidad de diseño y mantenimiento ayuda a obtener una confiabilidad adecuada del sistema en desarrollo. Sin embargo, muchas veces se requiere que los sistemas distribuidos trabajen en plataformas abiertas —esto es, que tengan la capacidad de trabajar en diferentes plataformas—, por esta razón existen algunas dependencias críticas, complejas y ocultas entre los diferentes elementos del sistema, lo que dificulta que éste tenga una buena confiabilidad. Es decir, el análisis y diseño de sistemas requiere que los sistemas que están en desarrollo, y los ya desarrollados se comporten de manera confiable[9]. Entendamos como confiabilidad a la *"habilidad del sistema para cumplir su tarea predefinida (a pesar de las fallas de hardware y software)"*[9].

Existen factores que deterioran la confiabilidad de un sistema como son: fallas, errores, y descuidos (a nivel hardware, sistema operativo, e interacción con otros sistemas, entre otros), por mencionar algunos.

Para saber si un sistema es adecuadamente confiable nos apoyamos en algunas medidas que nos ayudan a evaluar su grado de confiabilidad. Estas medidas pueden ser integridad, disponibilidad, seguridad y protección. La confiabilidad la podemos obtener pronosticándola o procurándola. Existen varios métodos para procurar una buena confiabilidad, como son las tecnologías de desarrollo de software, métodos de validación asistidos por computadora (para detectar y remover fallas), y manejo de excepciones, entre otras.

Los métodos de validación asistidas por computadora cubren el análisis del flujo de datos, análisis del flujo de control, simulación funcional (prototipo), evaluación analítica, y evaluación simulativa, entre otras.

Sin embargo, los métodos de validación no son completos, es decir, un método no puede garantizar todas las medidas de confiabilidad, pero se complementan unos a otros. Una herramienta que se utiliza para la validación son las redes de Petri, debido a que son una representación adecuada para la especificación y programación de diferentes lenguajes, se utilizan en diferentes fases en el ciclo de desarrollo de los sistemas de software, y se aplica en distintos métodos de validación. Además de proporcionar un gran poder de modelación para sistemas que tengan aspectos de concurrencia y paralelismo, y se aplica en diferentes niveles de abstracción[1][2].

Las redes de Petri tienen ventaja sobre otros métodos de validación —como el uso de lenguajes como UML, diagramas E-R, o autómatas finitos— en los cuales, además de permitir una visualización del comportamiento dinámico y estático del sistema, se utiliza, como ya se mensionó anteriormente, en distintas fases del ciclo de software, lo cual lo hace una herramienta muy completa para validación.

2 Redes de Petri

Las redes de Petri son una herramienta gráfica y matemática de modelación que se puede aplicar en muchos sistemas. Particularmente son ideales para describir y estudiar sistemas que procesan información y que tienen características concurrentes, asíncronas, distribuidas, paralelas, no determinísticas y/o estocásticas.

Las redes de Petri pueden incorporarse informalmente en cualquier área o sistema que pueda describirse gráficamente como diagramas de flujo y que necesitan de algunos medios para representar actividades paralelas o concurrentes. Las redes de Petri tienen pueden aplicarse como herramienta de validación (en diferentes métodos de validación) en distintas fases del desarrollo de un sistema. Sin embargo, el punto débil de las redes de Petri radica en la complejidad del problema, esto es, los modelos basados en redes de Petri, siempre tienden a ser muy grandes para su análisis, para solucionar este problema se han desarrollado técnicas de reducción y extensiones a las redes de Petri. Por lo general, para aplicar las redes de Petri a un problema, se le hacen modificaciones o extensiones.

2.1 Definiciones formales

Definición 1. Una red de Petri es un tipo particular de grafo dirigido que consiste de dos tipos de nodos (lugares y transiciones). Una red de Petri es una estructura algebráica[1] $PN = (P, T, I, O)$ donde:

- $P = p_1, p_2, \ldots, p_m$ es el conjunto de lugares.
- $T = t_1, t_2, \ldots, t_n$ es el conjunto de transiciones.
- $I : P \times T \to N$ es la función de entrada en la cual se especifican los lugares de entrada de la transición.
- $O : P \times T \to N$ es la función de salida en la cual se especifican los lugares de salida de la transición.

N es el conjunto de números naturales y además los conjuntos P y T cumplen con $P \cap T = \emptyset$.

En la representación gráfica, la red de Petri se dibuja como un grafo con dos tipos de nodos: los lugares se dibujan como círculos y las transiciones como barras o cajas. Un arco dirijido de una lugar p a una transición t define una entrada de dicha transición. Un arco dirijido de una transición t a un lugar p define la salida de la transición. En ocasiones es necesario colocar valores de peso a los arcos y se denota por $w(p,t)$, donde w es la función

$$w : (P \times T) \cup (T \times P) \to N.$$

Cuando un arco no tiene señalado su valor de peso, por omisión su valor es 1.

Definición 2. Una *marca* m de una red de Petri es una función $m : P \to N$, la cual asigna a cada lugar $p \in P$ un número de *tokens*. La presencia o ausencia de *tokens* indica el estado de un lugar, y la marca de lugares representa la disponibilidad de un recurso, o la ocurrencia de operaciones.

[1] Existen divesas notaciones de redes de Petri, nosotros preferimos utilizar la que aparece en[2].

La marca asigna a cada lugar un número entero no negativo. Gráficamente colocamos k puntos en un lugar p, si éste tiene asociado k tokens. Una marca se denota por M, un m vector, donde m es el número total de lugares. El p-éismo componente de M, denotado por $M(p)$, es el número de tokens en el lugar p.

Cuando se modela, se utilizan los conceptos de condición y evento, así, en las redes de Petri las condiciones se representan como lugares y los eventos como transiciones. Una transición (evento) tiene un cierto número de lugares de entrada y salida, las cuales representan las precondiciones y las postcondiciones del evento respectivamente.

