UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE ELECTRÓNICA, TELEMÁTICA Y AUTOMÁTICA

TESIS DOCTORAL

METODOLOGÍA PARA LA SIMULACIÓN DE REDES DE PETRI DE ALTO NIVEL TEMPORIZADAS

PABLO VICENTE HERNÁNDEZ MORERA

Las Palmas de Gran Canaria, 1996

23/1996-97

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
UNIDAD DE TERCER CICLO Y POSTGRADO

Reunido el día de la fecha, el Tribunal nombrado por el Excmo. Sr. Rector Magfco. de esta Universidad, el/a aspirante expuso esta TESIS DOCTORAL.

Terminada la lectura y contestadas por el/a Doctorando/a las objeciones formuladas por los señores miembros del Tribunal, éste calificó dicho trabajo con la nota de APTO CUM LAUDE
Las Palmas de Gran Canaria a 12 de febrero de 1997.

El/a Presidente/a: Dr. D. Juan Antonio de la Puente Alfaro,

El/a Secretario/a: Dr. D. Fco. Javier Guerra Santana,

El/a Vocal: Dr. D. Juan Carlos Dueñas López,

El/a Vocal: Dr. D. Javier Miranda González,

El/a Vocal: Dr. D. Miguel Angel Ferrer Ballester,

El Doctorando: D. Pablo Vicente Hernández Morera,

DEPARTAMENTO DE ELECTRÓNICA, TELEMÁTICA Y AUTOMÁTICA

Escuela Técnica Superior de Ingenieros de Telecomunicación
Universidad de Las Palmas de Gran Canaria

Tesis Doctoral

Metodología para la simulación de redes de Petri de alto nivel temporizadas

Autor
Pablo Vte. Hernández Morera
Ingeniero de Telecomunicación

Director
Juan Domingo Sandoval González
Dr. Ingeniero de Telecomunicación

Año 1996

DEPARTAMENTO DE ELECTRÓNICA, TELEMÁTICA Y AUTOMÁTICA

Escuela Técnica Superior de Ingenieros de Telecomunicación
Universidad de Las Palmas de Gran Canaria

Tesis Doctoral

Metodología para la simulación de redes de Petri de alto nivel temporizadas

Autor
Pablo Vte. Hernández Morera

Director
Juan Domingo Sandoval González

Año 1996

Resumen

La aparición de arquitecturas basadas en la existencia de varios procesadores fuertemente acoplados ha impulsado la aparición de técnicas de programación paralelas, donde el programa secuencial es particionado en subconjuntos disjuntos y ejecutado por diferentes procesadores.

Un campo de aplicación es en las simulaciones debido a su alto coste computacional. El formalismo elegido para la representación de los sistemas a simular es el de las Redes de Petri.

El objetivo de la Tesis es obtener un algoritmo que extraiga el máximo paralelismo de una Red de Petri de alto nivel temporizada garantizando unos resultados de simulación correctos, con la intención de facilitar su ejecución sobre una arquitectura multiprocesadora y disminuir su tiempo de ejecución. Los resultados obtenidos sólo serán correctos si se mantienen invariantes las relaciones de causalidad existentes entre los disparos.

El algoritmo de simulación paralela se basa en el concepto de distancia temporal entre dos transiciones de la Red. Mediante dicho concepto se conoce la posible interacción de dos transiciones en el tiempo, con el objeto de obtener el conjunto de transiciones de la Red mutuamente independientes para ser disparadas concurrentemente.

Cuando el conjunto de disparos o tareas a realizar es mayor que el número de procesadores surge la necesidad de optimizar la distribución de tareas, por lo que se analizan algunos algoritmos de distribución heurísticos.

Finalmente, el algoritmo de simulación paralela propuesto es ensayado sobre un modelo de red local Ethernet.

Palabras clave: simulación paralela, Red de Petri temporizada, sistema multiprocesador fuertemente acoplado, relación de causalidad, algoritmos de distribución de tareas.

Abstract

With the emergence of multiprocessors systems, a considerable effort is being made to develop techniques for parallel programming, where the sequential program is partitioned into nonoverlay subsets and executed by different procesors.

A possible scope is in the simulations since they are very time consuming. The formalism choosen for the representation of the system to be simulated is the Petri nets.

The objective of this thesis is to provide an algorithm to extract the maximum parallelism of a high level timed Petri net which guarantees a correct simulation results, with the idea of executing on a multiprocessor system and decreasing the execution time. The results produced will only be correct if the causality relations of the firings are kept invariant.

The algorithm for parallel simulation is based on the time distance concept between two transitions of a Petri net. This concept is used to find out the interaction between two transitions in time, in order to achieve the set of transitions of the net mutually independent to be fired at the same time.

Some heuristics scheduling are examined to generate an optimum allocation when the number of firings or tasks to be executed is greater than the number of processors.

Finally, the suggested algorithm for parallel simulation is evaluated on a Ethernet local network model.

Keywords: parallel simulation, timed Petri net, tightly coupled multiprocessor, causality relation, task assignment scheduling.

Agradecimientos

A la hora de agradecer la culminación de este trabajo de investigación que es mi Tesis Doctoral, quiero dar las gracias en primer lugar a mi esposa Isabel, por el apoyo constante que desde el principio he tenido, sin el cual no hubiera visto la luz el presente estudio.

En segundo lugar, a la dirección, entusiasmo y apoyo brindados por el director de este trabajo, Dr. D. Juan Domingo Sandoval González, Profesor Titular de "Ingeniería Telemática" de la Escuela Superior de Ingenieros de Telecomunicación de Las Palmas de Gran Canaria.

Por último agradezco el apoyo financiero prestado por K.R.H.K. MAYA S.A., a través del patrocinio de una beca concedida bajo el amparo institucional de la Fundación Universitaria de Las Palmas.

A todos ellos mi eterna gratitud.

Indice

Nomenclatura

Notación

L_i: lugar *i* de una red de Petri

t_j: transición *j* de una red de Petri

$\alpha(t)$: conjunto de lugares de entrada de la transición t

$\{t_i, t_j, \ldots, t_n\}$: secuencia de disparo de transiciones de una red de Petri, donde t_i se ejecuta con anterioridad a t_j, ..., y t_n es la última transición ejecutada

m_0: marcado inicial de una red de Petri

m_k: marcado del estado *k* de una red de Petri

y.x: identificador x del token y

ER: red de Petri Environment/Relationship

TER: red de Petri Environment/Relationship temporizada

STER: red de Petri Environment/Relationship temporizada fuerte

C(e): instante de simulación del evento e

S(t): chronos de sensibilización de *t*

$S_L(t)$: chronos del token sensibilizador de *t* perteneciente al lugar L

$TI_L(t)$: chronos del token inducido en L por la transición *t*

TMI: conjunto de transiciones mutuamente independientes

$s_{min}(t)$: conjunto de transiciones con tiempo de disparo menor que C(t)

$s_{eq}(t)$: conjunto de transiciones con tiempo de disparo igual que C(t)

$s_{max}(t)$: conjunto de transiciones con tiempo de disparo mayor que C(t)

$t_{i,j}$: disparo de t_i en el instante de tiempo *j*

a => *b*: *a* tiene una relación de causalidad con *b*

$d(t_1, t_n)$: distancia temporal desde t_1 a t_n

ß(e): lookahead del evento *e*

M: cantidad de procesadores

P: cantidad de procesos

p_m: proceso *m*

P_k: cantidad de procesos asignados al procesador *k*

U_{ij}: tiempo de transmisión del siguiente mensaje desde p_i a p_j

W_{ij}: tiempo de transmisión del siguiente mensaje desde p_i a p_j (valor mejorado)

LFD: lista de futuros disparos

BCT: bloque de control de tarea

µs: microsegundos

Capítulo 1: Introducción

1. Introducción

En muchas ocasiones, un fenómeno es estudiado examinando un modelo que lo represente. Un modelo es una representación de las principales características del sistema a estudiar. Su manipulación proporciona nuevos datos sobre su comportamiento en diferentes situaciones sin incurrir en el coste o peligrosidad de la manipulación del fenómeno real.

La definición y manipulación del modelo es llevada a cabo mediante técnicas que permiten el estudio cuantitativo del sistema real al que se pretende representar. Estas técnicas son denominadas simulaciones y se realizan mediante programas de ordenador muy complejos y con un gran coste computacional.

El modelo de un sistema real debe comprender: un conjunto de variables que permitan conocer el estado del sistema y los eventos o sucesos que puedan alterar el valor de las variables. Algunos formalismos permiten añadir al modelo una semántica temporal. Según el tipo de evolución temporal del modelo cabe distinguir entre modelos continuos y discretos.

En los modelos continuos las variables del sistema evolucionan de forma continua con respecto al tiempo. En los modelos discretos, las variables sólo cambian en un conjunto discreto de instantes de tiempo.

Pretendemos plantear en esta tesis un método para la simulación paralela de modelos discretos que distribuya la carga computacional entre varios procesadores, disminuyendo de esta forma el tiempo de ejecución de la simulación.

Los modelos discretos serán representados mediante Redes de Petri [PET 77][FER 87][JEN 87][MUR 89]. Las Redes de Petri cuentan con varias ventajas con respecto a los modelos obtenidos utilizando autómatas finitos, como la descripción de sistemas más complejos (con infinitos estados), así como de aquellos sistemas con funcionamiento asíncrono y concurrente. Las Redes de Petri proporcionan una semántica clara para definir el comportamiento del sistema, los aspectos temporales y de datos de una forma clara y no ambigua, con el grado de detalle deseado por el diseñador del modelo.

En los siguientes puntos se expondrán los campos de investigación: Redes de Petri, simulación paralela y algoritmos de distribución de procesos, donde se establecen las bases desde donde partirá el desarrollo de esta tesis.

1.1. Estado del arte en Redes de Petri

Las Redes de Petri constituyen un método simple y potente para describir y analizar el flujo de información y control de un sistema, mediante el modelado de los eventos y condiciones de un sistema, y la relación existente entre ellos.

Una red de Petri es una cuádrupla

$$\{L, T, \alpha, \beta\}$$

donde

L: es un conjunto finito, no vacío, de lugares.

T: es un conjunto finito, no vacío, de transiciones tal que $L \cap T = \varnothing$.

α: $T \rightarrow f(L)$ es la función de entrada, que define para cada transición sus lugares de entrada.

β: $T \rightarrow f(L)$ es la función de salida, que define para cada transición sus lugares de salida.

La representación más utilizada de la Red de Petri es un grafo que modela las propiedades estáticas del sistema. El grafo contiene dos tipos de nodos: lugares y transiciones, representando las condiciones y los eventos del sistema respectivamente. Los lugares se representan mediante círculos y las transiciones por medio de líneas rectas. Ambos tipos de nodos se unen mediante arcos dirigidos que representan a las funciones α y β. Si un arco se dirige del nodo i al nodo j, entonces i es una entrada a j, y j es una salida de i. Los arcos dirigidos relacionan los lugares con las transiciones y viceversa, describiendo las condiciones necesarias para que un evento se ejecute, así como las condiciones resultantes de la realización de un evento.

En la figura 1.1 se muestra un grafo de una Red de Petri con los lugares L_1, L_2, L_3, L_4, L_5 y L_6, y las transiciones t_1, t_2, t_3 y t_4. L_1 es una entrada a la transición t_1, mien-

tras que L_3 es una salida de t_1. De igual forma, t_1 es salida de L_1 y entrada a L_3.

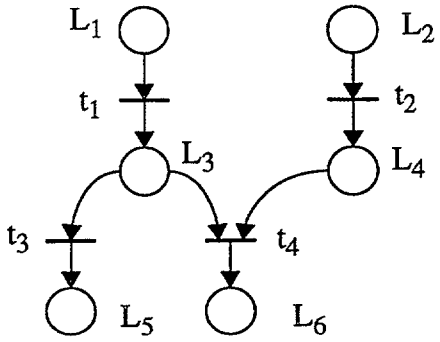


Figura 1.1. Grafo de una Red de Petri

El **marcado o estado de una Red de Petri** es una asignación de marcas o tokens a los lugares

$$\mu : L \rightarrow N$$

y es indicado mediante el vector

$$m = \{\mu_1, \mu_2, \ldots, \mu_n\}$$

donde el número de tokens en el lugar L_i es $\mu_i=\mu(L_i)$, i={1, ... , n} y n es el número de lugares de la Red. El marcado inicial de la Red se denota como m_0.

Las marcas o tokens se representan mediante puntos negros alojados en los lugares de la Red.

El movimiento de los tokens en una Red de Petri se realiza mediante los disparos de las transiciones de la Red. Para realizar el disparo de una transición, ésta debe encontrarse **sensibilizada**, esto es, todos sus lugares de entrada deben contener al menos un token

$$(\forall L_i \in \alpha(t_j))\ (\mu(L_i) > 0)$$

El disparo de una transición consiste en desalojar un token de cada uno de sus lugares de entrada, generando un nuevo token en cada lugar de salida de la transición disparada. Los tokens generados reciben el nombre de tokens de salida.

El conjunto de tokens que sensibilizan a una transición se denomina **tupla sensibilizadora.**

La ejecución de una Red de Petri es una secuencia de disparos de transiciones que da lugar a diferentes distribuciones de tokens m_j a partir de un marcado inicial, con j={1, ... ,k} donde k es el número de disparos.

El conjunto de todos los estados o marcados de una Red junto con el marcado inicial m_0 constituye el **espacio de estados** o **conjunto de alcanzabilidad** de la Red de Petri.

Un marcado m' se dice que es alcanzable desde otro marcado m si existe una serie de transiciones sensibilizadas cuyo disparo conducen a m', bien inmediatamente o a través de marcados intermedios.

La ejecución de la Red de Petri modela el comportamiento dinámico del sistema. El flujo de información resultante de la ejecución del sistema, se representa mediante la posición y el movimiento de los tokens. La posición de un token en un lugar determinado indica el cumplimiento de la condición representada por ese lugar.

En la figura 1.2(a) se muestra el grafo de la figura 1.1 con un token en los lugares L_1 y L_2. Tanto la transición t_1 como t_2 se encuentran sensibilizadas porque sus lugares de entrada contienen un token. El disparo de t_1 desaloja el token existente en L_1 y genera un nuevo token en L_3. El disparo de t_2 retira el token de L_2 y crea un nuevo token en el lugar L_4. El resultado de ambos disparos se muestra en la figura 1.2(b).

Especial atención merece el estado de la Red de Petri indicado en la figura 1.2(b), donde las transiciones t_3 y t_4 se encuentran sensibilizadas. El disparo de t_3 retira el token que reside en L_3 y genera un nuevo token en L_5, quedándose la transición t_4 desensibilizada por la ausencia del token en su lugar de entrada L_3. Por otra parte, el disparo de t_4 retira los tokens de sus lugares de entrada L_3 y L_4, y genera un nuevo token en L_6, desensibilizando a t_3. Se observa que el disparo de cualquiera de las transiciones t_3 o t_4, desensibiliza a la otra. Estas transiciones se encuentran en una **situación de conflicto.** Esta situación viene a modelar la circunstancia de dos eventos que no pue-

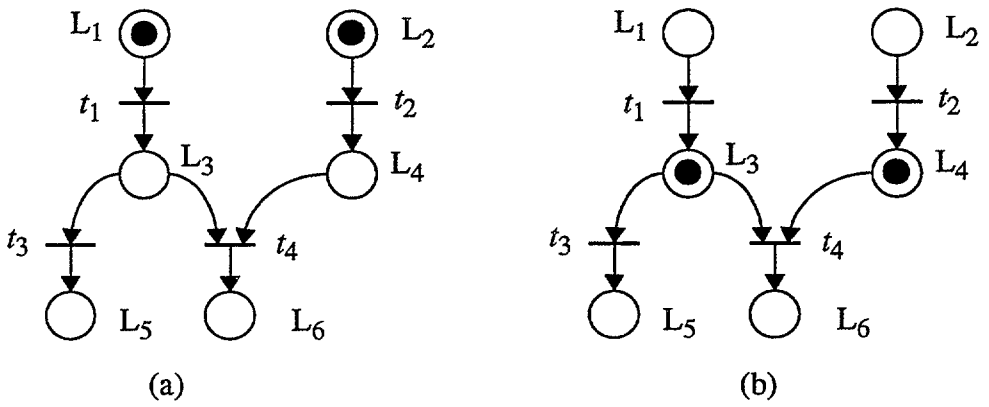


Figura 1.2. (a) Antes del disparo (b) después del disparo

den ser realizados simultáneamente.

La ejecución de una Red de Petri es una secuencia de disparos cuyo orden de ocurrencia es uno de los posiblemente permitidos, modelando de esta forma sistemas donde varias cosas pueden suceder simultáneamente y su orden de ocurrencia no es único.

En la figura 1.2 se observa que la ejecución de la red no es única, siendo posible obtener las siguientes ejecuciones:

m_0={1,1,0,0,0,0}, m_1={0,1,1,0,0,0} y m_2={0,0,1,1,0,0}
m_0={1,1,0,0,0,0}, m_1={1,0,0,1,0,0} y m_2={0,0,1,1,0,0}

Las Redes de Petri son no deterministas debido a la ambigüedad en el momento de decidir qué transición sensible debe ser disparada.

1.1.1. Propiedades de las Redes de Petri

Las características más importantes de las Redes de Petri son su capacidad para modelar la concurrencia y la ejecución asíncrona, lo que las hace ideales para simular sistemas con estas características.

La concurrencia queda reflejada en los disparos de las transiciones t_1 y t_2 de la figura 1.2. Ambas transiciones pueden ser disparadas simultánea e independientemente. Por lo tanto, el cambio de estado de la Red producido por el disparo de t_1 puede suceder en cualquier momento independientemente del estado en que se encuentre el otro ramal de la Red (formado por L_2, t_4 y L_4).

La característica asíncrona se pone de manifiesto en el hecho de que no existe medida del tiempo en la ejecución de la Red de Petri, sino un orden parcial en la ocurrencia de los disparos. Las Redes de Petri modelan la variabilidad de los estados en los que se puede encontrar un sistema, sin considerar la duración de cada uno de ellos.