El comportamiento de muchos sistemas se puede describir en términos de los estados del sistema y de sus cambios. En las redes de Petri, para simular el comportamiento dinámico de un sistema, un estado o *marca* de la red cambia de acuerdo con las siguientes reglas de transición:

1. Se dice que una transición t está habilitada si cada lugar p de entrada de t tiene al menos $w(p,t)$ *tokens*, donde $w(p,t)$. es el peso del arco de p a t.
2. Una transición habilitada puede o no dispararse (dependiendo en que evento tome o no el lugar).
3. Un disparo de una transición habilitada t remueve $w(p,t)$ *tokens* de cada lugar de entrada p de t, y agrega un $w(t,p)$ *tokens* por cada lugar de salida p de t, donde $w(t,p)$ es el peso del arco de t a p.

Las transiciones que no tienen lugares de entrada se les llama *transiciones fuente*. Una transición fuente siempre está habilitada. Por otro lado una transición sin lugares de salida consume tokens, pero no los produce.

Se denomina un ciclo a sí mismo, a un lugar p y una transición t, si cumplen que $p \in O(t)$ y $p \in I(t)$, es decir, p es un lugar de entrada y un lugar de salida de la transición t.

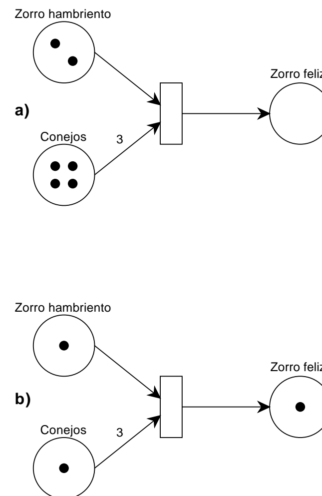

Figura 1: Ejemplo 1: Ilustración de una regla de transición.

Un ejemplo de una transición se muestra en la figura 1. En esta figura se muestra el resultado parcial del com-

portamiento poblacional de zorros y conejos, donde cada zorro hambriento debe comer 3 conejos para satisfacer su apetito. Podemos observar que la transición t tiene 2 lugares de entrada —1 para cada zorro ambriento y otra para los conejos—, y uno de salida (¡para el zorro feliz!). En la figura 1a, se muestra que hay dos zorros y cuatro conejos disponibles, por lo que la transición t se habilita. Después de que t se dispara, la marca se cambia al que se muestra en la figura 1b, y la transición t ya no está habilitada.

Observemos que las transiciones asumen que cada lugar puede almacenar un número ilimitado de tokens, es decir, se asume que cada red de Petri es una red de capacidad infinita. Sin embargo, en muchos casos de modelación, es preferible contar con una cota superior para el número de *tokens* que se almacenará en cada lugar. A estas redes se les considera como redes de capacidad finita y se denotan como (PN, M_0), donde M_0 representa la marca inicial, y cada lugar de p de la red tiene asociado una capacidad máxima $k(p)$ de tokens. Para que una transición t de una red de capacidad finita se habilite se agrega la *regla estricta de transición*[2], en la cual de debe cumplir que cada lugar p de salida, no exceda su capacidad $k(p)$ después de disparar t.

2.2 Aspectos de modelación con redes de Petri

Las redes de Petri se prestan para la modelación si la estructura de causa-evento[3] de un sistema se conoce y se utiliza para definir el modelo. Los lugares representan causas o condiciones, y las transiciones eventos. Todos los eventos modelados por las redes de Petri deben ser discretos en tiempo. Además los eventos deben generarse por condiciones en una parte local del estado actual del modelo.

Una red de Petri tiene una estructura gráfica que tiene lugares, transiciones, arcos y tokens. Los lugares son elementos pasivos de la red de Petri y junto con los tokens, se utilizan para modelar los estados del sistema. Las transiciones son elementos activos de la red que representan las acciones de un sistema, estas acciones originan cambios en el estado de la red. El conjunto de lugares, transiciones y arcos son finitos y estáticos, mientras que el conjunto de tokens y marcas pueden cambiar durante la ejecución de la red.

La propiedad de *valor de peso* de los arcos, hace posible que se especifiquen el número de tokens que consume la transición de los lugares de entrada y el conjunto de tokens que produce en la salida. Las redes de Petri de capacidad finita y peso en los arcos se les llama *sistemas de lugar/transición*.

Las clases originales de redes de Petri, y los sistemas de lugar/transición son muy conocidos por su uso en modelos de un alto grado de abstracción que tienen que analizarse de manera formal. Pero si el modelo debe respetar mas detalles del sistema, o si se debe respetar el tiempo en el modelo, entonces se deben desarrollar más clases de redes de Petri que consideran los aspectos deseados del modelo. Así, surgen las redes de Petri coloreadas, estocásticas, de tiempo, y orientadas a objetos, por mensionar algunas, que en general forman el grupo de redes de Petri extendidas.

Ejemplos de modelación con redes de Petri son el flujo de información y los protocolos de comunicación. Se muestran ejemplos de estos modelos.

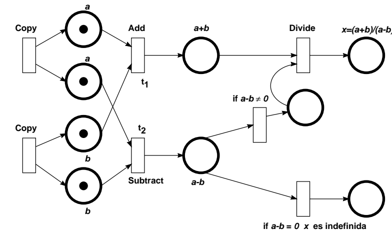

Figura 2: Se muestra el flujo de datos de la computación para $x = \frac{(a+b)}{(a-b)}$.

Ejemplo de modelación de flujo de datos

La red de Petri que se muestra en la figura 2. Se realiza una computación de flujo de datos cuando las instrucciones se habilitan para ejecutarse cuando llegan sus operandos, y se pueden ejecutar concurrentemente. En la representación de redes de Petri de la computación de flujo de datos, los tokens denotan los valores actuales de los datos, así como la disponibilidad del dato. En la red que se muestra en la figura 2, las instrucciones están representadas por las transiciones t_1 y t_2, y se pueden ejecutar concurrentemente y depositar sus resultados —$(a+b)$ o $(a-b)$— en sus respectivos lugares de salida.

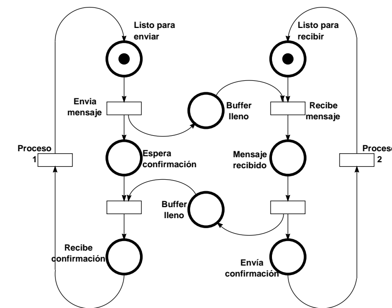

Figura 3: Modelo simplificado de un protocolo de comunicación.

Ejemplo de modelación de protocolos de comunicación.

Los protocolos de comunicación son otra área donde las redes de Petri se pueden utilizar para representar algunas características específicas y esenciales. Frecuentemente las propiedades de las redes de Petri como vida y seguridad se utilizan para como criterios de validación en los protocolos de comunicación. La red de Petri que se muestra en la figura 3 es un ejemplo muy sencillo de un protocolo de comunicación entre dos procesos. La figura 4 muestra la representación de una espera de proceso no

[2]A la regla que no considera capacidades obligatorias se le denomina *regla débil de transición*

[3]tambien se le conoce como condición evento

determinista donde t_{r1}, t_{r2} o t_{out} se disparan si se recibe la respuesta 1,2, o no hat respuesta en un tiempo especificado (t_{out}).

2.3 Propiedades de comportamiento

El principal *strength* de las redes de Petri, son su soporte para el análisis de propiedades y problemas asociados con sistemas concurrentes. Dos propiedades se pueden estudiar con un modelo de red de Petri:

- Propiedades de comportamiento. Estas propiedades dependen de la marca inicial M_0, estas propiedades son alcanzabilidad, acotamiento, vida, reversibilidad y estado inicial, persistencia, distancia sincrónica.
- Propiedades estructurales. Estas propiedades son independientes de la marca inicial M_0.

Figura 4: Representación de una espera no determinista en un proceso.

Alcanzabilidad

La alcanzabilidad es una base fundamental para estudiar las propiedades dinámicas de cualquier sistema. El disparo de una transición habilitada cambiará la distribución de los tokens en una red, de acuerdo a las reglas de transición mensionadas en la definicion 3. Una secuencia de disparos dará como resultado una secuencia de marcas. Se dice que una marca M_n es alcanzable de una marca M_0 si existe una secuencia de disparos que transformen a M_0 en M_n. Un secuencia de disparos se denota por $\sigma = M_0 \quad t_1 \quad M_1 \quad t_2 \quad M_2 \quad \cdots \quad t_n \quad M_n$ o simplemente por $\sigma = t_1 \quad t_2 \quad \cdots \quad t_n$ En este caso M_n es alcanzable por M_0 y se denota como $M_0[\sigma > M_n$. El conjunto de todas las marcas posibles alcanzables por M_0 en una red (N, M_0) se denota como $R(N, M_0)$. El conjunto de todas las secuencias de disparo desde M_0 en una red (N, M_0) se denota como $L(N, M_0)$.

El problema de alzanzabilidad en las redes de Petri es el problema de encontrar una $M_n \in R(N, M_0)$ deseada.

Acotamiento

Se dice que una red de Petri (N, M_0) es k-acotada o simplementa acotada si el número de tokens en cada lugar de la red no excede a un número finito k en cada marca alcanzable desde M_0. Se dice que una red de Petri es *segura* si es 1-acotada.

Activa

El concepto de activa está muy relacionado con la ausencia completa de candados mortales en sistemas opertaivos. Se dice que una red de Petri esta activa si, no importando que marca se alcance desde M_0, si aún se puede realizar una secuencia de dis

paro. Esto significa que una red de Petri activa garantiza la ausencia total de candados mortales, no importando la secuencia de disparos que se seleccione.

La propiedad activa es el ideal de muchos sistemas. Sin embargo, es impractico y costozo verificar esta propiedad en sistemas muy grandes como lo son los sistemas operativos. Así, la condicion de vida se define en diferentes niveles. Se dice que una transición t en una red de Petri (N, M_0) es:

1. *Muerta* ($L0$-viva) si t nunca se dispara en ninguna secuencia en $L(N, M_0)$.
2. $L1$-activa (*potencialmente disparable*) si t puede dispararse al menos una vez en alguna secuencia en $L(N, M_0)$.
3. $L2$-activa si dado cualquier entero positivo k, t puede dispararse al menos k en alguna secuencia en $L(N, M_0)$.
4. $L3$-activa si t aparece infinitamente, frecuentemente en alguna secuencia en $L(N, M_0)$.
5. $L4$-activa o activa si t es $L1$-activa para cualquier marca en $R(N, M_0)$.

Se dice que una red de Petri (N, M_0) es Lk-activa si cualquier transicción en la red es Lk-activa (con $k = 0, 1, 2, 3, 4$).

Reversibilidad y estado Inicial

Se dice que una red de Petri es reversible si M_0 es alcanzable para cualquier marca $M \in R(N, M_0)$. Así, una red reversible regresará a su estado inicial.

Cubrir

Se dice que una marca M en una red de Petri (N, M_0) se puede cubrir si existe una marca $M' \in R(N, M_0)$ tal que $M'(p) \geq M(p)$ para cada p en la red. La propiedad de cubrir está muy relacionada con la propiedad $L1$-vivo de las transiciones. Sea M la marca mínima para habilitar la transición t. Entonces t esta muerto si y solo si M no se puede cubrir. Esto es, t es $L1$-vivo si y solo si M se puede cubrir.

Persistencia

Se dice que una red de Petri (N, M_0) es persistente si, para cualesquiera dos transiciones habilitadas, el disparo de una transición no deshabilita a la otra. Una transición en una red persistente, estará habilitada hasta que se dispare. Esta noción de persistencia es muy utilizada en el contexto de esquemas de programas paralelos y circuitos asíncronos altamente independientes.