Otra característica destacable es su ejecución no determinista. Cuando una transición se encuentra sensibilizada no existe la obligación de dispararla, lo que da la posibilidad de elegir en un momento dado qué transiciones deseamos disparar y cuales no. Y por lo tanto el orden parcial de los disparos realizados en una Red, es tan sólo uno de los posibles. Esta característica se muestra en la figura 1.2(b), donde las transiciones t_3 y t_4 están sensibilizadas y sólo una de ellas puede ser disparada. Esta característica intenta reflejar situaciones de la vida real, donde el orden de ocurrencia de los eventos no es único.

Analizando algunas de las propiedades del modelo podemos obtener conocimiento de muchas propiedades de nuestro sistema.

A continuación se definirán las propiedades más importantes de una Red de Petri.

Una Red de Petri es **token viva** si para cada token incluido en un marcado *m* perteneciente a su espacio de estados existe otro marcado *m'* alcanzable desde *m* en el cual el token pertenece a una tupla sensibilizadora.

Los tokens que no cumplen la condición anterior se denominan **tokens muertos.**

Una Red de Petri es **transición viva** si para cada marcado *m* perteneciente a su espacio de estados y para cada transición *t*, existe otro marcado *m'* alcanzable desde *m* en el cual *t* es sensible.

Las transiciones que no cumplan esta condición se denominan **transiciones muertas.** La existencia de este tipo de transiciones puede conducir a situaciones de blo-

queo de la Red. Por el contrario, las transiciones que son disparables en todos los marcados del espacio de estados de la Red se conocen como transiciones vivas.

Una Red de Petri es **red viva** si en cualquier marcado *m* de su espacio de estados siempre existe al menos una transición sensible.

La propiedad transición viva implica la propiedad red viva, pero la propiedad token viva no necesariamente implica transición viva como se muestra en la Red de Petri de la figura 1.3. La Red es token viva, sin embargo la existencia de la transición muerta t_3 evita que la Red sea considerada transición viva.

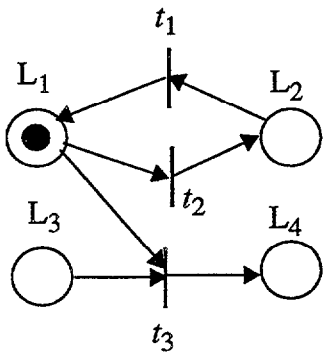


Figura 1.3. Red token viva

Una Red de Petri es ***k*-limitada** si el número de tokens existentes en cualquier lugar de la Red está limitado a *k*. Como caso particular, cuando *k* toma el valor unidad la Red se denomina **segura.**

Una Red de Petri es **conservativa** o **consistente** si el número de tokens en la Red se mantiene constante a través de su ejecución. Para detectar esta propiedad debe existir una asignación de números enteros mayores que cero a cada transición, de tal manera que en cada lugar de la Red la suma de los números asignados a sus transiciones de entrada sea igual a la suma de los números asignados a sus transiciones de salida. Un ejemplo de este tipo de Redes se ilustra en la figura 1.4. Si una Red de Petri es transición viva y conservativa, los estados por los que atraviesa su ejecución forman un ciclo, retornando la Red siempre a su marcado inicial [RAM 80][MUR 89].

La mayoría de las técnicas de análisis de una Red de Petri se reducen a un pro-

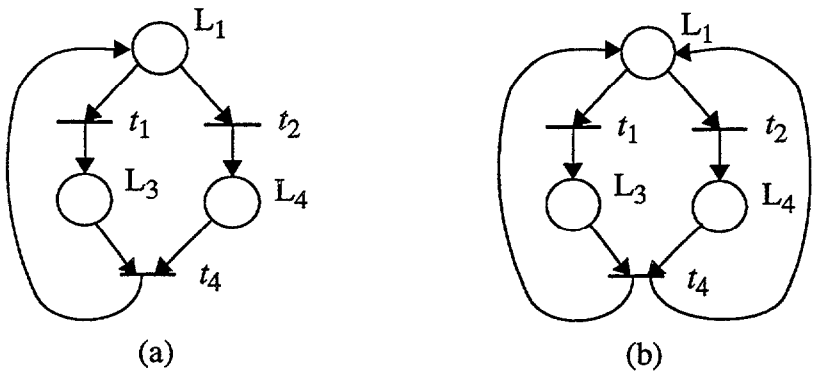


Figura 1.4. Red de Petri (a) inconsistente y (b) consistente

blema de alcanzabilidad, donde se intenta averiguar si en una Red de Petri con un marcado dado *m* se puede alcanzar otro marcado conocido *m'*.

El problema de alcanzabilidad se resuelve obteniendo el árbol de alcanzabilidad. Arbol cuyos nodos representan los estados de la Red, y los arcos representan los cambios de estado producidos por el disparo de alguna transición que conducen de un estado a otro. El árbol de alcanzabilidad del grafo de la figura 1.2 es el indicado en la figura 1.5.

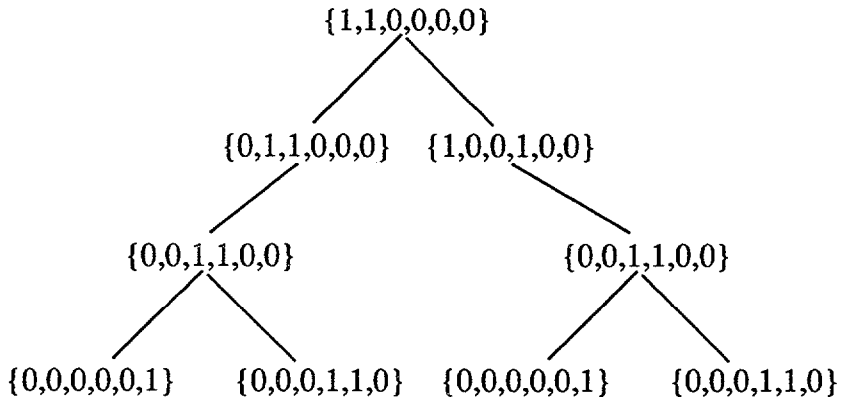


Figura 1.5. Arbol de alcanzabilidad de la Red de la figura 1.2

La obtención del árbol de alcanzabilidad responde de forma inmediata al problema de alcanzabilidad (¿Es alcanzable el estado *m'* desde el estado *m*?), incluso proporciona información acerca de la existencia de bloqueos en la Red, limitación de

tokens, si la Red es segura y otros.

La obtención del árbol de alcanzabilidad de una Red de Petri puede llegar a ser una tarea imposible cuando el conjunto de alcanzabilidad es infinito.

1.1.2. Extensiones de las Redes de Petri

Dos de las extensiones más utilizadas son las Redes de Petri generalizadas y los arcos inhibidores.

Las Redes de Petri generalizadas permiten que un lugar proporcione o reciba más de un token con el disparo de una transición, gráficamente se indica mediante un número asociado al arco que indica el número de tokens a compartir. Un ejemplo se muestra en la transición t_1 de la figura 1.6. Su disparo desaloja dos tokens del lugar L_1 y un token del lugar L_3, creando un nuevo token en L_2 (si el arco no tiene un número asociado, éste toma el valor uno).

Un arco inhibidor que conecta un lugar L a una transición *t*, permite el disparo de la transición *t* si y sólo si no existen tokens en L. El disparo de la transición a través del arco inhibidor no implica consumo de tokens y su utilización aumenta la capacidad descriptiva de las Redes. Su representación gráfica se muestra en la figura 1.6, donde el arco inhibidor añade a la transición t_1 la condición de que L_2 se encuentre vacío.

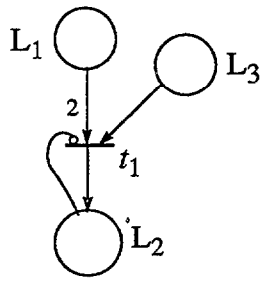


Figura 1.6. Red de Petri generalizada con arco inhibidor

Hoy en día, existe una gran dificultad para representar sistemas complejos con este tipo de Redes. Esta situación ha derivado en la aparición de las Redes de alto nivel que contemplan la capacidad de integrar aspectos funcionales de nuestro sistema en la Red de Petri, y en la aparición de las Redes de Petri temporizadas con la introducción

del factor tiempo, posibilitando el modelado de sistemas de tiempo real [DUE 94].

1.1.1.3. Redes de alto nivel

Las primeras Redes de Petri de alto nivel fueron las Redes de Petri coloreadas. Estas Redes se caracterizan por diferenciar los tokens de la Red mediante colores, de manera que una transición estará sensibilizada si y sólo si en sus lugares de entrada se encuentran los tokens del color exigido [JEN 87].

Este tipo de Redes no añaden más potencia de modelado con respecto a las Redes de Petri vistas hasta ahora, puesto que cualquier Red de Petri coloreada puede ser transformada en una Red de Petri convencional. Sin embargo, mejoran el poder descriptivo de las Redes de Petri.

Más tarde, se implementaron Redes donde los tokens eran tuplas de variables, y las transiciones incluían expresiones o funciones. Son las conocidas como Redes Environment/Relationships (denotadas ER) [BIL 88][GHE 91] que tienen las siguientes extensiones:

- Los tokens llevan asociado un conjunto de variables o identificadores con sus respectivos valores.

- Cada transición tiene asociada una acción compuesta de un predicado y un resultado. El predicado de una transición comprende el conjunto de condiciones definido sobre los identificadores de los tokens de los lugares de entrada de la transición, que indican si la transición está sensibilizada y establecen los tokens que forman la tupla sensibilizadora. El resultado indica las condiciones que debe satisfacer los tokens generados por el disparo de la transición. Estas condiciones deben especificar inequívocamente los identificadores que forman la estructura de los tokens de salida, así como sus respectivos valores.

En las Redes ER se emplea la siguiente notación: para especificar el identificador x de los tokens que se encuentran en el lugar L_y, se escribirá $L_y.x$. Las acciones se engloban entre llaves, especificando los lugares de la Red que intervienen en el disparo, el predicado y el resultado en el orden indicado.

{Lugares participantes| Predicado → Resultado}

Los lugares de entrada y salida de la transición se delimitan mediante los símbolos "<" y ">" del siguiente modo

< Lugares de entrada, Lugares de salida >

Cuando el número de lugares de entrada o salida excede de la unidad, estos se encierran a su vez entre llaves "< >". Seguidamente, se especifican las condiciones que componen el predicado y el resultado, separados por el símbolo "→".

A continuación se muestra una acción donde L_1 y L_2 son los lugares de entrada de la transición, y L_3 de salida. El predicado está formado por dos condiciones que imponen que el valor del identificador *a* sea igual en los tokens sensibilizadores, y el identificador *b* sea mayor en el token de L_2 que en el de L_1. El disparo de la transición produce un token en L_3 con los identificadores y valores indicados en las dos condiciones que componen el resultado: un valor de *a* igual al correspondiente valor del token sensibilizador de L_1, y un valor de *b* que satisfaga la última condición indicada en la acción.

$$\{<<L_1,L_2>,L_3>|\ L_1.a=L_2.a \text{ y } L_1.b<L_2.b \rightarrow L_3.a=L_1.a \text{ y } L_1.b<L_3.b<L_2.b\}$$

Cuando un mismo lugar L se encuentre incluido en una acción como lugar de entrada y de salida, L hará referencia a los token del lugar que sean entrada y L' a los de salida.

La mayor parte de las propiedades definidas para las Redes de Petri pueden ser aplicadas a las Redes ER. Tan sólo cabe destacar las siguientes modificaciones:

- El estado de una Red ER viene especificado por el marcado de la Red junto con los valores de las variables o identificadores de cada token.

- A diferencia de las Redes de Petri convencionales, en las Redes ER la existencia de un token en cada lugar de entrada de una determinada transición no supone necesariamente que esa transición esté sensible. También es necesario que cada uno de los token cumpla las condiciones especificadas en el predicado de la transición. Esto puede derivar en situaciones en las que un token no llegue a cumplir nunca el predicado de la transición, lo que impide el avance

de dicho token por el resto de la Red, dando lugar a un token muerto.

Una Red ER es ***k*-limitada débil** únicamente si en cada estado alcanzable por la Red y para cada lugar, el número de tokens de un lugar que pertenezcan a alguna tupla sensibilizadora no es mayor que *k*. Esta propiedad no limita el número de tokens de un lugar, tan sólo aquellos que pudieran pertenecer a una tupla sensibilizadora. Es claro que una Red limitada en su concepto más amplio es necesariamente limitada débil y no viceversa.

También se puede extender la definición de transiciones en conflicto ya dada anteriormente contemplando dos tipos de conflicto: **conflicto estático** y **conflicto dinámico**. Una Red ER está libre de conflicto estático si para cualesquiera dos transiciones t_1 y t_2 distintas se cumple que $\alpha(t_1) \cap \alpha(t_2) = \varnothing$, es decir, las transiciones no comparten ningún lugar de entrada. Las transiciones t_1 y t_2 están libre de conflicto dinámico si y sólo si para cada estado alcanzable por la Red, no existen dos tuplas sensibilizadoras de t_1 y t_2 tales que el disparo de una de ellas desensibiliza a la otra, es decir, si las tuplas sensibilizadoras de t_1 y t_2 no comparten ningún token en común. Evidentemente, el conflicto dinámico implica necesariamente un conflicto estático, pero no es cierta la implicación en sentido contrario, puesto que es necesario que el token en común valide el predicado de ambas transiciones.

Los conceptos expuestos asi como la ejecución de las Redes ER se ilustran mediante la Red de la figura 1.7. Los tokens contiene dos identificadores de tipo numérico *a* y *b*. Las acciones ligadas a cada transición se muestran en la parte superior.

En la figura 1.7(a)-1.7(e) se muestran diferentes estados de la ejecución de la Red de Petri. Junto a cada token se indica los valores de sus identificadores *a* y *b*.

En el estado mostrado en la figura 1.7(a) las transiciones t_1 y t_2 se encuentran sensibilizadas, puesto que sus lugares de entrada contienen un token y no se impone el cumplimiento de ninguna condición a ambos tokens. El resultado del disparo de ambas transiciones se muestra en la figura 1.7(b). Los tokens de los lugares de entrada han sido retirados, y cuatro nuevos tokens han sido generados en los lugares de salida según las condiciones impuestas en el resultado de las acciones ligadas a cada transición disparada. En este nuevo estado, t_1 y t_2 vuelven a estar sensibilizadas. Además la transición t_3 tiene un token en su lugar de entrada L_3 que satisface su predicado, luego también la transición t_3 está sensibilizada. Aunque existe un token en cada uno de los lugares de entrada de t_4, ninguno de ellos cumple el predicado de la transición, luego t_4 no puede

token = {<a>,<b>}

t_1={<L_1,<L_1,L_3>>| → L'_1.a=L_3.a=L_1.a y L'_1.b=L_3.b=L_1.b+1}
t_2={<L_2,L_4>| → L'_2.a=L_4.a=L_2.a+1 y L'_2.b=L_4.b=L_2.b}
t_3={<L_3,L_5>| L_3.b=1 → L_5.a=L_3.a y L_5.b=0}
t_4={<<L_3,L_4>,L_6>| L_3.b>1 y L_4.a>1 → L_6.a=L_4.a y L_6.b=L_3.b}

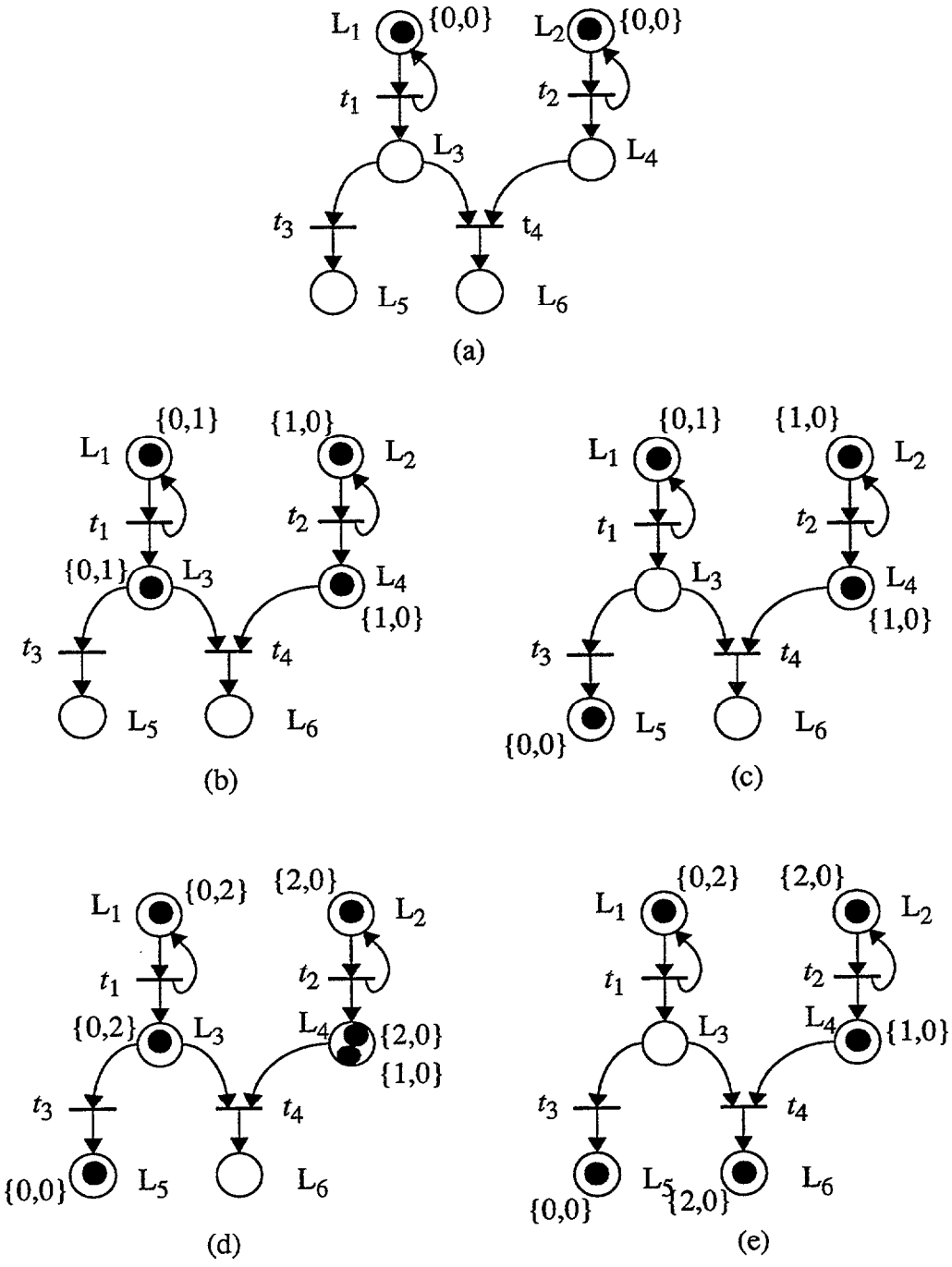


Figura 1.7. Diferentes estados de ejecución de una Red de Petri de alto nivel

considerarse sensibilizada.