Distancia sincrónica

La noción de distancia sincrónica es un concepto fundamental introducido por C.A. Petri. Es una métrica muy relacionada al grado de dependencia mutua entre dos eventos en un sistema de condición/evento. Definiremos

la distancia sincrónica entre dos transiciones t_1 y t_2 en una red de Petri (N, M_0) por

$$d_{12} = \max |\overline{\sigma}(t_1) - \overline{\sigma}(t_2)|$$

donde σ es una secuencia que inicia en cualquier marca M en $R(N, M_0)$ y $\overline{\sigma}(t_i)$ es el número de veces que la transición t_i (con $i = 1, 2$) dispara en σ.

Equidad

Se han propuesto distintas nociones de imparcialidad en la literatura de redes de Petri. Presentamos dos conceptos básicos de equidad:

Equidad Acotada. Se dice que dos transiciones t_1 y t_2 estan en una relación de equidad acotada si el número máximo de veces en que una dispara mientras la otra no está acotada. Se dice que una red de Petri (N, M_0) es de equidad acotada si cada par de transiciones en la red esta en una relación de equidad acotada.

Equidad incondicional o global . Se dice que una secuencia σ es de equidad incondicional si es finita o si cada transición en la red aparece infinitamente en σ. Se dice que una red (N, M_0) es de equidad incondicional si cada secuencia de disparo σ de $M \in R(N, M_0)$ es de equidad incondicional

3 Métodos de análisis

Los métodos de análisis para las redes de Petri se pueden clasificar en tres grupos generales:

1. Método de arbol de alcanzabilidad o cubrimiento.
2. Enfoque de matriz de ecuaciones.
3. Técnicas de reducción o descomposición.

3.1 Arbol de alcanzabilidad

Dada una red de Petri (N, M_0) desde una marca inicial M_0 podemos obtener tantas nuevas marcas como transiciones habilitadas. Así, de cada nueva marca podemos obtener más marcas. Este proceso genera un árbol de marcas. Los nodos representan las marcas generadas a partir de M_0 (la raíz) y sus suscesores, y cada arco representa un disparo de una transición, la cual transforma una marca en otra.

Algunas de las propiedades que se pueden estudiar utilizando el árbol de alcanzabilidad T para una red de Petri (N, M_0) son las siguientes:

1. Una red (N, M_0) es acotada y así $R(N, M_0)$ es finito si y solo si ω no aparece en nigún nodo etiquetado en T.
2. Una red (N, M_0) es libre si y solo si solo aparecen ceros y unos en las etiquetas de los nodos de T.
3. Una transición t es muerta si y solo si no aparece como etiqueta de un arco en T.
4. Si M es alcanzable desde M_0, entonces existe un nodo etiquetado M' tal que $M \leq M'$.

3.2 Matriz de incidencia y ecuación de estado

Este enfoque es muy apropiado cuando se puede describir y analizar completamente el comportamiento dinámico de las redes de Petri por medio de ecuaciones. Sin embargo, la solución de las ecuaciones involucradas en este comportamiento es limitado, debido a la naturaleza no deterministica en los modelos de redes de Petri, y porque las soluciones pueden contener enteros negativos.

Matriz de inicidencia Para una red de Petri pura[4] N con n transiciones y m lugares, la matriz de incidencia $A = [a_{ij}]$ es una matriz $m \times n$ de enteros y su entrada esta definida por:

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

donde $a_{ij}^{+} = w(i, j)$ es el peso del arco de la transición i a su lugar de salida j, y $a_{ij}^{-} = w(i, j)$ es el peso del arco de la transición j a su lugar de entrada i. Esto es, la matriz de incidencia representa los lugares de entrada con valores negativos, y los valores positivos representan salidas. Además una tansición j se encuentra habilitada en una marca M si

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

Ecuación de estado La matriz de estado de una red de Petri se escribe como:

$$M_k = M_{k-1} + A^T u_{k'}$$

Donde M_k es el vector columna de $m \times 1$ de que presenta el estado actual k. M_{k-1} es el vector columna de $m \times 1$ que representa el estado anterior $k-1$. A^T es la matriz transpuesta de la matriz de incidencia de la red de Petri. Y $u_{k'}$ es el vector columna con $n-1$ ceros y un 1 en la j-ésima posicón, indicando la transiciónn que hbrá de dispararse.

4 Avances del diseño e implementacón de PetrA

La aplicación PetrA debe cumplir las siguientes propiedades:

- Capacidad de trabajar con diferentes documentos en forma simultánea.
- Capacidad de almacenar la red de Petri como un documento.
- Contar con un editor gráfico para el diseño de la red de Petri.
- Ejecutar la simulación de la red de Petri sobre el diseño gráfico correspondiente.
- Tener la capacidad de realizar análisis con enfoque a la matriz de ecuaciones.
- Contar con medios para ayudar a desarrollar las técnicas de reducción o descomposición.

[4] Una red de Petri pura es aquella red que no tiene lugares que son entrada y salida de la misma transición.

- Tener la capacidad de poder los diferentes tipos de redes de Petri que existen (por ejemplo redes coloreadas u orientadas a objetos).
- Capacidad de asignar una distribución topológica a la red de Petri.

4.1 Estrategia

La estrategia que seguimos es la siguiente: hemos separado el problema en tres estapas. En la primera etapa se busca tener un sistema de modelación y simulación sencillo, donde las principales operaciones sean editar el modelo de la red de Petri y mostrar su ejecución. En la segunda etapa, se le incorporará técnica de análisis con en enfoque matricial y, en la tercera etapa se le incorporarán las técnicas de reducción a la red de Petri, agregándole, la capacidad de manejar subredes.

4.2 La plataforma de desarrollo

Se ha seleccionado como plataforma de desarrollo a OpenStep por varios motivos, entre los que destacan la elegancia del desarrollo y manejo de aplicaciones orientadas a objetos, la capacidad de utilizar múltiples hilos de control en una aplicación, y la factibilidad de exportar de manera casi transparente a GNUStep y MacOs X, y de portarlo a Windows (compilando la aplicación con el producto Yellow Box disponible en este departamento).

4.3 Consideraciones

Una red de Petri es un grafo dirigido con dos tipos de nodos —lugares y transiciones—, los arcos dirijidos tienen un peso asociado y los tokens se almacenan en los lugares.

Los lugares transiciones y arcos determinan las características estáticas de la red de Petri, y las marcas, o el número de tokens almacenado en cada lugar, determinan las características dinámicas del sistema. Además se debe tener en consideración que las transiciones actuarán dependiendo de si están o nó habilitadas, tomando en cuenta que el disparo de una transición puede deshabilitar y/o habilitar otras transiciones.

4.4 Elementos principales de una red de Petri

Con las consideraciones pertinentes podemos iniciar la discución del diseño de la aplicación PetrA.

Los elementos principales que podemos distinguir en el problema que deseamos atacar, y que se representarán en clases, son los siguientes:

Red de Petri Los objeto de esta clase tendrán que representar gráficamente a la red de Petri, además de tener métodos para realizar su ejecución y su analísis.

Lugar Los objetos de esta clase deben encargarse de representar a los nodos lugares de la gráfica de red de Petri. Entre sus atributos destacan el número de tokens que almacena.

Transición Los objetos de esta clase deben representar las características gráficas y de comportamiento de este tipo de nodo. Estos objetos deben conocer sus lugares de entrada y salida. Estos objetos deben de tener una unidad de ejecución para realizar los cambios de marcas de la red.

Arco Los objetos de esta clase deben representar gráficamente el arco dirigido entre una transición y un lugar, o entre un lugar y una transición. Este objeto tiene asociado el atributo de peso y debe tener la propiedad de tener algoritmos para dibujar, no sólo con líneas rectas los arcos entre dos nodos.

Token Los objetos de esta clase representan a los tokens. Para una red sencilla no tendrá relevancia, inclusive podría omitirse, pero se considera por si se desea extender para redes de Petri coloreadas. Por lo que su atributo será el color.

4.5 La estructura de la aplicación

En la sección anterior se ha hecho la descripción de los objetos principales de una red de Petri, sin embargo estos objetos no bastan para desarrollar una aplicación, pues hay que tomar en consideración las interacciones con el ambiente de trabajo. La figura 6 muestra la estructura de la aplicación PetrA. En esta estructura se

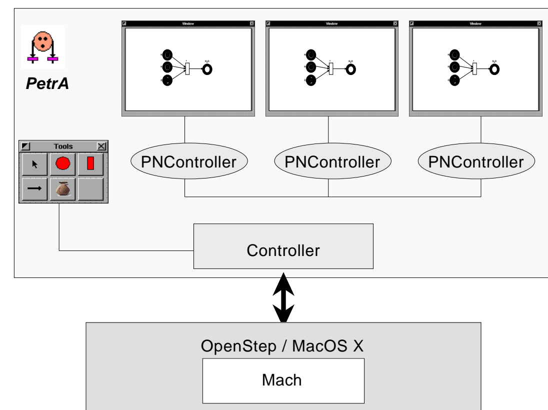

Figura 5: PetrA: Aplicación Multihilos

muestra la existencia un objeto de control (*Controller*), el cual se encarga de realizar las interacciones entre los controladores de cada red de Petri, el sistema operativo y algunas operaciones del usuario a través del menú de la aplicación y el panel de herramientas. El objeto *Controller* también se encarga de manejar los mensajes que se envían desde panel de herramientas a alguna red de Petri específica.

Cada red de Petri tiene asociado un objeto de control propio (*PNController*) que se encarga de manejar las operaciones específicas de la red de Petri con el ambiente de trabajo. Este objeto de control se encarga de mantener el identificador de la ventana de trabajo y el identificador del objeto gráfico que dibuja la red.

4.6 Las clases de la aplicación PetrA

En las figuras 6 y 7 se puede visualizar la relación entre las diferentes clases de la aplicación.

Las clases con la que se desarrolla la aplicacion son: *Controller, PNController, PNView, PNMatrix, Matrix, Element, Figure, Place, Transition* y *Connection*.

La clase Controller

El objeto de esta clase se encarga de establecer las interacciones entre el ambiente de trabajo (del sistema y de la aplicación) y el objeto de control de la red de Petri (*PNController*)

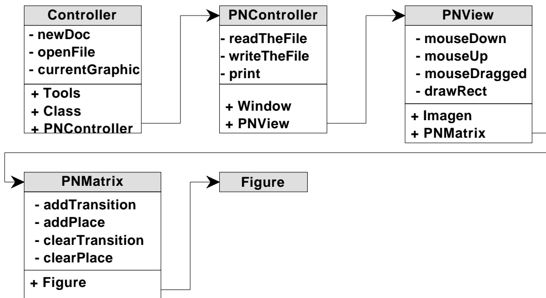

Figura 6: Relación entre clases

Esta clase se encarga de atender los mensajes que manda el usuario a través del menú principal. Este menú contiene un submenú etiquetado como *document*, donde se ecuentran agrupadas una serie de opciones de gran ayuda para trabajar con documentos (o archivos) asociados a la red de Petri. En esta opción el usuario puede abrir, crear, guardar y cerrar un documento. El objeto de la clase *Controller* trabaja estas opciones de la siguiente forma:

new document Cuando el usuario elige esta opción, entonces el objeto *Controller* crea e inicializa un objeto de la clase *PNController* y lo mantiene como el objeto actual de trabajo.

open document Cuando el usuario elige esta opción el objeto *Controller* abre una ventana para la selección del documento. Despues crea e incializa un objeto de la clase *PNController*, manteniéndolo como objeto actual de trabajo, y finalmente a este objeto *PNController* le manda el mensaje para que lea del archivo seleccionado. Este archivo debe almacenar los diferentes objetos que conforman la red de Petri (los objetos *PNView, Matrix, Element* y *Figure* que tenga la red).