El nuevo estado después del disparo de la transición t_3 se muestra en la figura 1.7(c). En la figura 1.7(d) se muestra el resultado después del disparo de las otras dos transiciones sensibilizadas t_1 y t_2. En este nuevo estado, además de las transiciones t_1 y t_2 sólo la transición t_4 cumple las condiciones de su predicado. El resultado del disparo de la transición t_4 se indica en la figura 1.7(e).

Las transiciones t_3 y t_4 se encuentran en un conflicto estático y no dinámico, puesto que los conjuntos de valores de a y b que validan el predicado de ambas transiciones son disjuntos. También aparece un token muerto en L_4 con valores de a=1 y b=0. De la ejecución de la Red se observa que es 1-limitada débil.

1.1.4. Redes de Petri temporizadas

En este punto se expone el método propuesto en [GHE 91] para introducir el concepto de tiempo en las Redes ER.

Cada token contiene una variable numérica llamada **chronos** representando el tiempo en el que el token fue creado.

Las acciones asociadas con las transiciones son las responsables de determinar el valor de la variable chronos de los tokens generados en los lugares de salida, en función de los valores de los identificadores de los tokens sensibilizadores. Se considera que el disparo de una transición no consume tiempo.

A diferencia del resto de las variables que pudiera contener un token, la variable chronos debe mantener un comportamiento similar al concepto tiempo que todos conocemos, por lo que las acciones deben satisfacer los siguientes axiomas:

Axioma 1: El valor de la variable chronos de un token generado por un disparo no puede ser inferior al valor del chronos de cualquiera de los tokens sensibilizadores.

Axioma 2: El valor de la variable chronos de todos los tokens generados por un disparo de una transición debe ser igual. Este valor recibe el nombre de tiempo de

disparo de la transición y se denota como C(t), siendo t la transición disparada.

Axioma 3: Los tiempos de disparo deben ser crecientes con respecto a su ocurrencia, o dicho de otra forma, las secuencias de disparo deben ser ordenadas en el tiempo.

Una **secuencia de disparos** es una secuencia finita $\{t_1, t_2, t_3, \ldots\ldots, t_n\}$, donde la transición t_1 está sensibilizada en el marcado inicial m_0, y cada transición t_i, con $2 \leq i \leq n$, está sensibilizada en el marcado m_{i-1} y su disparo produce el marcado m_i.

Una **secuencia de disparo es ordenada en el tiempo** si para cualesquiera dos transiciones t_n y t_m, tales que el disparo de t_n ocurrió antes que el disparo de t_m, entonces el tiempo del disparo de t_n es menor o igual que el tiempo del disparo de t_m, $C(t_n) \leq C(t_m)$.

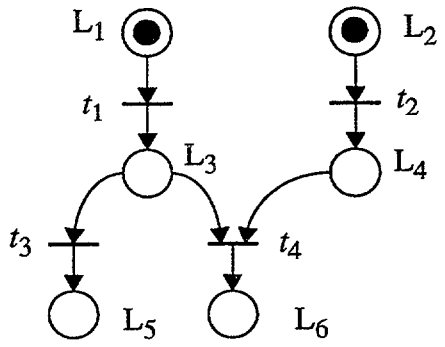
Los axiomas 1 y 2 son de aplicación directa a las condiciones que componen las acciones de cada transición. El axioma 3 trata de imponer un orden total en el conjunto de disparos de la Red de Petri. Este orden no tiene porqué ser único si existen transiciones con tiempos de disparo iguales. En [GHE 91] se demuestra que cada secuencia de disparos de una Red ER donde se satisfacen los axiomas 1 y 2 es equivalente a otra secuencia de disparo que satisface el axioma 3.

Dos **secuencias de disparo** son **equivalentes** si conducen al mismo estado final de la Red desde el mismo estado inicial.

Se define una **Red ER temporizada (TER)** como una Red donde todos los tokens contienen la variable chronos, y los axiomas 1 y 2 son satisfechos.

En la figura 1.8 se muestra una Red de Petri temporizada junto con las acciones ligadas a sus transiciones y su marcado inicial.

Los tokens sólo contienen como variable a chronos. Inicialmente dos tokens con valor de chronos cero se encuentran en L_1 y L_2. Las transiciones t_1 y t_2 se encuentran sensibilizadas con $C(t_1)=1$ y $C(t_2)=5$. Para satisfacer el axioma 3, la transición t_1 será la primera en dispararse creando un nuevo token en L_3 con un valor de chronos igual a 1. Ahora, las transiciones t_2 y t_3 están sensibilizadas con $C(t_2)=5$ y $C(t_3)=10$. Por lo tanto el siguiente paso será disparar t_2. El nuevo estado resultante consiste en un token en L_3



$t_1 = \{<L_1,L_3>| \rightarrow L_3.chronos = L_1.chronos + 1\}$
$t_2 = \{<L_2,L_4>| \rightarrow L_4.chronos = L_2.chronos + 5\}$
$t_3 = \{<L_3,L_5>| \rightarrow L_5.chronos = L_3.chronos + 9\}$
$t_4 = \{<<L_3,L_4>,L_6>| \rightarrow L_6.chronos = max(L_3.chronos,L_4.chronos) + 6\}$
Marcado inicial
$L_1.chronos = 0; L_2.chronos = 0$

Figura 1.8. Red de Petri temporizada

con chronos igual a 1 y otro token en L_4 con un chronos igual a 5. Los pasos siguientes serán analizados en el siguiente punto.

Como se indicó anteriormente, la acción de cada transición debe contener una condición para determinar el tiempo de disparo de ésta. Esta condición se denomina **condición temporal** y generalmente se compone de dos elementos: las condiciones a evaluar para obtener el chronos de sensibilización de la transición S(t) y la duración de la transición [GHE 89], de manera que:

$$C(t) = S(t) + \text{duracion de } t \qquad (1.1)$$

Se define el **chronos de sensibilización** S(t) de una transición *t* como el valor de chronos a partir del cual *t* se encuentra sensibilizada.

Para la obtención del chronos de sensibilización de una transición *t* se deben seguir los siguientes pasos:

1.- Identificar el valor de chronos de cada token sensibilizador de la transición *t*, denotado como $S_L(t)$ donde el subíndice indica el lugar de entrada de *t* donde

reside el token sensibilizador.

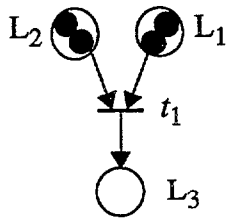
Para obtenerlo se establece una **condición de lugar** para cada lugar perteneciente a $\alpha(t)$ con objeto de determinar el token sensibilizador de dicho lugar. La condición del lugar puede ser: "mínimo valor de chronos de entre los tokens del lugar" o "máximo valor de chronos de entre los tokens del lugar", de aquí en adelante referenciadas como "mínimo del lugar" y "máximo del lugar" respectivamente.

2.- Obtener el chronos de sensibilización S(t) de la transición *t*.

El valor de S(t) corresponde al valor máximo de entre todos los chronos de los tokens sensibilizadores seleccionados con las condiciones de lugar. De esta forma se satisface el axioma 1 enunciado anteriormente.

$$S(t) = \max[S_L(t)] \qquad L \in \alpha(t)$$

Las condiciones de lugar definidas para la Red de la figura 1.9 proporcionan los siguientes resultados: $S_{L1}(t_1)=2$ y $S_{L2}(t_1)=1$. El chronos de sensibilización $S(t_1)$ será igual a 2, y el tiempo de disparo $C(t_1)=3$.



t_1={<<L_1,L_2>,L_3>|→ L_3.chronos=max(min(L_2.chronos),max(L_1.chronos)) + 1}
Marcado: L_1.chronos = 0 y 2; L_2.chronos =1 y 5

Figura 1.9. Determinación del chronos de sensibilización

Como se observa, las condiciones de lugar seleccionan los token que forman la tupla sensibilizadora. Debe recordarse que la evaluación de la condición de lugar se realiza exclusivamente sobre los tokens que satisfacen el resto de condiciones no temporales del predicado de la transición.

Si un lugar es seguro no se necesita establecer una condición de lugar, pues la obtención del token sensibilizador es inmediata. Las transiciones de la figura 1.8 no establecen condiciones de lugar al ser una Red de Petri segura.

1.1.4.1. Modelo de tiempo fuerte y débil

En una Red de Petri convencional, una transición sensibilizada puede dispararse aunque no está obligada a ello.

Esta característica es perfectamente trasladable a las Redes de Petri temporizadas anteriormente vistas. En algunos casos, sin embargo, la simulación de un modelo puede requerir que si una transición se encuentra sensibilizada deba ser disparada.

Una **secuencia de disparo es fuerte** si está ordenada en el tiempo y además, en todos los marcados producidos por la secuencia de disparo la transición disparada ha sido siempre la de menor tiempo de disparo de entre todas las transiciones sensibilizadas. De esta manera se obliga a que una transición sensibilizada sea disparada antes de que el disparo de cualquier otra transición con un tiempo de disparo mayor pueda llegar a desensibilizarla, evitando su disparo.

Se sustituirá el axioma 3 enunciado en el punto 1.1.4. por el siguiente axioma 3' siempre que se desee un modelo de tiempo donde las transiciones sensibilizadas sean forzadas a dispararse.

Axioma 3': Todas las secuencias de disparo deben ser fuertes.

Se denominará modelo de tiempo fuerte aquel en el que las transiciones sensibilizadas son obligadas a dispararse. Por el contrario, el modelo de tiempo débil es aquél que sigue las reglas de la Red de Petri convencional. Es obvio que las ejecuciones derivadas de un modelo de tiempo fuerte constituyen un subconjunto de las ejecuciones de un modelo de tiempo débil.

De esta manera, se conoce una **Red ER temporizada fuerte** (denotada STER) como aquella Red TER que cumple los axiomas 1, 2 y 3'.

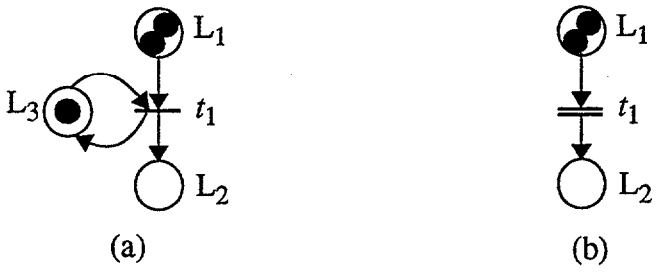
Continuando con la ejecución de la Red de Petri de la figura 1.8 (marcado {0,0,1,1,0,0}), el valor de la variable chronos de los tokens situados en los lugares L_3 y L_4 es 1 y 5 respectivamente. En este estado, las transiciones t_3 y t_4 se encuentran sensibilizadas y en conflicto dinámico. Las dos opciones posibles son:

1- Disparar la transición t_3 en un tiempo de disparo $C(t_3)$ igual a 10. La transición t_4 queda desensibilizada.

2- Disparar la transición t_4 en un tiempo de disparo $C(t_4)$ igual a 11. La transición t_3 queda desensibilizada.

En ambos casos los axiomas 1, 2 y 3 son satisfechos, pero sólo el disparo contemplado en la opción número 1 corresponde a una ejecución de un modelo de tiempo fuerte. Por lo tanto, cuando existan transiciones en conflicto, se disparará la transición con el tiempo de disparo más pequeño al objeto de satisfacer el cumplimiento del axioma 3'.

El cumplimiento del axioma 3' puede dificultar el modelado de sistemas como colas LIFO. En este caso el diseñador del modelo debe buscar la forma para posibilitar que todos los tokens sean disparados. Como ejemplo, en la figura 1.10(a) se muestra el modelo de una cola LIFO, donde el token permaneciendo en el lugar L_3 almacena el instante de tiempo del último disparo con el objetivo de garantizar que éstos sean crecientes en el tiempo.



$t_1 = \{<<L_1,L_3>,L_2> \rightarrow L_2.chronos=L'_3.chronos=max(max(L_1.chronos),L_3.chronos) + 1\}$

Figura 1.10. Modelo de cola LIFO

Mediante este tipo de grafos se modela el uso en exclusiva de algún recurso, de

forma que nadie puede hacer uso del recurso hasta que sea liberado (aparición del token en L_3). Con el objeto de conseguir claridad en los grafos de Redes de Petri, éste tipo de estructuras se representará mediante una transición de doble barra como se indica en la figura 1.10(b), que comprenda la transición y el lugar asociado que almacena el último instante de disparo.

1.2. Paradigma de simulación paralela Chandy-Misra

En una simulación paralela el sistema simulado es dividido en varios subsistemas, cada uno de los cuales consiste de un conjunto de variables de estado independientes. Estos subsistemas son simulados concurrentemente por un conjunto de procesadores que se comunican entre sí a través de canales físicos de comunicación o mediante una memoria compartida.

El rendimiento de una simulación paralela depende de dos factores: el paralelismo inherente en el sistema a simular y del paralelismo que pueda obtenerse del algoritmo de simulación sobre una arquitectura concreta [WON 95].

Los sistemas distribuidos asíncronos constan de varios procesos concurrentes, cuya ejecución deriva en tantas secuencias de eventos como procesos existan en el sistema, donde un evento es un cambio de estado del proceso. En este tipo de sistemas no existe un tiempo y estado global del sistema. Cada proceso tiene conocimiento exacto de su propio estado y la posibilidad de conocer el estado de otro proceso mediante el intercambio de mensajes, conocimiento que no ofrece ningún tipo de garantía debido a los inevitables tiempos de transmisión de las comunicaciones.

En la figura 1.11 se muestra un sistema que se desea simular, compuesto por dos procesos fuente p_1 y p_3 (no reciben mensajes), un proceso servidor p_2 (recibe y envía mensajes) y un proceso terminal p_4 (no envía mensajes).

Las relaciones entre los eventos ocurridos en un sistema se representan mediante un grafo de precedencia de eventos. En la figura 1.12 se muestra un ejemplo de una secuencia de eventos del sistema de la figura 1.11, donde el instante de tiempo real del evento e_i es i.

La aparición de un evento e_j en el sistema como consecuencia de la ejecución de

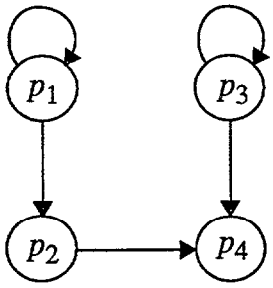


Figura 1.11. Sistema a simular

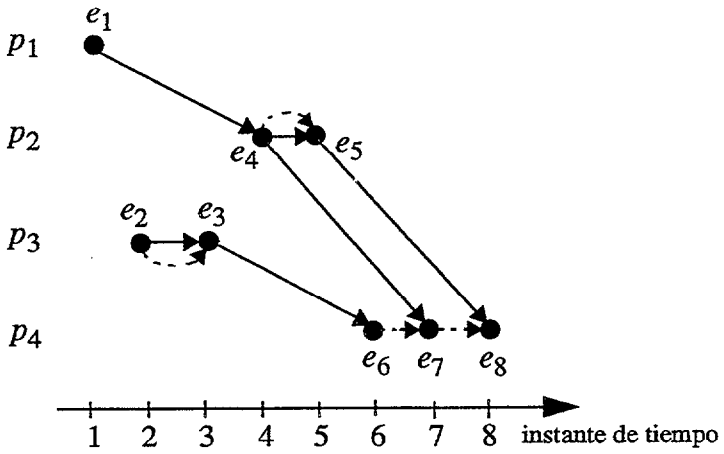


Figura 1.12. Grafo de precedencia de eventos del sistema de la figura 1.11

otro evento e_i se indica mediante una flecha de línea continua desde e_i a e_j en el grafo de precedencia de eventos. Una flecha de línea discontinua dirigida desde el evento e_i al evento e_j significa que ambos eventos deben ejecutarse en el mismo proceso, y e_i debe ejecutarse con anterioridad a e_j. Para simular correctamente el comportamiento del sistema, el evento e_i debe ser procesado antes del evento e_j si existe una flecha (de línea continua o discontinua) dirigida desde e_i a e_j.

En una simulación secuencial los eventos son realizados en una secuencia temporal no decreciente, lo que garantiza que las relaciones indicadas en el grafo de precedencia de eventos no sean violadas.

La ejecución secuencial de los eventos indicados en la figura 1.12 se muestra en la figura 1.13. Se supone que el tiempo de ejecución de los eventos e_1, e_2 y e_3 es 1 y el del resto de eventos es 2.

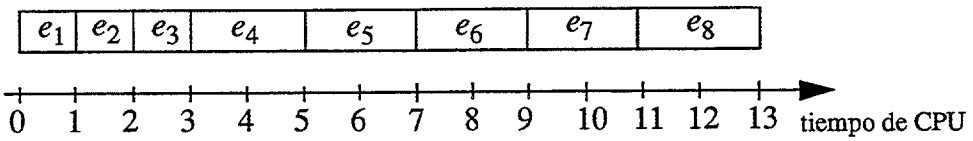


Figura 1.13. Simulación secuencial

En una simulación paralela, el sistema a simular se debe particionar de una forma coherente y cumplir unas reglas de sincronización que garantice unos resultados correctos.

La simulación paralela de un sistema es correcta si y sólo si se mantienen las relaciones entre eventos (representadas mediante flechas en el grafo de precedencia de eventos). Esto implica que todos los eventos de un mismo proceso sean ejecutados en una secuencia temporal no decreciente y se satisfagan las relaciones de causalidad manteniéndolas invariantes [WON 95].