save document Cuando el usuario elige esta opción, entonces el objeto *Controller* le manda un mensaje al objeto actual de trabajo para que escriba su contenido en un archivo. El nombre del archivo es un atributo que se almacena en los objetos *PNController*.

read document Cuando el usuario elige esta opción, entonces el objeto *Controller* le manda un mensaje al objeto actual de trabajo para que lea el contenido de un archivo para desplegar una red de Petri.

close document Cuando el usuario elige esta opción, entonces el objeto *Controller* le manda un mensaje al objeto actual de trabajo para que libere la memoria que está ocupando, esto es el objeto *PNController* le mandará un mensaje a todos los objetos de su Ventana para que se guarden. Si se detecta que la red de Petri a sufrido modificaciones, entonces se procede a notificar al usuario si desea perder las modificaciones o salvarlas.

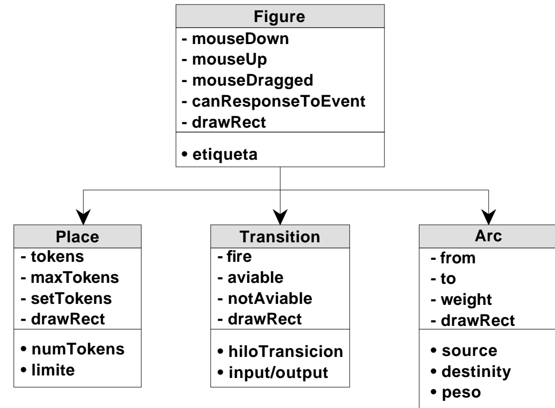

Figura 7: Jerarquía de Figura

El panel de herramientas mantiene un conjunto de objetos posibles que se pueden agregar a una red de Petri. El panel despliega opciones de objetos *Place, Transition* y *Arc* y la opción de selección. Cuando se opta por un objeto para agregase a una red de Petri, entonces *Controller* hace un llamado a un objeto contructor para que cree el nuevo objeto de clase correspondiente al objeto seleccionado. Cuando el objeto *PNController* requiera conocer la clase del objeto que agregará a la red de Petri, entonces le preguntara a Controller por el identificador de la clase a la que pertenece el nuevo objeto *Figure*.

En la implantación de este mecanismo se estableció una correspondencie entre el nombre del icono de la figura y su clase, con esta correspondencia evitamos que cuando se desee agregar una nueva clase al *PetrA*, únicamente se agrege la clase y el boton con su icono en el panel de herramientas, y de esta forma evitar agregar código a este objeto.

La clase PNController

Los objetos de esta clase se encargán de mandar mensajes a la red de Petri (representada con la clase *PNView* y la ventana que lo contiene). El nombre del documento asociado a la red es un atributo especial que tiene este objeto. Cuando el objeto se inicializa el nombre que tiene es *UNTITLED*, pero cuenta con métodos para cambiar este valor. Un objeto *PNController* se encarga de leer y/o escribir en el archivo correspondiente a los objetos que conforman la red de Petri.

La clase PNView

Esta clase representa al objeto red de Petri discutido en la sección 4.4. Los métodos de esta clase se encargan, principalmente, de procesar los eventos de ratón para saber si se agrega un elemento, se selecciona un elemento (para moverlo o borrarlo), o si se arrastra el ratón (para crear un arco o realizar el movimiento de un grafo).

Esta clase se encargará de mantener a un objeto *PNMatrix* (motor interno de la red de Petri) y a un objeto *análisis*[5], para realizar tareas de análisis.

[5] En este documento hemos omitido a los objetos de esta

Cuando se trabaja en el ambiente de desarrollo MacOS X y OpenStep se debe respetar la jerarquía *View* con la que trabajan los elementos gráficos de este sistema[15]. Así, los objetos PNView son los objetos *superview* de los objetos *figure*, por ende, los objetos *figure* son objetos *subview* del objeto PNView. Los objetos de esta clase deben saber el tipo de clase que van a agregar (*Place, Transition* o *Connection*) a la red de Petri, esto con el propósito de enviar el mensaje adecuado al objeto *PNMatrix*.

La jerarquía *View* que mantienen las aplicaciones de OpenStep y MacOSX no maneja de manera sencilla algunos comportamientos que se desean en los editores gráficos. Por ejemplo, para agregar un nuevo elemento gráfico entonces se selecciona el nuevo evento y se ejecuta un evento de ratón sobre el objeto *PNView*, pero si este evento ocurre sobre un objeto subview (*figure*) del objeto *PNView*, entonces quien atiende este evento es el objeto *figure* y no el *PNView*, con lo que estos objetos deberán pasar el evento al objeto *superview*. Este comportamiento implica que los objetos *Figure* deben ser capaces de tomar la desición de manejar el evento o pasarlo al superview. Los objetos *PNView* mantienen una comunicación muy estrecha con sus subviews para manejar de forma adecuada los distintos eventos del usuario que recibe para su manejo.

La clase Matrix

Los objetos de esta clase se encargan de manejar una matriz genérica, manejandola como una red de objetos que se interconectan entre si, como se observa en la figura 9. Los objetos de esta clase tienen métodos para agregar y/o quitar renglones o columnas, para avanzar a alguna posición de la matriz y proporcionar sus dimensiones (número de renglones y columnas).

La clase Element

Los objetos de esta clase son los que forman los nodos de la malla que forma la matriz. Estos objetos tiene identificadores a los elementos que se encuentran en las cuatro posiciones adyacentes a un nodo. También tendrá un identificador para poder asociarle cualquier objeto, lo que generara que la matriz sea genérica.

La clase PNMatrix

Esta clase es el motor interno principal de la red de Petri, pues mantiene los identificadores de los objetos gráficos que conforman la red de Petri. Esta clase es subclase de *Matrix*. Esta clase se encarga de llevar la relación del total de transiciones, lugares y arcos que tiene la red de Petri. Tiene los métodos para egregar y/o quitar transiciones o lugares.