El paradigma de simulación paralela Chandy-Misra [CHA 79] establece un protocolo de simulación que garantiza que la ejecución de los eventos mantenga su orden parcial. El paradigma asume las siguientes premisas:

1 - La comunicación entre procesadores obedece a un esquema FIFO.
2 - La topología del conjunto de procesadores es estática. Se supone que lo procesos se comunican a través de unos canales mediante el envío de mensajes, y que estos canales no cambian.
3 - Un proceso fuente que genere eventos, lo hará siempre siguiendo un esquema FIFO (los eventos serán generados en secuencias temporales monótonamente crecientes).

Se define el concepto **lookahead** $\beta(e)$ de un evento e como el mínimo tiempo de ejecución de un evento. Todos los eventos e' producto de la ejecución del evento e,

serán generados en unos instantes de tiempo

$$C(e') \geq C(e) + \beta(e) \qquad (1.2)$$

siendo C(e) el instante de tiempo correspondiente a la creación del evento e.

El lookahead se corresponde con el intervalo de tiempo en el futuro, dentro del cual un proceso puede predecir su comportamiento.

El paradigma establece las siguientes reglas de espera:

Regla de espera en la entrada: Antes de que un proceso p ejecute un evento e, p debe haber recibido un evento (incluyendo e) por cada uno de sus canales de entrada. Además el evento e debe ser el evento con el instante de tiempo más pequeño de entre los existentes en los canales de entrada.

Regla de espera en la salida: Si el evento e' creado en el instante de tiempo C(e') por el proceso p_i debe ser enviado al proceso p_j para su procesamiento, el envio se realizará una vez que el proceso p_i empiece ha ejecutar un evento e, para el que se cumple la condición

$$C(e') \leq C(e) + \beta(e) \qquad (1.3)$$

Si varios eventos cumplen la expresión 1.3, serán enviados en instantes de tiempo no decrecientes.

La regla de espera en la salida viene motivada por la posibilidad de que el valor del lookahead sea variable, lo que puede provocar que el procesamiento de un evento e genere un mensaje con un instante de tiempo menor que el mensaje producto del procesamiento del evento anterior a e.

La regla de espera en la salida no es necesaria para procesos fuente pues la tercera premisa del protocolo asegura que la secuencia de envíos es monótonamente creciente. Asignando el valor infinito al lookahead de los eventos generados por un proceso fuente conseguimos que siempre se satisfaga la regla de espera en la salida (este valor no significa que el tiempo de ejecución de un evento sea infinito).

La finalización de la simulación por un proceso es indicada mediante el mensaje de fin de simulación (mensaje *eos*). Tienen un lookahead de valor ∞ y un instante de tiempo ∞.

Cuando el proceso fuente ejecuta el último evento envía un mensaje *eos*. Un proceso que reciba este tipo de mensaje, lo relanza por sus canales de salida. Cuando el proceso haya recibido un mensaje *eos* por cada canal de entrada pasa a un estado de inactividad. Un proceso sin canales de salida se vuelve inactivo una vez reciba por cada uno de sus canales de entrada un mensaje *eos*.

En la figura 1.14 se muestra la ejecución de la simulación paralela del sistema de la figura 1.11 sobre 4 procesadores interconectados de igual forma que el sistema que se pretende simular. Cada proceso se asocia a un procesador distinto, y esta relación permanece inalterable durante toda la simulación. Se supone que todos los eventos excepto los generados por los procesos fuente tienen un lookahead de valor 2. Con flechas se señala el momento de envío y recepción de mensajes. Acompañando a cada flecha se indican los mensajes transmitidos ordenados cronológicamente.

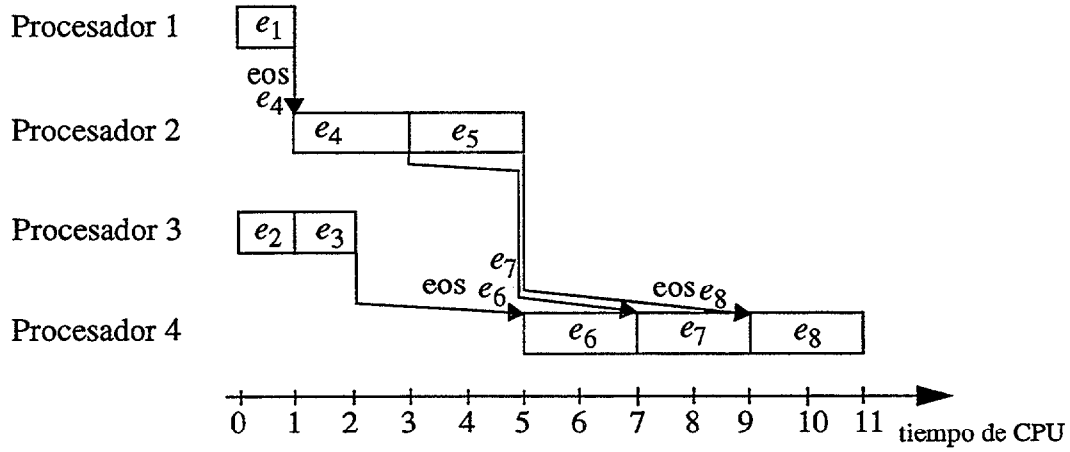


Figura 1.14. Simulación paralela sobre 4 procesadores

La simulación comienza con la realización de los eventos e_1 y e_2 concurrentemente por los procesos p_1 y p_3 respectivamente (ambos son procesos fuente). Una vez finalizado e_1, p_1 envía el evento e_3 al proceso p_2 y a continuación un mensaje *eos*. A continuación de e_2 comienza la ejecución de e_5. El procesador p_2 sólo tiene un canal de entrada y el evento recibido cumple la regla de espera en la entrada, dando inicio el pro-

ceso de e_3.

Al finalizar e_5, el proceso p_3 envía a p_4 el evento e_6 seguido de un mensaje *eos*. El procesador p_4 no puede comenzar a trabajar hasta que no se cumpla la regla de espera en la entrada. La finalización del evento e_3 conlleva la ejecución de e_4 y el envío del evento e_7.

La regla de espera en la salida obliga a retrasar el envío del evento e_7 al proceso p_4 hasta que p_2 no comience la ejecución de un evento que cumpla la condición 1.3. Finalizada la ejecución de e_4, se evalúa el cumplimiento de la regla de espera en la salida para el envío del evento e_8. La recepción del mensaje *eos* por el proceso p_2 asegura la no aparición de eventos con instantes de tiempo menores a los que están esperando ser enviados, luego los eventos e_7, e_8 y el mensaje *eos* se envían a p_4, pasando el proceso p_2 a un estado de inactividad. En este momento p_4 cumple la regla de espera en la entrada y procesa el evento con el menor instante de tiempo de entre los presentes en sus canales de entrada, e_6. A continuación se procesan los eventos e_7, e_8 y el procesador p_4 se vuelve inactivo.

Es evidente la mejora obtenida respecto al tiempo de ejecución secuencial de la simulación del sistema. En la simulación secuencial el sistema finalizaba su ejecución en el instante 13 mientras que la simulación paralela lo hace en el instante 11.

El algoritmo de Chandy-Misra es también aplicable cuando el número de procesadores M es inferior al número de procesos P, $M \leq P$.

Para mantener la topología de la red estática, se realiza una asignación inicial de los procesos del sistema a los diferentes procesadores. Si P_k es el conjunto de procesos asignados al procesador k se cumple

$$\bigcap_{k=1..M} P_k = \varnothing \qquad (1.4)$$

Esta asignación permanece inalterable durante la realización de la simulación.

En la figura 1.15(b) se muestra la ejecución de la simulación paralela para el caso de dos procesadores interconectados como indica la figura 1.15(a). Al procesador 1 le han sido asignados los procesos $P_1=\{p_1, p_2\}$ y al procesador número 2 los procesos $P_2=\{p_3, p_4\}$.

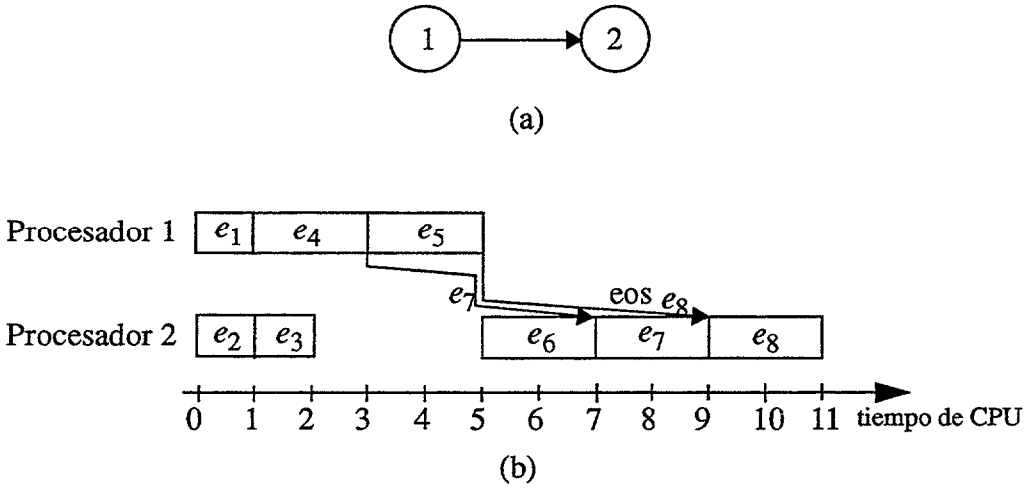


Figura 1.15. Simulación paralela sobre 2 procesadores

Aunque el empleo de dos procesadores conduzca en este caso a un tiempo de ejecución de la simulación similar al de cuatro procesadores, el número de recursos hardware necesarios ha disminuido aumentando de este modo su eficiencia.

El cumplimiento de las reglas de espera establecidas en el paradigma puede conducir a situaciones de bloqueo como la mostrada en la figura 1.16. La simulación de la red comienza con el envío del evento e desde el proceso p_1 al proceso p_2. El proceso p_2 no puede ejecutar e hasta que no reciba otro evento del proceso p_5 (regla de espera en la entrada). El proceso p_5 no generará ningún evento hasta que reciba un evento por su canal de entrada, un evento que en definitiva tiene que ser enviado por el proceso p_2.

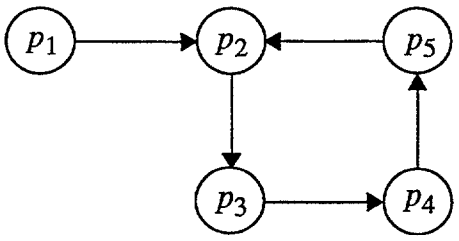


Figura 1.16. Red con bloqueo

Para evitar el bloqueo o rearrancar la simulación se emplean los métodos que se

comentan en los siguientes puntos.

1.2.1. Envío de mensajes nulos

Para evitar el bloqueo en las Redes se emplean los denominados mensajes nulos [CHA 79] que llevan información referente a los instantes de tiempo. Supongamos que los procesos de la figura 1.16 tienen un lookahead β constante y el reloj local de cada uno de ellos se inicia al valor cero. Al comienzo de la simulación, p_1 envía un mensaje e al proceso p_2 con un instante de tiempo C(e). El proceso p_2 no puede ejecutar ningún evento hasta no recibir un mensaje de p_5.

El proceso p_2 envía un mensaje nulo a p_3 con un instante de tiempo $0+\beta$ (valor de su reloj local más el lookahead). La recepción del mensaje nulo en el proceso p_3 le garantiza que no recibirá ningún otro mensaje de p_2 con un instante de tiempo menor que el indicado en el mensaje y el reloj local de p_3 aumenta a β. El proceso p_3 envía un mensaje nulo a p_4 con un instante de tiempo $\beta+\beta$, y el reloj local de p_4 es incrementado al valor 2β. El resto de procesos procederán de igual forma, y así el proceso p_2 recibirá de p_5 un mensaje nulo con un instante de tiempo 4β, que le garantiza que p_5 no enviará ningún mensaje con un instante de tiempo menor que 4β.

Si los mensajes nulos recorren varias veces la red, llegará un momento en el que el proceso p_2 recibirá un mensaje nulo con un instante de tiempo mayor que el instante de tiempo de e, C(e), lo que le asegura que no recibirá ningún mensaje con un instante de tiempo menor que el indicado en el mensaje nulo. En este momento no tiene ningún sentido que el proceso p_2 siga esperando por un evento de entrada de p_5, por lo que procede a actualizar su reloj local al valor C(e) y ejecutar el evento e.

Simulaciones llevadas a cabo con este esquema han puesto en evidencia una gran cantidad de mensajes nulos transmitidos, lo que deriva en un procesamiento superior al que exige la simulación al tener que procesar varios mensajes nulos por cada mensaje de información enviado por otro proceso. Posibles mejoras a este esquema de trabajo pueden encontrarse en [MIS 86].

1.2.2. Secuencia de computaciones paralelas

Como alternativa al anterior, se encuentra el llamado secuencia de computaciones paralelas [CHA 81] donde no se emplean los mensajes nulos. Comprende una secuencia de fases de simulación distribuida separadas por una fase denominada interfaz. En la figura 1.17 se muestra esta secuencia.

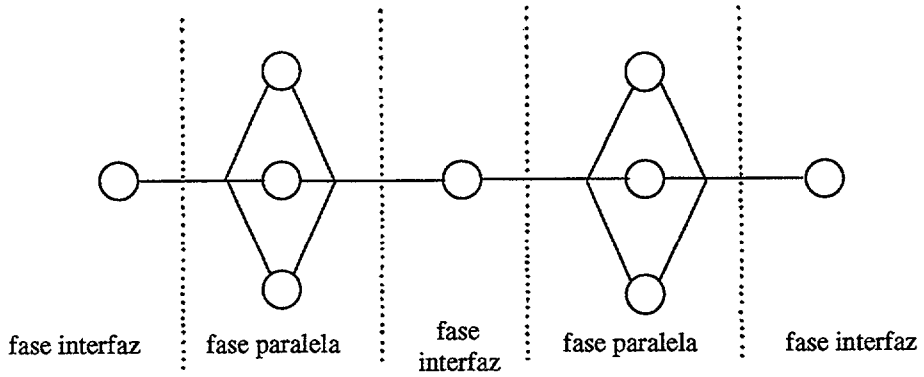


Figura 1.17. Secuencia de computaciones paralelas

El algoritmo trabaja de acuerdo a la siguiente secuencia de computaciones:

- Fase paralela: ejecución de la simulación siguiendo el esquema de simulación paralela Chandy-Misra explicado anteriormente, hasta que se detecta un bloqueo.

- Fase de interfaz: se rompe el bloqueo.

Es necesario un proceso llamado controlador que sincronice estas acciones. Su única función es detectar la terminación de una fase e iniciar la siguiente, sin llevar a cabo ninguna computación.

Una vez el sistema simulado se encuentra bloqueado, el controlador ordena a cada uno de los procesos comenzar la fase de interfaz. En esta nueva fase se implementa un mecanismo para desbloquear el sistema. Todos los procesos cooperan para localizar los eventos con el instante de tiempo más pequeño. Estos eventos serán ejecutados desbloqueando el sistema.

Cada proceso p_i calcula el tiempo del siguiente mensaje a ser transmitido por el proceso p_i al proceso p_j suponiendo que el proceso p_i no va a recibir más mensajes. A este tiempo se le denotará como U_{ij}. Suponiendo que U_{kr} es el mínimo de todos los U_{ij} calculados, el proceso p_k no recibirá ningún otro mensaje con un instante de tiempo menor que U_{kr}, y por lo tanto el proceso p_k podría enviar su mensaje con un instante de tiempo U_{kr}, rompiendo de esta manera el bloqueo.

Si el proceso p_i no tiene ningún mensaje que transmitir a p_j, U_{ij} tomará el valor infinito.

El valor de U_{ij} puede ser mejorado considerando los mensajes que van a ser transmitidos por los procesos que tengan un camino dirigido al proceso p_i. El nuevo valor se denota como W_{ij} y se calcula del siguiente modo:

$$W_{ij} = \begin{cases} t^* & \text{si } p_i \text{ tiene un mensaje con un instante de tiempo } t^* \text{ para } p_j \\ \max\left(t_{ij}, \min[U_{ij}, \min_r\{W_{ri}\}]\right) & \text{en otro caso} \end{cases} \quad (1.5)$$

donde t_{ij} es el instante de tiempo del último mensaje enviado por el proceso p_i al proceso p_j.

El cálculo de W_{ij} se lleva a cabo de forma distribuida mediante n ciclos de computación, donde n es el número de procesos.

El objetivo del método es encontrar el mayor número de procesos que puedan ser iniciados para romper el bloqueo, garantizando que si el proceso p_m es iniciado para ejecutar un evento en un instante de tiempo t^*, todos los mensajes recibidos por p_m a continuación deben tener un instante de tiempo mayor que t^*, de manera que no afecten al evento ejecutado por p_m.

El proceso p_i no recibirá ningún mensaje del proceso p_r con un instante de tiempo menor que W_{ri}, ni enviará ningún mensaje con un instante de tiempo menor que W_{ij}. Por lo tanto, el reloj interno del proceso p_i y de sus canales de comunicación será incrementado para reflejar este hecho, pasando a evaluar la regla de espera en la entrada o salida correspondiente a cada canal.

Si el conjunto de canales sobre los que se evalúan las reglas es diferente del

existente en el momento del bloqueo, el proceso es reanudable (puede continuar). Una vez calculado el valor de W, cada proceso informa al controlador si es reanudable o no, pasando a la fase de simulación paralela. En [CHA 81] se demuestra que siempre existirá al menos un proceso reanudable y al menos un mensaje será transmitido antes de volver a entrar el sistema en bloqueo.

1.3. Estado del arte de los algoritmos de distribución de procesos

La reciente aparición de sistemas distribuidos ha introducido la necesidad de desarrollar e incorporar técnicas de gestión de los procesos concurrentes, con el objeto de obtener un nivel de rendimiento del sistema proporcional a la cantidad de recursos de los que se dispone. El problema es distribuir los procesos o tareas entre los diferentes procesadores de forma que se minimize o maximize un parámetro, como el tiempo de ejecución, los retardos de comunicaciones, la utilización de los recursos u otros. Al parámetro mencionado se le denomina función objetivo.

1.3.1 Clasificación de los algoritmos de distribución de procesos

El estudio del problema de la distribución de procesos ha derivado en un gran número de metodologías [CAS 88] mostradas en la figura 1.18.

La distribución de procesos conocida como local consiste en la asignación de los procesos sobre las rodajas de tiempo de un procesador. Por otra parte, la distribución global comprende la decisión de dónde ejecutar un proceso en un sistema multiprocesador.

Dentro de la distribución global cabe distinguir entre distribución estática o dinámica.

En la distribución estática [SAR 86][MA 82], la asignación o distribución de los procesos entre los procesadores se realiza con anterioridad a la ejecución del programa, generalmente en la compilación. Su principal ventaja es que su tiempo de ejecución (overhead) no incide sobre el tiempo de ejecución del programa. Como desventajas se pueden enumerar:

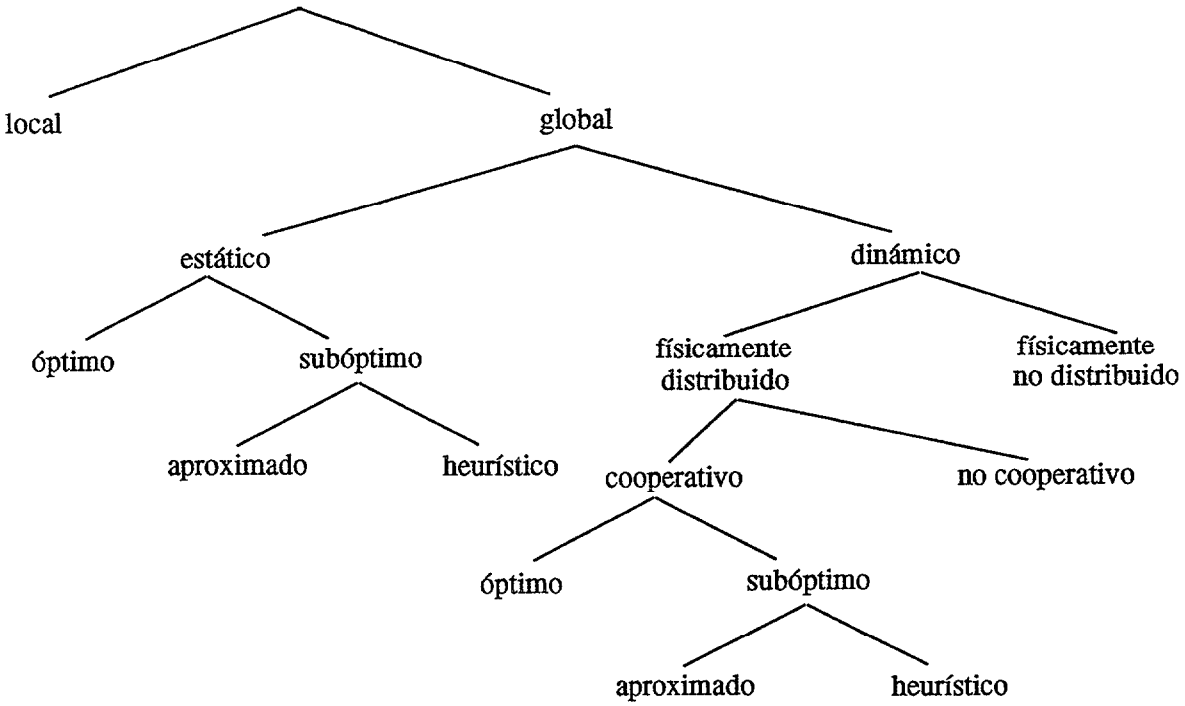


Figura 1.18. Clasificación de métodos de distribución de procesos

- Se necesitan métodos fiables que permitan estimar el tiempo de ejecución y/o los costes de comunicación de cada proceso. Esta estimación es difícil debido a los bucles, condiciones, ... presentes en cualquier proceso.

- La topología del sistema debe permanecer invariante.

La distribución dinámica de los procesos se realiza durante la ejecución del programa. La técnica más conocida es la de balanceo/compartición de carga [WAN 85][EAG 86][SHI 92][ZHU 95][WIL 93], consistente en transferir tareas desde procesadores con mucha carga hacia procesadores con poca carga de forma que se igualen las cargas de los procesadores del sistema distribuido. Esta técnica puede ser con expropiación o sin expropiación dependiendo de la posibilidad de transferir una tarea después de que haya comenzado su ejecución en un procesador.

El método de balanceo/compartición de carga cuenta con tres criterios:

- Criterio de información: determina la cantidad de información referente a un proceso que debe estar disponible al recurso encargado de realizar la distribución.
- Criterio de transferencia: determina las condiciones necesarias para que un proceso sea transferido.
- Criterio de asignación: determina el procesador hacia el cual el proceso será transferido.

La ventaja de la distribución dinámica de procesos es la flexibilidad. Su principal desventaja es su overhead debido a las operaciones de transferencia de procesos, que incide directamente en el tiempo de ejecución global del programa.

Las distribuciones de procesos pueden ser subdivididas en óptimas o subóptimas, dependiendo de la posibilidad de obtener una solución óptima de acuerdo con nuestra función objetivo.

Ante la imposibilidad de obtener una solución óptima con un coste computacional razonable, se utilizan métodos que proporcionen soluciones cercanas a la óptima. Según el método se distinguen entre distribuciones subóptimas aproximadas y distribuciones subóptimas heurísticas.

Las distribuciones subóptimas aproximadas intentan obtener una solución aceptable sin rastrear la totalidad del espacio de soluciones (lo que conllevaría un tiempo de ejecución muy elevado). Se utilizan criterios para reducir el espacio de soluciones hasta una solución "buena". Con este fin se necesita una métrica que permita evaluar cuan buena es una solución para decidir la finalización de la búsqueda.

Como alternativa a la distribución subóptima aproximada, el método heurístico [SHI 90][LO 88][EFE 82] se presenta como más sencillo y rápido. Utiliza algún criterio que comprende un parámetro especial, fácil de calcular y monitorizar, para realizar la distribución de los procesos.

La distinción entre algoritmos distribuidos y no distribuidos atiende a la localización de la autoridad que toma las decisiones. Esta responsabilidad puede estar localizada en un sólo procesador o repartida entre los procesadores.

Otra característica diferenciadora de los métodos de distribución es el grado de autonomía que tiene cada procesador para la toma de decisiones relativas a uno de sus recursos. En la distribución cooperativa cada procesador toma en consideración infor-

mación facilitada por otros procesadores antes de tomar una decisión.

La clasificación mostrada en la figura 1.18 pretende abarcar todas las posibilidades existentes para el problema de la distribución de procesos. Sin embargo, pueden existir características de nuestro problema que conduzcan a una política de distribución que difiera de las enumeradas o comprenda características de diferentes tipos de distribuciones.

1.3.2 Resultados del peor algoritmo de distribución

Cada algoritmo de distribución da lugar a diferentes tiempos de ejecución que inciden directamente sobre el tiempo de ejecución del programa global.

En [SAR 89] se demuestra que el tiempo de ejecución obtenido con un determinado algoritmo de distribución está acotado mediante la expresión 1.6.

$$\max(T_{crit}, T_{total}/P) \leq T_{par}(P) \leq (T_{crit} \cdot (P-1))/P + T_{total}/P \qquad (1.6)$$

donde

P: es el número de procesadores del sistema multiprocesador.

T_{total}: es el tiempo de ejecución total de los procesos que se desea distribuir.

$T_{par}(P)$: es el tiempo de ejecución de la totalidad de los procesos sobre un sistema multiprocesador compuesto por P procesadores. El valor de $T_{par}(P)$ depende del algoritmo de distribución empleado.

T_{crit}: es el tiempo de ejecución de la totalidad de los procesos sobre un número infinito de procesadores. Conforme el número de procesadores tienda a infinito, $T_{par}(P)$ se aproximará a T_{crit}.

La expresión 1.6 es válida siempre que no se fuercen tiempos muertos en los procesadores, es decir, no puede haber procesadores inactivos si existen procesos pendientes de realización.

La obtención de la distribución óptima es un problema NP-completo, y de la expresión 1.6 se puede derivar el caso más pesimista de $T_{par}(P)$.

Para ello se parte de las siguientes expresiones.

$$T_{par}(P) \geq T_{crit} \tag{1.7}$$

$$T_{par}(P) \geq T_{total}/P \tag{1.8}$$

Utilizando las expresiones 1.7 y 1.8 en el límite superior de $T_{par}(P)$ de la expresión 1.6 se obtienen las expresiones 1.9 y 1.10.

$$T_{par}(P) \leq (1 + (P-1)/P) \cdot T_{par}(P)^{opt} \tag{1.9}$$

$$T_{par}(P) \leq (2 - 1/P) \cdot T_{par}(P)^{opt} \tag{1.10}$$

La expresión 1.10 muestra que la variación de $T_{par}(P)$ con diferentes algoritmos de distribución es menor que $2*T_{par}(P)^{opt}$, donde $T_{par}(P)^{opt}$ es el valor de $T_{par}(P)$ cuando se utiliza una distribución de tareas óptima.

1.4. Tiempo virtual en sistemas distribuidos

Un sistema distribuido es un sistema formado por un conjunto de procesos espacialmente separados. Pueden ser clasificados en sistemas fuertemente acoplados y débilmente acoplados. Los procesos en un sistema fuertemente acoplado comunican a través de una memoria compartida. En un sistema débilmente acoplado, los procesos comunican intercambiando mensajes con unos tiempos de transmisión impredecibles (pero distintos de cero) [MAT 89].

En los sistemas distribuidos es muy difícil obtener un tiempo global y un estado global. En [LAM 78] se define el concepto de tiempo virtual y se implementa mediante unos relojes lógicos. Diversos autores [MOR 85] y [CHA 85] describen algoritmos para obtener la mejor aproximación del estado global del sistema haciendo uso del tiempo virtual.

Un proceso consiste en una secuencia de eventos, donde un evento es un cambio de estado del proceso de duración cero.

Los eventos se encuentran relacionados de forma que los eventos de un mismo proceso están ordenados por su orden de ocurrencia local, y cada evento de recepción tiene su correspondiente evento de envío (el contrario no tiene porqué ser cierto, pues puede darse la circunstancia de detener el sistema en un estado, donde un evento de envío no tenga su correspondiente evento de recepción, debido a los tiempos de transmisión). Esta relación entre los eventos de un sistema distribuido permite definir un orden parcial en el sistema. La relación mencionada se denomina relación de causalidad.

La relación de causalidad sobre un conjunto de eventos de un sistema, denotada mediante el símbolo "=>", es la más pequeña relación que satisface las siguientes condiciones:

1.- Si a y b son eventos en el mismo proceso y la ocurrencia de a precede a la de b, entonces $a => b$.
2.- Si a es un evento de envío de un mensaje, y b es el correspondiente evento de recepción del mensaje, entonces $a => b$.
3.- Si $a => b$ y $b => c$, entonces $a => c$.

Suponiendo que $\neg(a => a)$, la relación de causalidad impone un orden parcial irreflexivo sobre el conjunto de los eventos de un sistema, de manera que $a => b$ implica que la realización del evento a debe preceder a la de b.

Una manera gráfica de representar el orden parcial de los eventos es mediante un grafo de precedencia de eventos.

En ocasiones es imposible determinar qué evento fue el primero en realizarse, y por lo tanto no se puede asegurar un orden parcial entre ambos eventos. En [LAM 78] se utiliza el concepto de tiempo virtual para resolver las situaciones anteriores.

La implementación del tiempo virtual se lleva a cabo asociando un reloj lógico C_i a cada proceso P_i del sistema distribuido. Este reloj lógico asignará un número $C_i(a)$ a cada evento a realizado por el proceso P_i, donde el número $C_i(a)$ simboliza el instante de tiempo de realización del evento a.

A diferencia del tiempo real que avanza de por sí, el tiempo virtual es de naturaleza discreta y avanza con la sucesión de eventos. Si en un proceso no se hace nada, el tiempo real avanzará, mientras que el reloj lógico se mantendrá inmóvil a la espera de la

ocurrencia del siguiente evento.

Es obvio que si un evento *a* ocurre antes que otro evento *b*, entonces *a* ocurrió en un instante de tiempo anterior al instante correspondiente al evento *b*. Más formalmente esta condición es conocida como condición de reloj, y establece que para cualesquiera dos eventos *a* y *b* tal que $a \Rightarrow b$, se cumple que $C(a) < C(b)$.

Mediante el sistema de relojes lógicos implementado se puede establecer un orden total de los eventos de un sistema denotado mediante el símbolo $\overset{t}{\Rightarrow}$, simplemente ordenando los eventos según los instantes de tiempo en los que sucedieron. Para establecer un orden en el caso de que varios eventos tengan el mismo tiempo de ocurrencia se define un orden arbitrario entre los procesos, de manera que

$$a \overset{t}{\Rightarrow} b \text{ si y sólo si } C_i(a) < C_j(b) \text{ o bien } C_i(a) = C_j(b) \text{ y } P_i < P_j$$

El orden total no es único y depende tanto del sistema de relojes elegido como del orden parcial definido sobre los procesos.

El orden total de los eventos de un sistema distribuido puede distorsionar de alguna manera la realidad en tanto en cuanto se supone existe una secuencialidad en la realización de los eventos. Esta secuencialidad implica la pérdida del concepto de simultaneidad, pues eventos que se realizan simultáneamente se les asigna un orden parcial, perdiendo información que en algunos casos podría ser muy valiosa.

Dos eventos *a* y *b* son mutuamente independientes si y sólo si $\neg(a \Rightarrow b)$ y $\neg(b \Rightarrow a)$, pudiendo ser sus tiempos lógicos tanto iguales como diferentes.

Con el tiempo asociado a cada evento se obtiene una serie de propiedades dadas a continuación en forma de lemas.

Lema 1: Sean *a* y *b* dos eventos cualesquiera con sus instantes de tiempo $C(a)$ y $C(b)$ tal que $C(a) = C(b)$, entonces se cumple que $\neg(b \Rightarrow a)$ y $\neg(a \Rightarrow b)$, siendo por tanto los eventos a y b mutuamente independientes.

Lema 2: Sean a y b dos eventos cualesquiera con sus instantes de tiempo $C(a)$ y $C(b)$ que cumplen $C(a) < C(b)$, entonces $\neg(b \Rightarrow a)$.

El enunciado del lema 2 es obvio, pues la realización de un evento en el futuro nunca puede afectar a un evento actual o pasado. Por contra, en estas condiciones no se

puede asegurar si los eventos *a* y *b* están relacionados o no, es decir, si *a* => *b* o los eventos *a* y *b* son mutuamente independientes.

1.5. Trabajos relacionados

En [PUE 93a], [PUE 93b] y [DUE 94] se muestra un algoritmo para la simulación distribuida conservativa de Redes de Petri temporizadas sobre un sistema multiprocesador. La Red de Petri es particionada en subredes más pequeñas y ejecutadas por varios procesadores.

La partición de la Red puede ocasionar que un lugar pertenezca a más de una subred, lo que hace necesario establecer un protocolo denominado de transporte, para resolver los conflictos por recursos compartidos entre subredes.

Además se describe el protocolo de sincronización del tiempo en las distintas subredes y el protocolo de detección de bloqueos para detectar y corregir los bloqueos que pudieran aparecer en la ejecución.

El algoritmo está orientado a conjuntar submodelos realizados por grupos de trabajo diferentes, y cuyos niveles de abstracción y notación pueden ser diferentes, más que a disminuir el tiempo de ejecución de la simulación, aunque implícitamente se obtiene esta disminución.

En [MOR 96] se presenta bajo el nombre de paralelización parcial una solución para disminuir el tiempo de ejecución de programas secuenciales. En el programa secuencial se identifican aquellas porciones de código que son susceptibles de ser realizadas simultáneamente, con el objeto de ejecutarse por diferentes procesadores.

De forma que la ejecución del programa atraviesa momentos donde la ejecución es secuencial (código no paralelizable) y otras donde es paralelo.

Como contrapunto al enorme trabajo que requiere la implementación de una paralelización, esta solución permite obtener una disminución sustancial del tiempo de ejecución del programa secuencial original con muy poco esfuerzo por parte del programador.

Otra de sus ventajas es su adaptabilidad a diferentes arquitecturas multiprocesa-

doras.

En [DUE 94] pueden encontrarse otros trabajos desarrollados para la ejecución de Redes de Petri temporizadas sobre arquitecturas paralelas enfocados desde ideas diferentes a las que en esta tesis se proponen.

1.6. Objetivo de la tesis

Las Redes de Petri de alto nivel temporizadas con una semántica del tiempo fuerte es el formalismo que va a ser utilizado para modelar sistemas de la vida real. Su característica concurrente las hace ideales para trasladar el paralelismo inherente en el sistema a su modelo correspondiente.

El objetivo de la tesis es encontrar un método que permita explotar el paralelismo en la simulación del modelo.

Con este fin, se analizarán las relaciones temporales de los disparos de transiciones programados en cada momento. Este análisis proporcionará los disparos programados que pueden ser realizados simultáneamente, respetando siempre las reglas de causalidad que garantizan unos resultados correctos de la simulación.

La ejecución simultánea de los disparos programados seleccionados se realizará sobre una arquitectura multiprocesadora, siendo necesario un método de distribución que equilibre la carga de los diferentes procesadores cuando el número de procesadores sea menor que la cantidad de disparos a realizar simultáneamente.

Evidentemente, la simulación llevada a cabo con el método planteado redundará en una disminución de su tiempo de ejecución con respecto a los alcanzados con los métodos tradicionales.

1.7. Estructura de la memoria

En este capítulo se ha presentado el estado actual de los tres campos sobre los que se va a desarrollar esta tesis: Redes de Petri, simulación paralela y algoritmos de distribución.

La exposición del tiempo virtual nos permite introducir los conceptos necesarios para afrontar las ideas planteadas en los próximos capítulos.

En el capítulo 2 se define el concepto de distancia temporal en Redes de Petri temporizadas. Este concepto se utiliza para obtener las condiciones necesarias para comprobar la existencia de una posible relación de causalidad entre transiciones.

El capítulo 3 describe la solución más inmediata para implementar una simulación paralela aprovechando la concurrencia inherente en las Redes de Petri. Esta primera solución es mejorada mediante un estudio de las relaciones temporales de las transiciones sensibilizadas.

En el capítulo 4 se presentan los métodos de distribución heurísticos como los más idóneos para nuestro algoritmo. Se describen y ensayan dos algoritmos de distribución de procesos para estas simulaciones. El capítulo finaliza describiendo un método para asignar un orden parcial entre los procesos en función de su tiempo de ejecución. Orden necesario establecer para la distribución de este tipo de procesos entre los procesadores.