Se seleccionó una forma matricial de conexiones entre los objetos gráficos de la red para representar al grafo dirigido, ya que, como se observa en el ejemplo de la figura 8, la forma tradicional para representar a un grafo dirigido, por ejemplo la figura 8*a*, lo hace con una lista de elementos (figura 8*b*), o una matriz de conectividad (figura 8*c*). La lista de elementos le da al grado una forma compacta y consistente de representación. Sin

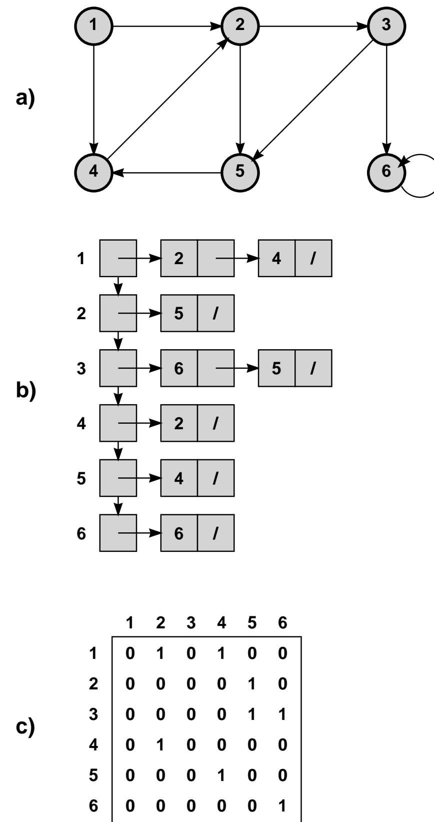

Figura 8: Representación tradicional de un grafo dirigido

embargo, como ya hemos mensionado, una red de Petri tiene dos tipos de grafos nodos y lugares, y los arcos que conectan a los nodos cumplen que no van de lugar a lugar o de transición a transición. Con lo que sería reduntdante meter una matriz de conectividad tradicional. Sin embargo si se toma en consideración a la matriz de incidencia asociada a una red de Petri, entonces podemos observar que los lugares se representan en los renglones y las transiciones en las comlumnas.

Otro aspecto que se debe tener en consideración es que cuando se está editando una red de Petri, su número de grafos varía (incluyendo cuando se aplican ténicas de reducción de uso de subredes). Por lo que resultaría dificil y redundante hacerlo con una estructura estática con una matriz de conectividad. Si se utiliza la representación por listas ligadas, resultaría muy costoso y dificil el mantener todos las ligas actualizadas, además de que este método hace que sea difícil que una transición encuentre sus lugares de entrada. Otro aspecto en contra es que los arcos se están tratando como si fueran objetos y deben almacenarse y manejarse en algún lugar, si se trabaja con la lista ligada habría que crear otra lista de arcos y mantener un control de índices entra ambas listas, lo cual hace más enredado el manejo.

Si trabajamos con un enfoque matricial (figura 9)de elementos gráficos obtenemos:

- Los objetos *Place* se almacenan el la primera columna de la matriz (teniendo una lista ligada entre los diferenetes lugares.

---

clase debido a que su diseño e implantación se ará cuando se llegue a la segunda fase

- Los objetos *Transition* se almacenan en el primer renglon de la matriz (teniendo una lista ligada entre las diferentes transiciones.
- Los objetos *Connection* se encuentran en los elementos que forman la intersección entre columnas *transitions* y renglones *places*. Permitiendo que el tiempo de búsquedas para los lugares de entrada y salida de una transisicón sea el mismo.

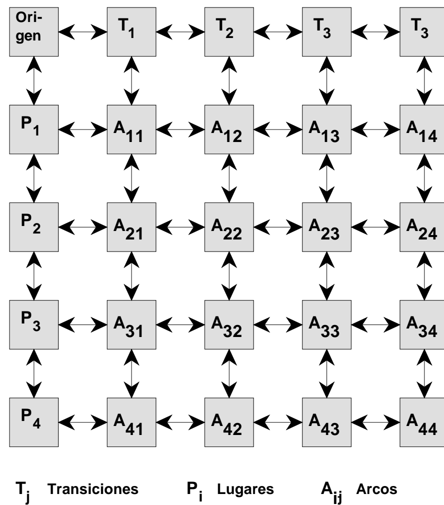

Figura 9: Representación matricial para redes de Petri

El objeto matriz se representa como una red de elementos ligados en cuatro direccines: arriba, abajo, adelante, atrás. Con esta representación el manejo dinámico de objetos (agregar y quitar nodos) se hace muy fácil con métodos para borrar y crear columnas (eliminar y crear transición respectivamente), y métodos para borrar y crear renglones (eliminar y crear lugar). Como la matriz almacena a diferentes objetos se debe crear para que sea genérica (que almacene cualquier tipo de objetos), con lo cual obtenemos que podríamos almacenar en uno de sus elementos a otra matriz, lo cual será muy útil cuando se llegue a la fase de manejo de subredes.

La clase Figure
Esta es una clase genérica, y sirve de base para crear las clases *Place, Transition* y *Conecction*. Esta clase contiene la etiqueta sociada a un grafo o arco, y su coordenada inicial. contiene los métodos para manejar los eventos del usuario de manera coordinada con el objeto *PNView* (superview).

La clase Place
Los objetos de esta clase representas los nodos lugar de la red de Petri, cuentan con un número de tokens, una variable para indicar el números máximo de tokens (si se desea trabajar con una red de Petri acotada). Posteriormente se planea asociarle un objeto que maneje los distintos tipos de tokens que tiene el lugar (en caso de desear trabajar con redes coloreadas).