Finalmente, los algoritmos obtenidos son ensayados en el capítulo 5 sobre un modelo de red local ethernet modelada mediante una Red de Petri. El simulador paralelo es simulado en una máquina monoprocesadora con el objeto de predecir su rendimiento.

Capítulo 2: Distancia temporal y causalidad

2. Distancia temporal y causalidad

2.1. Introducción

El modelo de espacio-tiempo de Minkowski representa la realidad combinando el espacio n-dimensional junto con el tiempo para obtener una estructura n+1-dimensional. En los grafos de precedencia de eventos, nos restringimos al valor n=1 con una dimensión espacial y otra temporal.

El modelo asegura que debido a la velocidad finita de las señales, un evento sólo puede afectar a los eventos que residen en el interior del llamado cono de luz. Cada evento tiene unas coordenadas espacio-tiempo, de manera que conocidas las coordenadas de dos eventos y la velocidad de la señal se puede llegar a conocer de forma sencilla si un evento puede influir en el otro, o por el contrario son mutuamente independientes.

En la figura 2.1 se representa el cono de luz del evento *p* para una estructura bidimensional. Cualquier evento situado fuera del cono de luz (zona no sombreada) no se verá afectado por la realización del evento *p*. El evento *x* puede verse afectado por el evento *p*, pues *x* reside en su cono de luz. En cambio, el evento *y* nunca se verá afectado por el evento *p*, lo que pone de manifiesto la independencia entre ambos eventos. De manera que si simulamos un sistema real con los eventos *p*, *y* y *x*, podemos simular concurrentemente *p* e *y*.

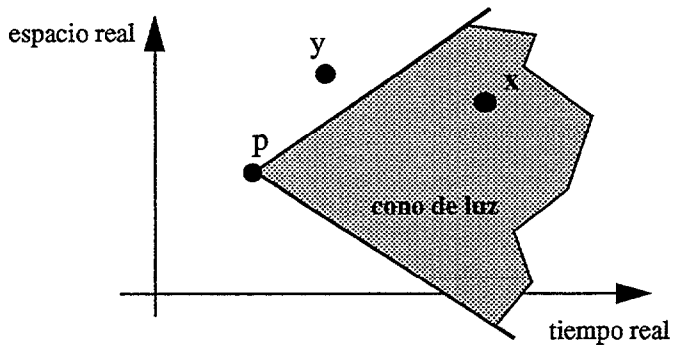


Figura 2.1. Cono de luz

Para la simulación de los eventos se establece una transformación que asigna un número (tiempo virtual) a cada evento, de forma que si a y b son dos eventos tal que $a \Rightarrow b$, se debe cumplir que $C(a) < C(b)$ (condición de reloj).

La transformación mencionada establece un orden lineal sobre el conjunto de eventos a costa de eliminar información vital sobre ellos, como es su coordenada espacial. Mantiene invariantes las relaciones de causalidad pero no preserva las negaciones, que proporcionan información para determinar si dos eventos son mutuamente independientes [MAT 89]. Esta es la causa de que dados dos eventos con sus instantes de tiempo distintos, no se pueda conocer si son independientes o mantienen una relación de causalidad entre ellos.

2.2. Distancia temporal

En el caso de una Red de Petri temporizada donde los eventos son los disparos de las transiciones, todo lo expuesto en el punto anterior es perfectamente aplicable.

Con el objeto de suplir la falta de información referente a las coordenadas espaciales del conjunto de eventos, se expondrá el concepto de distancia temporal que proporcionará la distancia entre dos transiciones de una Red de Petri temporizada. Previamente se definirá el concepto de camino lo que facilitará su exposición.

Se define un **camino** desde el $nodo_1$ al $nodo_n$ de una Red de Petri, como la secuencia de nodos: $nodo_1$ $nodo_2$ $nodo_3$ .... $nodo_n$, de forma que cualquier $nodo_i$ de la secuencia tal que $2 \leq i \leq n-1$, es al mismo tiempo un lugar de entrada para el $nodo_{i+1}$ y un lugar de salida para el $nodo_{i-1}$. Al $nodo_1$ se le considerará el nodo origen del camino, y al $nodo_n$ el nodo destino del camino. En el caso concreto de considerar los nodos origen y destino como transiciones, el camino entre las transiciones t_1 y t_n vendría especificado mediante la secuencia de transiciones y lugares t_1 L_1 t_2 L_2 .... t_n.

Como consecuencia de la "conexión" entre las transiciones t_1 y t_n a través del camino indicado, un disparo de la transición t_1 generaría un token de salida en el lugar L_1, que podría sensibilizar la transición t_2 del camino. Un disparo de la transición t_2 actuará de igual forma, pudiendo sensibilizar a la siguiente transición del camino t_3, y así sucesivamente, de manera que alguno de los token de salida de la penúltima transición disparada perteneciente al camino podría sensibilizar a la transición t_n. En defini-

tiva, el camino entre dos transiciones equivale a la posibilidad de que el disparo de la transición origen conduzca a la sensibilización de la transición destino en algún estado posterior de la Red de Petri. Al token generado en el lugar L_i mediante el proceso descrito anteriormente se le denomina **token inducido** por la transición t_1 en el lugar L_i. Una transición sólo puede generar tokens inducidos en aquellos lugares de la Red hacia los cuales se pueda establecer un camino desde la propia transición.

Se define la **distancia temporal** desde la transición t_1 a la transición t_n de una Red de Petri temporizada, denotada $d(t_1, t_n)$, como la mínima distancia temporal de todos los caminos que enlazan t_1 con t_n.

La distancia temporal de un camino es la suma de la duración de las transiciones incluidas en el camino de t_1 a t_n exceptuando los extremos, las transiciones t_1 y t_n. Si la duración de la transición no es determinista se considera su valor mínimo.

Si no existe un camino entre dos transiciones, se considera que la distancia temporal entre ambas es infinito.

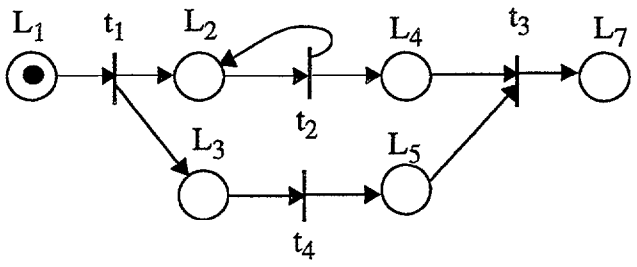
En la Red de Petri de la figura 2.2 existe una infinidad de caminos posibles desde la transición t_1 a t_3:

$t_1\ L_3\ t_4\ L_5\ t_3$
$t_1\ L_2\ t_2\ L_4\ t_3$
$t_1\ L_2\ t_2\ L_2\ t_2\ L_4\ t_3$
$t_1\ L_2\ t_2\ L_2\ t_2\ L_2\ t_2\ L_4\ t_3$
.....
.....
$t_1\ L_2\ t_2\ L_2\ t_2\ L_2\ t_2 \ldots\ldots\ldots L_4\ t_3$

La distancia temporal por el primer camino indicado se corresponde con la duración de la transición t_4, que es 19, mientras que la distancia temporal del segundo camino tiene un valor de 2. El resto de caminos indicados incluyen al camino $t_1\ L_2\ t_2\ L_4\ t_3$, luego su distancia temporal será en consecuencia superior a 2.

La distancia temporal entre la transición t_1 y t_3 es la mínima distancia temporal del conjunto de caminos existentes que unen ambas transiciones, luego $d(t_1, t_3) = 2$.

La distancia temporal definida tiene la siguientes propiedades:



$t_1 = \{<L_1,L_2> \rightarrow L_2.chronos = L_1.chronos + 3\}$
$t_2 = \{<L_2,L_4> \rightarrow L_4.chronos + 2 <= L_2.chronos <= L_4.chronos + 4\}$
$t_3 = \{<<L_4,L_5>,L_7> \rightarrow L_7.chronos = max(L_4.chronos,L_5.chronos) + 5\}$
$t_4 = \{<L_3,L_5> \rightarrow L_5.chronos = L_3.chronos + 19\}$

Figura 2.2. Red de Petri temporizada

- $d(t_i,t_j) \geq 0$
- No es reflexiva $d(t_i,t_j) \neq d(t_j,t_i)$
- Si $t_i = t_j$ entonces $d(t_i, t_j) = 0$

La distancia temporal permite conocer el mínimo valor de la variable chronos de un token inducido por una transición en un lugar cualquiera de la Red de Petri temporizada. Considerando la transición sensibilizada t_j generadora del token inducido en el lugar L, la variable chronos de éste tendrá como mínimo el valor

$$C(t_j) + d(t_j, t_x) \qquad (2.1)$$

donde t_x es una transición de salida del lugar L.

En el caso de que no exista camino desde la transición t_j hasta el lugar L, el valor de la variable chronos del token inducido sería infinito. Este valor no implica que el disparo de la transición t_j conduzca en algún estado posterior de la Red de Petri a la existencia de un token en el lugar L. El valor infinito es utilizado para indicar que nunca podrá crearse un token inducido por la transición t_j en el lugar L.

Suponiendo que el token presente en el lugar L_1 de la figura 2.2 tiene un valor de chronos igual a 5, el tiempo de disparo de la transición t_1 es 8, y por tanto el valor de la

variable chronos de los token inducidos por la transición t_1 en los lugares L_2, L_3, L_4, L_5 y L_7 es 8, 8, 10, 27 y 15 respectivamente.

2.2.1. Algoritmo para el cálculo de distancias temporales

Se presenta un algoritmo recursivo para calcular la distancia temporal desde una transición dada al resto de transiciones de la Red de Petri [DIJ 59]. El algoritmo es utilizado en las redes de computadores para encontrar el camino más corto entre dos nodos determinados.

El algoritmo implementa un puntero que va recorriendo la Red desde la transición inicial. A cada transición de la Red se le asocia una variable denominada *distancia* que representa la distancia temporal que tiene desde la transición inicial a lo largo de un camino conocido como el más corto.

El algoritmo comienza iniciando el puntero a la transición inicial y la variable *distancia* de cada transición a cero.

La duración de la transición inicial se supone cero, para ser consistente con la definición dada de distancia temporal en la que no se contabiliza la duración de las transiciones extremo.

El algoritmo busca todos los posibles caminos que correspondan a una secuencia $t_i L t_j$, donde la transición origen t_i es la transición apuntada por el puntero. De esta forma el puntero realiza el recorrido de la Red transición a transición.

El puntero avanza hasta la transición destino t_j de uno de los caminos localizados. En la variable *distancia* de la transición destino se almacena el valor de *distancia* asociado a la transición origen más la duración de la transición origen.

Si el puntero alcanza una transición con un valor de *distancia* inferior al valor de *distancia* de la transición origen más su duración, el algoritmo finaliza su recorrido por ese ramal y retrocede hasta la primera bifurcación anterior. En caso contrario prosigue con el recorrido. El algoritmo finaliza cuando no existan caminos por donde discurrir el puntero.

Las transiciones que no tengan asociadas ningún valor de *distancia* son inaccesi-

bles desde la transición inicial, considerando por tanto su distancia temporal desde la transición inicial de valor infinito.

El algoritmo se representa mediante un diagrama de flujo en la figura 2.3. En la figura 2.4 se muestran dos Redes de Petri temporizadas a las que se les ha aplicado el algoritmo propuesto.

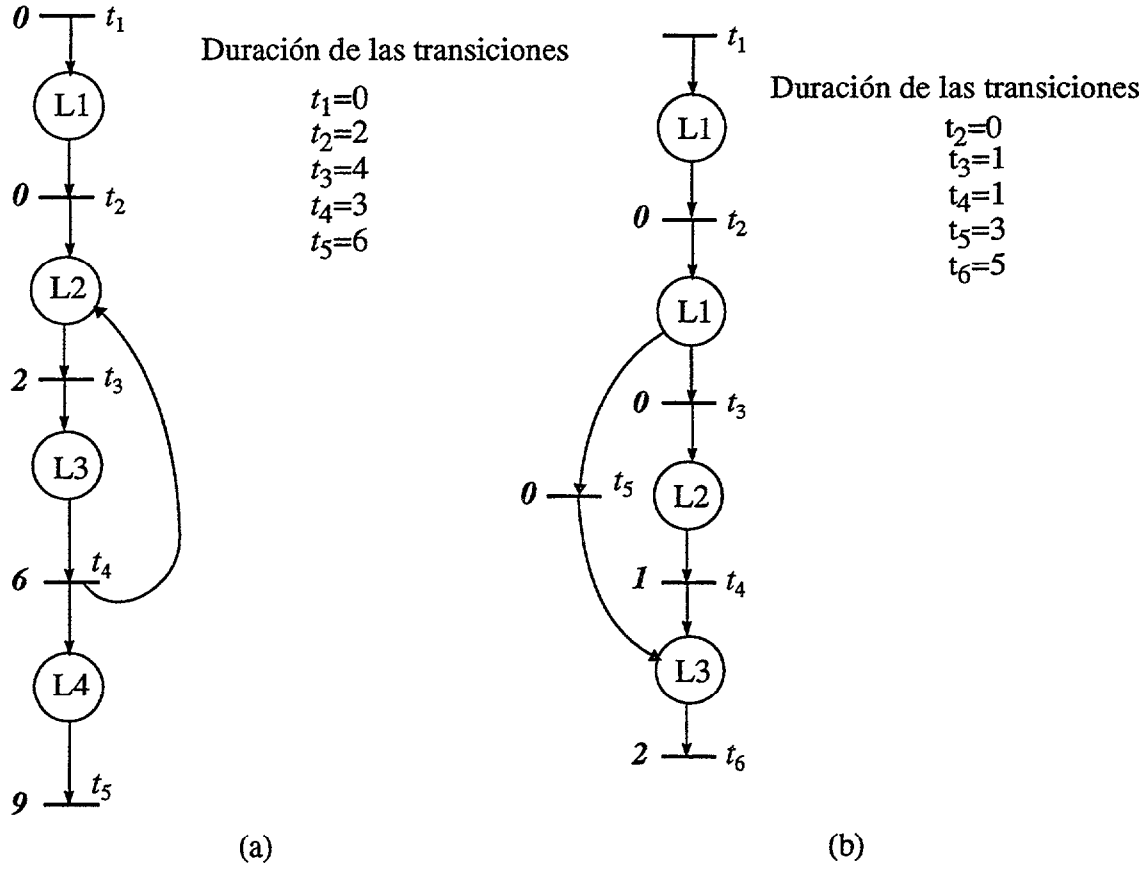


Figura 2.4. Ejecución del algoritmo de distancias sobre dos grafos

En la Red de la figura 2.4(a) se pretende calcular la distancia desde la transición t_1 al resto de transiciones de la Red. El puntero del algoritmo es iniciado a la transición t_1, y su variable *distancia* a cero. El puntero avanza hasta la transición t_2 y en su variable *distancia* se almacena el valor de *distancia* de t_1 más la duración de t_1, cuyo valor es 0 (se supone que la duración de la transición inicial es cero). Seguidamente el puntero avanza hasta la transición t_3 y le almacena el valor 2. El puntero se adelanta hasta la transición t_4 que se le asigna el valor 6 (valor de *distancia* de t_3 más la duración de t_3).

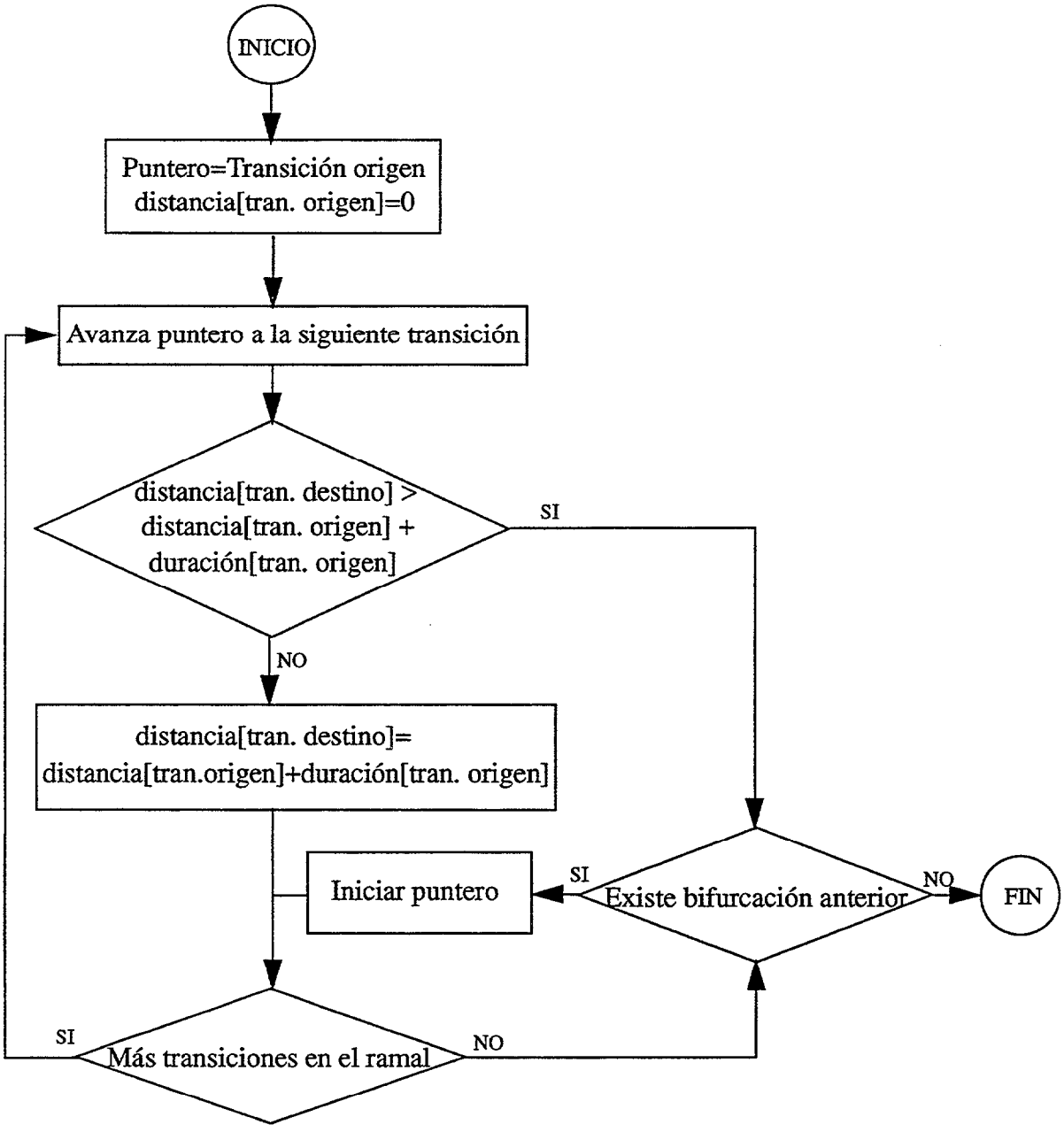


Figura 2.3. Diagrama de flujo del algoritmo de cálculo de la distancia temporal

La transición t_4 tiene una bifurcación con dos arcos de salida. Seleccionando el arco izquierdo se prosigue hasta la transición t_5, asignándole el valor 9. Al no existir más transiciones por ese ramal el puntero retrocede hasta la bifurcación anterior t_4.

Escogiendo el otro camino de la bifurcación se alcanza la transición t_3, que contiene el valor 2. Por ser mayor el valor de *distancia* de la transición origen t_4 más su duración (6+3) que el valor asociado a la transición t_3 (2), se finaliza el recorrido del puntero por ese ramal. No existiendo más caminos posibles, el algoritmo finaliza y el valor de *distancia* asociado a cada transición representa la distancia temporal desde la transición inicial t_1.

El almacenamiento en cada transición de la distancia temporal desde la transición origen, permite evaluar la conveniencia de continuar con un camino si llegamos a una transición por dos caminos diferentes. De manera que si llegamos a una transición que alberga una distancia temporal menor que la del puntero, significa que el camino que estamos explorando no es mínimo, pues existe otro camino que ha llegado al mismo lugar y en menos tiempo (el almacenado en la transición). Y por otra parte se evita la existencia de recorridos que formen bucles infinitos.

En la Red de la figura 2.4(b) se pretende calcular la distancia desde la transición t_2 al resto de la Red. Fijamos el puntero a la transición t_2 y se inicia su variable *distancia* a cero. La transición t_2 es una bifurcación con dos caminos posibles. Se escoge recorrer el camino del arco de la izquierda llegando a la transición t_5 y asignándole a su variable *distancia* el valor 0. El puntero prosigue su recorrido hasta la transición t_6, almacenándole el valor 3 y dando por finalizada la búsqueda por este ramal de la Red por no existir posibilidad de avance del puntero.

El puntero retrocede hasta la transición anterior t_2 donde tuvo lugar la bifurcación. Recorremos el ramal derecho por las transiciones t_3, t_4 y t_6, asignando los valores 0, 1 y 2 respectivamente. Finalmente observar que la variable *distancia* de la transición t_1 no tiene ningún valor asociado, por lo que su distancia desde la transición inicial t_2 es infinito.

2.3. Estudio de causalidad en Redes de Petri de alto nivel temporizadas

Sean dos transiciones sensibilizadas t y t' de una Red de Petri de alto nivel temporizada, se dice que t tiene una relación causa-efecto o relación de causalidad con la transición t' si y sólo si el disparo de t afecta al disparo de la transición t'.

Para que el disparo de una transición t afecte al disparo de otra transición t', los token inducidos por t en el lugar de entrada de t' deben cumplir las siguientes condicio-

nes:

Condición 1: Deben llegar al lugar de entrada de la transición *t'* antes de su disparo.

Condición 2: Deben participar en la sensibilización de la transición *t'* modificando su tupla sensibilizadora.

El concepto de distancia temporal permite evaluar el cumplimiento del primer requisito comparando el valor de la variable chronos del token inducido por *t* con el tiempo de disparo de *t'*.

La comprobación del segundo requisito es más laboriosa pues se hace necesario conocer el valor del resto de los identificadores del token inducido con el objeto de evaluar el cumplimiento del predicado de la transición *t'*.

El trabajo de investigación desarrollado en esta tesis contempla la comprobación de la primera condición, y restringe la segunda condición a la verificación de si el token inducido llega a tiempo de modificar la tupla sensibilizadora (evaluación de la condición temporal del predicado), por lo que su cumplimiento sólo manifiesta una posible relación de causalidad. De esta forma se pretende poner de manifiesto una posible relación de causalidad independientemente de la semántica asociada a cada transición.

Se estudiarán los dos casos posibles según el tipo de condición de lugar utilizada para determinar el token sensibilizador en cada lugar de entrada de la transición. Estas son: "mínimo valor de chronos de entre los tokens del lugar" o "máximo valor de chronos de entre los tokens del lugar" ("mínimo del lugar" y "máximo del lugar" respectivamente).

Se analizará en qué circunstancias el disparo programado de otra transición *t* afecta o no al disparo programado de la transición *t'*. Es obvio que el tiempo de disparo de la transición *t* debe ser menor que el correspondiente a *t'*, de lo contrario sería imposible que *t* pudiera afectar a *t'*.

Las transiciones *t* y *t'* se suponen están sensibilizadas, y se puede establecer un camino que tenga como nodo origen a la transición *t* y como nodo destino a la transición *t'*. Si no existe camino desde *t* a *t'*, obviamente *t* nunca podrá afectar a *t'*.

Se supone que la transición *t* genera un token inducido denotado como $TI_L(t)$ en

el lugar de entrada L de la transición *t'*. De los lugares de entrada de la transición *t'*, L representa al incluido en el camino desde *t* a *t'* sobre el que se ha obtenido la distancia temporal (el más corto).

Con el objeto de obtener mayor claridad en la figura 2.5 se muestra la situación descrita.

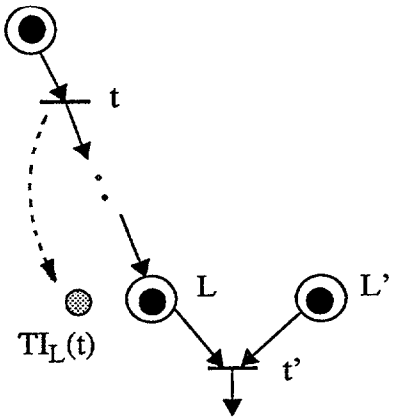
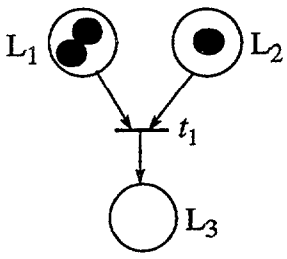


Figura 2.5. Estudio de causalidad

Se considerarán los casos que se pueden producir dependiendo del valor de la variable chronos de $TI_L(t)$ con respecto a S(t'), $S_L(t')$ y C(t') que representan el chronos de sensibilización de la transición *t'*, el chronos del token sensibilizador del lugar L y el tiempo de disparo de *t'* respectivamente.

A continuación se muestra con un ejemplo la obtención de los valores de S(t'), $S_L(t')$ y C(t') en la Red de la figura 2.6.

Los valores de $S_{L1}(t_1)$ y $S_{L2}(t_1)$ se corresponden con el valor de chronos del token sensibilizador de la transición t_1 en los lugares L_1 y L_2, es decir, 5 y 8 respectivamente. El chronos de sensibilización $S(t_1)$ es el máximo entre $S_{L1}(t_1)$ y $S_{L2}(t_1)$ según se indica en el predicado de la transición, luego $S(t_1)$ es 8. Y el tiempo de disparo $C(t_1)$ es 13, igual a $S(t_1)$ más la duración de t_1.



$t_1 = \{<<L_1,L_2>,L_3>| \rightarrow L_3.chronos = max(min(L_1.chronos), min(L_2.chronos)) + 5\}$
Marcado
$L_1.chronos=5$ y 6; $L_2.chronos=8$

Figura 2.6. Determinación de los valores $S(t_1)$, $S_L(t_1)$ y $C(t_1)$

Es obvio que se cumple siempre que

$$S_L(t) \leq S(t) \leq C(t) \qquad (2.2)$$

Condición de lugar "mínimo del lugar"

Caso A $TI_L(t) < S_L(t') \leq S(t') \leq C(t')$

El token inducido por la transición *t* alcanza L con un valor de la variable chronos menor que el chronos del token sensibilizador previamente existente en L.

La condición de lugar "mínimo del lugar" garantiza que de los token existentes será el token de menor chronos el que forme parte de la tupla sensibilizadora de la transición *t'*.

La aparición del token inducido en las condiciones descritas $TI_L(t) < S_L(t')$, puede conducir a que la evaluación de la condición "mínimo del lugar" proporcione el token inducido como parte de la tupla sensibilizadora, en vez de cualquiera de los token presentes en ese lugar.

En estas circunstancias en las que el disparo de la transición *t'* puede verse afec-

tado por el disparo de la transición *t*, es obligado mantener una secuencialidad en los disparos de ambas transiciones, debiendo disparar la transición *t* antes que la transición *t'*, de otra forma el estado resultante no sería el correcto, quedando de manifiesto una posible relación de causalidad entre ambas transiciones.

Caso B $S_L(t') \leq TI_L(t) \leq S(t') \leq C(t')$

El token inducido por la transición *t* alcanza L con un valor de la variable chronos mayor o igual que el chronos del token sensibilizador de *t'* perteneciente al lugar L pero menor o igual que el chronos de sensibilización de la transición *t'*.

El token inducido en L sólo puede alterar el resultado de la condición de lugar si $TI_L(t) = S_L(t')$, pero el cambio del token sensibilizador no produce ningún cambio sustancial en el disparo de la transición *t'* puesto que con cualquiera de los token sensibilizadores se obtiene el mismo resultado.

Caso C $S_L(t') \leq S(t') < TI_L(t) \leq C(t')$

El token inducido en L por la transición *t* tiene valor de chronos comprendido entre el valor del chronos de sensibilización de la transición *t'* y el tiempo de disparo de *t'*. Con este planteamiento y siguiendo el mismo razonamiento empleado en los casos anteriores, el token inducido en L no perjudicará al disparo de la transición *t'*.

Caso D $S_L(t') \leq S(t') \leq C(t') < TI_L(t)$

Si el token inducido por la transición *t* llega a L con un valor de la variable chronos mayor que el instante de tiempo del disparo de la transición *t'*. Ya no existe posibilidad de modificar el disparo de una transición que ha sido previamente realizado.

De esta discusión se puede concluir que la transición *t* puede afectar a la transición *t'* en el caso A, donde

$$TI_L(t) < S_L(t') \qquad (2.2)$$

Haciendo uso de la distancia temporal mediante la expresión 2.1 obtenemos

$$C(t) + d(t, t') < S_L(t') \qquad (2.3)$$

Condición de lugar "máximo del lugar"

Caso A' $TI_L(t) \leq S_L(t') \leq S(t') \leq C(t')$

La condición de lugar "máximo del lugar" selecciona como token sensibilizador el de mayor chronos de todos los presentes en el lugar L.

El token inducido sólo puede alterar el resultado de la condición de lugar si $TI_L(t) = S_L(t')$, pero el cambio del token sensibilizador no produce ningún cambio sustancial en el disparo de la transición *t'* por lo que se considera que la influencia de la transición *t* sobre *t'* es nula.

Caso B' $S_L(t') < TI_L(t) \leq S(t') \leq C(t')$

En este caso el token inducido es el de mayor chronos de los existentes en L, siendo seleccionado como token sensibilizador.

En estas circunstancias se pone de manifiesto una posible relación de causalidad, por lo que el disparo de la transición *t* debe preceder al de la transición *t'*.

Caso C' $S_L(t') \leq S(t') < TI_L(t) < C(t')$

En este caso es aplicable toda la argumentación expuesta para el caso B'.

Caso D' $S_L(t') \leq S(t') \leq C(t') < TI_L(t)$

El token inducido por la transición *t* no puede afectar al disparo de una transición que ha sido previamente realizado.

De los casos analizados con la condición de lugar "máximo del lugar" se concluye que la transición *t* puede afectar a la transición *t'* sólo en los casos B' y C, donde

$$S_L(t') < TI(t) < C(t') \tag{2.4}$$

y

$$S_L(t') < C(t) + d(t, t') < C(t') \tag{2.5}$$

Debe considerarse también la posibilidad de que la transición *t* pueda afectar a *t'* por más de un lugar de entrada, es decir, que exista otro camino que enlace *t* con *t'* y que

discurra por un lugar de entrada de *t'* distinto a L. De esta forma la transición *t* generará otro token inducido $TI_{L'}(t)$ en un lugar de entrada L' distinto de L.

Evidentemente $TI_{L'}(t) \geq TI_L(t)$, puesto que $TI_L(t)$ ha sido generado por el camino más corto.

Si el valor de chronos de $TI_L(t)$ no refleja una relación de causalidad entre las transiciones *t* y *t'* según las expresiones 2.3 y 2.5, debería analizarse si *t* puede generar otro token inducido $TI_{L'}(t)$ en un lugar de entrada de t' distinto de L, y el rango de valores de la variable chronos de $TI_{L'}(t)$ que ponga de manifiesto una relación de causalidad entre las transiciones *t* y *t'*.

En la tabla 2.7 se muestran los resultados de este análisis.

No son contemplados los casos que no pueden producirse. Por ejemplo si el token inducido en L cumple que $S(t') < TI_L(t) \leq C(t')$, el token inducido en L' necesariamente debe cumplir que $S(t') < TI_{L'}(t) \leq C(t')$ puesto que $TI_{L'}(t) \geq TI_L(t)$. En estas circunstancias no se debe analizar los casos que corresponden a chronos de $TI_L(t)$ inferiores al chronos de sensibilización de *t'*.

En la tabla 2.7 se observa que la condición que pone de manifiesto una posible relación de causalidad entre las dos transiciones aplicable con ambos tipos de condiciones de lugar es

$$TI_L(t) < C(t') \tag{2.6}$$

o lo que es lo mismo.

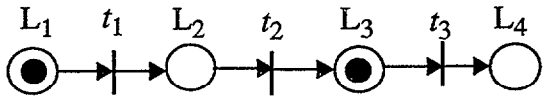
$$C(t) + d(t, t') < C(t') \tag{2.7}$$

En la Red de Petri de la figura 2.8 se estudiará qué valores de la variable chronos del token residiendo en el lugar L_1 podrían causar una relación de causalidad entre las

Tabla 2.7. Resultados del análisis de causalidad

Condición de lugar en L	$TI_L(t)$	Causalidad	Condición de lugar en L'	$TI_{L'}(t)$	Causalidad
mínimo del lugar	Caso A	SI			
	Caso B	NO	mínimo del lugar	Caso A	SI
				Caso B	NO
				Caso C	NO
				Caso D	NO
			máximo del lugar	Caso A'	NO
				Caso B'	SI
				Caso C'	SI
				Caso D'	NO
	Caso C	NO	mínimo del lugar	Caso C	NO
				Caso D	NO
			máximo del lugar	Caso C'	SI
				Caso D'	NO
	Caso D	NO	mínimo del lugar	Caso D	NO
			máximo del lugar	Caso D'	NO
máximo del lugar	Caso A'	NO	mínimo del lugar	Caso A	SI
				Caso B	NO
				Caso C	NO
				Caso D	NO
			máximo del lugar	Caso A'	NO
				Caso B'	SI
				Caso C'	SI
				Caso D'	NO
	Caso B'	SI			
	Caso C'	SI			
	Caso D'	NO	mínimo del lugar	Caso D	NO
			máximo del lugar	Caso D'	NO

transiciones t_1 y t_3.



$t_1 = \{<L_1,L_2>| \rightarrow L_2.chronos = L_1.chronos + 5\}$
$t_2 = \{<L_2,L_3>| \rightarrow L_3.chronos = L_2.chronos + 3\}$
$t_3 = \{<L_3,L_4>| \rightarrow L_4.chronos = L_3.chronos + 8\}$
Marcado inicial
$L_1.chronos = x, L_3.chronos = 3$

Figura 2.8. Red de Petri temporizada

El estudio de causalidad se realiza mediante la evaluación de la expresión 2.7 y los resultados se muestran en la tabla 2.9.

Tabla 2.9. Evaluación de 2.7 en diferentes condiciones

	chronos token(L_1)	**chronos token(L_3)**	**$C(t_1)$**	**$d(t_1,t_3)$**	**$C(t_3)$**	**Causal**
1	2	3	7	3	11	SI
2	3	3	8	3	11	NO
3	4	3	9	3	11	NO

Sólo en el caso 1 existe una relación de causalidad entre ambas transiciones, debiendo ser disparadas secuencialmente primero la transición t_1 y luego t_3.

2.3.1. Causalidad en transiciones en conflicto

Especial atención merece el caso en el que una transición t' tenga un conflicto estático, en cuyo caso se debe considerar la posibilidad de que el disparo de otra transición t desensibilice a la transición t'.

En la figura 2.10 se muestran dos transiciones t' y t'' en conflicto estático. Las transiciones t y t' se encuentran sensibilizadas y se puede establecer un camino desde la

transición t a la transición t'' de modo que $d(t, t'') < \infty$.

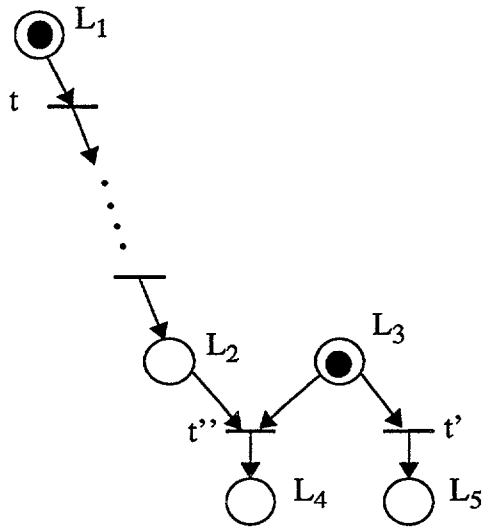


Figura 2.10. Red de Petri temporizada con las transiciones t y t' en conflicto estático

La transición t puede generar un token inducido en el lugar L_2 sensibilizando a la transición t''. Para que el disparo de t'' desensibilice a t' debe cumplirse que

$$C(t'') \leq C(t') \tag{2.8}$$

Descomponiendo los tiempos de disparo se obtiene

$$\text{duración de } t'' + S(t'') \leq \text{duración de } t' + S(t') \tag{2.9}$$

donde S(t'') y S(t') son los chronos de sensibilización de las transiciones t'' y t' respectivamente.