La clase Transition
Los objetos de esta clase deben ejecutar acciones sobre los lugares de la red de Petri, así, estos objetos tiene asociados un objetos hilo que se encargará de leer de sus lugares de entrada y arcos el número de tokens y pesos para verificar si está en posibilidades de disparar, en caso de estarlo, procederá a escribir en los lugares de salida el valor correspondiente. Entonces delegamos al sistema la selección de la transición que se eligirá cuando se ejecute la red de Petri.

Debido a que tenemos que los tokens son atributos de los objetos *Place* es necesario que los objetos *Transition* envien mesajes a los objetos *Place* que tiene en la entrada, sin embargo, esta información la tiene el objeto del tipo PNMatrix, por lo que se opto en que cada objeto transition, tenga un identificador a la columna de la matriz donde se encuentra.

La clase Connection
Esta clase tendrá asociada la representación gráfica del arco dirigido entre los nodos de la red. Los beneficios que obtenemos al tratar al arco un objeto es que el arco tendrá que manejar su valor de peso, y no dejar este atributa para la transición. Otra ventaja es que una vez que se conocen los nodos de entrada y salida podemos trazar una recta entre estos dos puntos, sin embargo, sabemos que mientras mas grande y compleja es una gráfica, es deseable que los arcos tracen curvas entre los puntos seleccionados, asi este objeto tendrá la capacidad de trazar curvas entre dos puntos de un plano. En la versión de OpenStep se le incorpora el algoritmo de Bezier para su funcionamiento. En la versión de MacOS X no se necesita debido a que esta plataforma cuenta con una clase que realiza esta tarea[6].

5 Resultados

Contamos con la infraestructura para trabajar con una aplicación que trabaja redes de Petri con la posibilidad de extender el uso las distintos tipos de redes de Petri.

6 Trabajos Futuros y extensiones

El trabajo inmediato es terminar las tres fases que se mencionaron en la sección 4. Para ello, se incorporará un objeto de análisis, a la clase *PNView*, el cual interactuará con el objeto de la clase *PNMatrix* para trabajar los métodos de análisis matriciales. Finalmente se debe incorporar el manejo de subredes de Petri, lo cual resulta sencillo si tomamos en consideración que estas subredes la podemos representar como otro objeto *PNView* (con sus objetos de análisis y de contención de objetos) asociado a una ventana. Para trabajar estas subredes en la red de Petri base, cada subred se maneja y incorpora al objeto de la clase *PNMatrix* como si fuera una transición.

Por otro lado, las extensiones que se antojarían de manera inmediata serian:

1. Agregar el manejo de redes de Petri coloreadas, esto de manera sencilla con la incorporación de un nuevo objeto *Place* que contenga una lista de numeros que corresponda con cada color token que se desee manejar. Además con el uso de los hilos en los objetos transición se podrían ejecutar de manera paralela.

[6] La clase *BezierPath*.

2. Agregar el manejo de redes de Petri estocásticas creando una subclase de *Transition* donde se incorporen retardos en la ejecución del hilo de la transición[4].
3. Intentar distribuir el trabajo de la ejecución de los módulos de la aplicación para taner un mejor desempeño cuando se trabaje con redes de Petri grandes.

7 Conclusiones

Hemos encontrado que la representación matricial interna de la red de Petri nos ayuda a dotar a la aplicación de una manera sencilla carácterísticas que parecerían dificiles con el modelo tradicional de representación de grafos.

Además de que la incorporación de nuevas clases de tipo *Transition* y *Place* se pueden agregar de manera natural para ampliar el tipo de redes de Petri que maneja.

Referencias

[1] J.L. Peterson, "Petri Nets Theory and The Modeling of Systems", Prentice-Hall, N.J., 1981.

[2] T. Murata, "Petri Nets: Properties, analisys and applications", Proc. IEEE, vol 77, pp. 541-579, Apr. 1989.

[3] Mu Der Jeng and Frank DiCesare, "A Review of Synthesis for Petri Nets with Applications to Automated Manufacturing Systems", IEEE Transc. on systems, man, and cybernetics, vol 23, no.1, January/February 1993.

[4] Fred D.J. Browden; "Modelling Time in Petri Nets"; July, 1996.

[5] Fred D.J. Browden; "Role-Based Extended Petri Net Models and their Applications"; International Congress on Modelling and Simulation, 1995, Newcastle, Australia.

[6] Holger Giese, Jörg Graf and Guido Wirtz; "Modeling Sofware Systems with Object Coordination Nets"¡ Intitut für Informatik; Westfälische Wilhelms-Universität, Germany, 1998.

[7] Stefan Schöf, Michael Sonnenschein, Ralf Wietin, "High-level Modeling with THORNs"; Oldenbuger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge- und Systeme (OFFIS); Escherweg 2. D-26121 Oldenburg (Germany).

[8] Patrick Lam; "A Petri Net Simulator in Java – Design document", February, 1999, http://www.sable.mcgill.ca/~plam/petri,

[9] Monika Heiner, "Petri Net Based Software Dependability Engineering - Tutorial Notes", Brandenburg University of Technology, Department of Computer Science, Germany, 1995.

[10] Noriega Ponce, Alfonzo; "Un ambiente para el desarrollo de controladores lgícos programables en computadoras personales"; Tesis M.C, CINVESTAV, Ing. Eléctrica, México D.F., 1992.

[11] Valdemar Ganzález Avila, "Simulación con Redes de Petri", Tesis de M.C., CINVESTAV, Ing. Eléctrica, CINVESTAV, 1993, México, D.F.

[12] G.J. Holzmann; "Design and Validation of Computer Protocols"; Prentice Hall, 1991.

[13] H.J. Schneider and A.I. Wasserman; "Automated Tools for Information Systems Designs", Edit. North Holland, 1991.

[14] Mendel Rosenblum and Mani Varadarajan, "SimOS: A Fast Operating System Simulation Environment"; Technical Report:CSL-TR-94-631; Computer Systems Laboratory, Department of Electrical Engineering and Computer Science; Standford University; July 1994.

[15] NeXT Computer, Inc., "NeXT Manuals", 1995.