Debido a que las transiciones t'' y t' comparten el lugar L_3, se cumple que

$$S(t'') \geq S(t') \tag{2.10}$$

Considerando la condición de la expresión 2.10, la expresión 2.9 puede reducirse a la siguiente

$$\text{duración de t''} \leq \text{duración de t'} \qquad (2.11)$$

La condición 2.11 junto con la condición 2.12 que impone la necesidad de que el token inducido por la transición *t* llegue a L_2 antes del disparo de *t'* son los requisitos necesarios que deben satisfacerse simultáneamente para que la transición *t* afecte a *t'*.

$$C(t) + d(t, t'') < C(t') \qquad (2.12)$$

2.4. Conclusiones

El concepto de distancia temporal permite conocer el mínimo valor de la variable chronos de un token inducido en un lugar cualquiera de la Red.

La distancia temporal es una generalización del concepto de lookahead visto en el capítulo anterior. Mientras el lookahead relaciona temporalmente los eventos de entrada y salida de un mismo proceso, la distancia temporal considera los eventos de entrada y salida de cualesquiera dos procesos del sistema.

De esta forma, la regla de espera en la entrada del paradigma de Chandy-Misra puede ser evaluada no sólo a nivel local (canales de entrada de un proceso), sino a nivel de sistema (procesos con un camino dirigido al proceso en estudio). No se necesita esperar a un evento por cada canal de entrada de un proceso para averiguar cuál será el evento con el instante de tiempo más pequeño.

Se obtuvo la expresión 2.7 que permite conocer la existencia de una posible relación de causalidad entre dos disparos programados de transiciones de una Red de Petri temporizada, a falta de comprobar que el resto de identificadores del token inducido satisfacen el predicado de la transición afectada para verificar que esa relación de causalidad es real.

La consideración de una posible causalidad es muy pesimista (puesto que habrá posibles relaciones de causalidad que en realidad no lo serán y en algunos casos se forzará la ejecución secuencial de disparos de transiciones que en la práctica pueden ser realizados simultáneamente), pero como contrapartida su verificación es muy simple y rápida.

Capítulo 3: Simulación de Redes de Petri temporizadas

3. Simulación de Redes de Petri temporizadas

Las simulaciones en las que se concentra este capítulo son las simulaciones conservativas, en las que no se ejecuta un evento *e* en un instante de tiempo *t*, hasta estar seguro que los efectos de todos los eventos ocurridos antes que *t* y que puedan afectar a *e* son conocidos.

3.1. Simulación centralizada de Redes de Petri temporizadas

La simulación conservativa de una Red de Petri temporizada se realiza mediante el algoritmo Event Scheduling/Time Advance [BAN 96], que garantiza que todos los disparos se ejecuten en orden cronológico mediante la implementación de una lista de futuros disparos (LFD).

Esta lista contiene todas las transiciones cuyos tiempos de disparo están programados para ocurrir en el futuro. En un instante de tiempo t_{po}, la LFD contiene todas las transiciones cuyo tiempo de disparo ha sido previamente iniciado para ocurrir en el futuro.

Las transiciones que componen la LFD son ordenadas según su tiempo de disparo de menor a mayor, de manera que los disparos sean realizados cronológicamente (en cumplimiento del axioma 3), esto es, deben satisfacer

$$t_{po} \leq C(t_i) \leq \ldots. \leq C(t_j) \qquad (3.1)$$

donde t_{po} es el valor actual del tiempo simulado, y t_i, ... , t_j son las transiciones sensibilizadas pertenecientes a la LFD.

El disparo de la transición t_i recibe el nombre de disparo inmediato, por ser el siguiente disparo que se realizará.

Una vez ordenados los elementos que componen la LFD, el tiempo simulado es avanzado al valor $C(t_i)$, y la transición t_i es disparada y retirada de la LFD.

La ejecución del disparo inmediato puede dar lugar a nuevas sensibilizaciones de transiciones, cuyos tiempos de disparo son calculados y en función de este valor colocadas en la posición correspondiente dentro de la LFD.

A continuación, el tiempo simulado avanza hasta el valor del tiempo de disparo del nuevo disparo inmediato, la transición es disparada y retirada de la LFD. El proceso se repite hasta que se finalice la simulación.

Como alternativa a las operaciones de inserción de nuevas transiciones y extracción del disparo inmediato, la LFD puede ser iniciada y ordenada en cada nuevo estado de la Red de Petri.

Así la simulación debe seguir cíclicamente los siguientes pasos hasta su finalización:

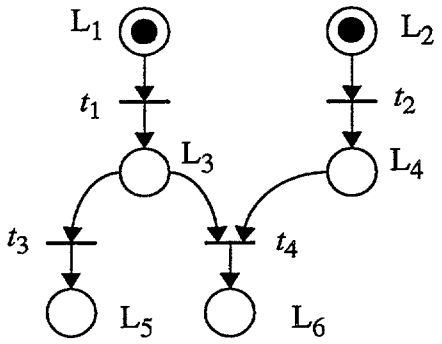
Paso 1 - Iniciar la LFD mediante la obtención del conjunto de transiciones sensibilizadas.
Paso 2 - Ordenar la LFD de acuerdo a los tiempos de disparo de las transiciones que lo componen.
Paso 3 - Seleccionar el disparo inmediato.
Paso 4 - Ejecutar el disparo inmediato.
Paso 5 - Avanzar el tiempo simulado al tiempo de disparo del disparo inmediato.

Mediante la figura 3.1 se ilustrará el funcionamiento del algoritmo explicado.

Inicialmente la Red de Petri tiene las transiciones t_1 y t_2 sensibilizadas con $C(t_1)=1$ y $C(t_2)=2$. El tiempo simulado es cero y la LFD contiene las transiciones $\{t_1, t_2\}$ ya ordenadas. El disparo inmediato corresponde a la transición t_1. La transición t_1 es disparada y el tiempo simulado es avanzado a $C(t_1)$.

En este nuevo estado de la Red, se encuentran sensibilizadas las transiciones t_2 y t_3 con $C(t_2)=2$ y $C(t_3)=6$. La LFD está compuesta por $\{t_2, t_3\}$. El disparo inmediato es la transición t_2. Se dispara t_2 y el tiempo simulado avanza a $C(t_2)$.

El disparo de t_2 sensibiliza a la transición t_4 con $C(t_4)=8$ y la nueva LFD la componen las transiciones $\{t_3, t_4\}$. Se dispara la transición de t_3 y el tiempo simulado avanza a $C(t_3)$. El disparo de t_3 desensibiliza a la transición t_4 por lo que en el nuevo



$t_1 = \{<L_1,L_3>| \rightarrow L_3.chronos = L_1.chronos + 1\}$
$t_2 = \{<L_2,L_4>| \rightarrow L_4.chronos = L_2.chronos + 2\}$
$t_3 = \{<L_3,L_5>| \rightarrow L_5.chronos = L_3.chronos + 5\}$
$t_4 = \{<<L_3,L_4>,L_6>| \rightarrow L_6.chronos = \max(L_3.chronos,L_4.chronos) + 6\}$
Marcado inicial
$L_1.chronos = 0; L_2.chronos = 0$

Figura 3.1. Red de Petri

estado resultante de la Red no existe ninguna transición sensibilizada y la simulación se considera finalizada.

La ejecución de la simulación de la Red de Petri de la figura 3.1 sobre una máquina monoprocesadora consistirá en la realización secuencial de los disparos indicados.

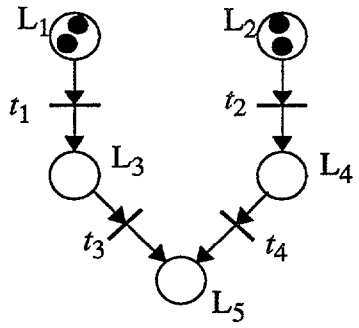
Durante la ejecución de una Red de Petri se pueden obtener varias transiciones sensibles con el mismo tiempo de disparo. Si éstas se encuentran en la cabeza de la LFD, el disparo inmediato lo compondrán todas ellas. Y su ejecución en una máquina monoprocesadora será secuencial en un orden de ejecución aleatorio.

En la figura 3.2 se muestra una Red de Petri temporizada cuya ejecución deriva en la siguiente secuencia de disparos

$$\{ t_{1,1}, t_{1,2}, t_{2,2}, t_{3,3}, t_{2,4}, t_{3,4}, t_{4,4}, t_{4,6} \} \qquad (3.2)$$

donde $t_{i,j}$ simboliza el disparo de la transición i en el instante de simulación j.

En la figura 3.3 se muestra gráficamente su ejecución secuencial, donde el orden



$t_1 = \{<L_1,L_3>| \rightarrow L_3.chronos = min(L_1.chronos) + 1\}$
$t_2 = \{<L_2,L_4>| \rightarrow L_4.chronos = min(L_2.chronos) + 2$
$t_3 = \{<L_3,L_5>| \rightarrow L_5.chronos = min(L_3.chronos) + 2$
$t_4 = \{<L_4,L_5>| \rightarrow L_5.chronos = min(L_4.chronos) + 2$
Marcado inicial
$L_1.chronos = 0$ y 1; $L_2.chronos = 0$ y 2

Figura 3.2. Red de Petri temporizada

de realización de los disparos de las transiciones con igual tiempo de disparo es aleatorio. El eje horizontal representa el tiempo de ejecución de la simulación, y sobre él se indica la transición disparada y el instante de tiempo simulado. Se ha supuesto que el tiempo de procesamiento de la transición t_1 es 2 y el del resto de las transiciones es 3.

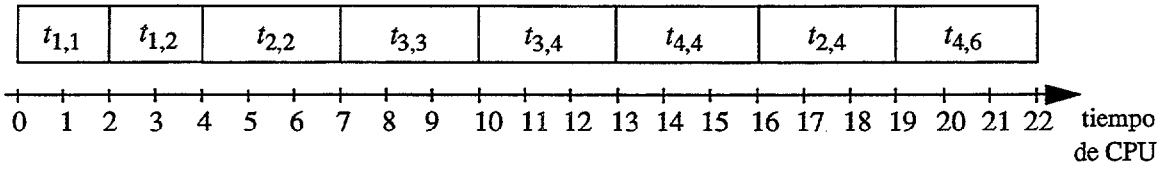


Figura 3.3. Ejecución centralizada de la Red de la figura 3.2

3.2. Simulación paralela de Redes de Petri temporizadas

El algoritmo Event Scheduling/Time Advance expuesto en el punto anterior no impone ningún orden de realización para la ejecución de las transiciones con igual tiempo de disparo. Ese grado de libertad permite su realización simultánea. De esta forma, se puede implementar una simulación paralela consistente en distribuir la ejecu-

ción de los disparos de las transiciones con igual instante de disparo entre un conjunto de procesadores M (en el caso de que el número de procesadores M sea inferior al número de disparos a realizar simultáneamente, es necesario una política de distribución de tareas que equilibre la carga entre los diferentes procesadores, con el objetivo de obtener un tiempo de ejecución menor).

En la figura 3.4 se representa gráficamente la ejecución de la Red de la figura 3.2 sobre dos procesadores mediante la implementación descrita en el párrafo anterior.

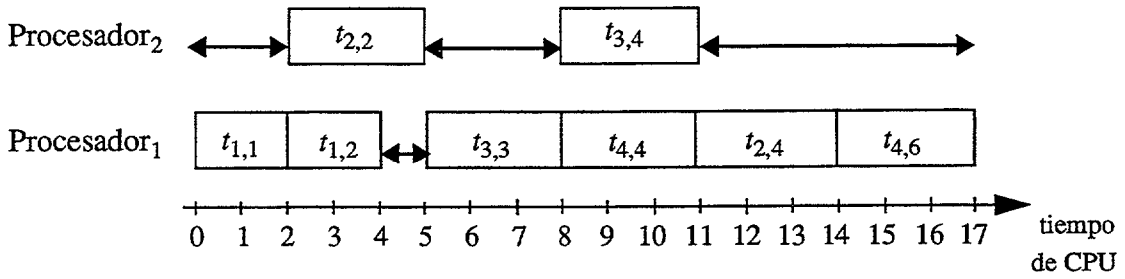


Figura 3.4. Simulación paralela sobre 2 procesadores

Aunque el tiempo de ejecución de la simulación paralela ha disminuido considerablemente con respecto a la ejecución monoprocesadora, se observa una infrautilización de los recursos disponibles (zonas delimitadas mediante flechas) debido a que el algoritmo Event Scheduling/Time Advance exige finalizar la totalidad de los disparos inmediatos antes de recomponer la LFD, dando lugar a la aparición de estados de inactividad en algunos de los procesadores. Esto queda reflejado en las siguientes situaciones:

- Cuando no existe más de una transición con el mismo instantes de disparo. En estas circunstancias, la simulación paralela sólo requiere un único procesador manteniendo al resto de procesadores en estado de inactividad. Esto se observa en la figura 3.4 en los intervalos de tiempo de ejecución [0,2], [5,8], etc.

- Cuando existiendo varias transiciones con el mismo tiempo de disparo, la finalización de su total ejecución no es simultánea por todos los procesadores. Esto se observa en el intervalo de tiempo de ejecución [4,5].

3.2.1. Disminución de los estados de inactividad

A continuación se expondrán algunas ideas que nos permitirán reducir las zonas inactivas, disminuyendo en consecuencia el tiempo total empleado en la ejecución de la simulación.

Un examen detallado de la ejecución de la Red de la figura 3.2 muestra que mediante la siguiente secuencia de disparos se obtiene el mismo estado final de la ejecución

$$\{ <t_{1,1}, t_{2,2}>, <t_{1,2}, t_{2,4}, t_{3,3}, t_{4,4}>, <t_{3,4}, t_{4,6}> \} \qquad (3.3)$$

en donde las transiciones sensibilizadas con distintos tiempos de simulación (tiempo de disparo) han sido realizadas en el mismo instante de ejecución siempre y cuando no se manifestara una relación de causalidad entre ellas, lo que garantiza la corrección de los resultados [RIG 89]. Los disparos de las transiciones realizados simultáneamente se han agrupado entre los símbolos "<" y ">".

Los estados de inactividad de los procesadores se reducen al dispararse un mayor número de transiciones simultáneamente en cada momento, de manera que la simulación paralela sobre dos procesadores finalizaría en un tiempo de ejecución menor que 17.

Se estudiará hasta qué instante de ejecución puede ser adelantado un instante de simulación dado, para no violar las relaciones de causalidad existentes.

En la figura 3.5 se muestra el grafo de precedencia de eventos de las transiciones t_1 y t_2, donde e_k representa el disparo de una transición en el instante simulado k. En la parte inferior de la figura, el eje horizontal indica el instante de tiempo de CPU en el que es simulado cada disparo.

Supongamos que la simulación paralela se encuentra en un estado donde ha simulado todos los disparos con un instante de simulación menor que i, es decir, el tiempo transcurrido de ejecución de la simulación es $t\text{-}1$. El siguiente disparo que se realizará será e_i en el tiempo de ejecución t. Se pretende averiguar si podemos realizar simultáneamente con el disparo e_i algún otro disparo.

Si realizamos el disparo e_{i+n} en el tiempo de ejecución t, gráficamente se com-

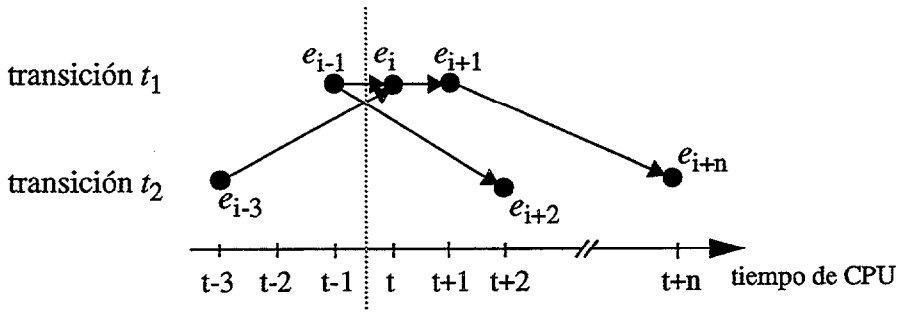


Figura 3.5. Grafo de precedencia de eventos

prueba la violación de la relación de causalidad que impone la condición de realizar el disparo e_{i+1} antes que el disparo e_{i+n}. El disparo e_{i+1} no puede ser adelantado pues se exige la realización previa de e_i (la distancia temporal de una transición a ella misma es cero, lo que revela una relación de causalidad entre los diferentes disparos de una misma transición), y éste todavía no ha sido ejecutado.

El disparo e_{i+2} es partícipe de una relación de causalidad que exige la realización de e_{i-1} antes de la de e_{i+2}. Al haber sido ya ejecutado el disparo e_{i-1}, el disparo e_{i+2} puede ser realizado conjuntamente con el disparo e_i.

De este análisis se puede concluir:

- El disparo de una transición que deba esperar a la realización de un disparo previo, puede ser adelantado hasta el instante de ejecución inmediatamente posterior al instante de ejecución del disparo previo.

- El disparo del resto de transiciones puede realizarse en cualquier momento.

- Las transiciones que sean disparadas conjuntamente deben ser mutuamente independientes.

Un método gráfico para obtener conjuntos de transiciones mutuamente independientes consiste en desplazar los disparos de transiciones hacia instantes de ejecución más pequeños, sin alterar las relaciones de causalidad, es decir, las flechas que indican una relación causa-efecto deben mantener una dirección y sentido de izquierda a derecha.

En la simulación de una Red de Petri temporizada, la existencia de una posible relación de causalidad debe ser descubierta mediante la evaluación de la expresión 2.7.

Aplicando las conclusiones expuestas en la simulación paralela de la figura 3.2 se obtiene una disminución del tiempo de ejecución de la simulación como se muestra en la figura 3.6 que corresponde a la secuencia de disparo

$$\{ <t_{1,1}, t_{2,2}>, <t_{1,2}, t_{2,4}, t_{4,4}>, <t_{3,3}, t_{4,6}>, t_{3,4} \}. \qquad (3.4)$$

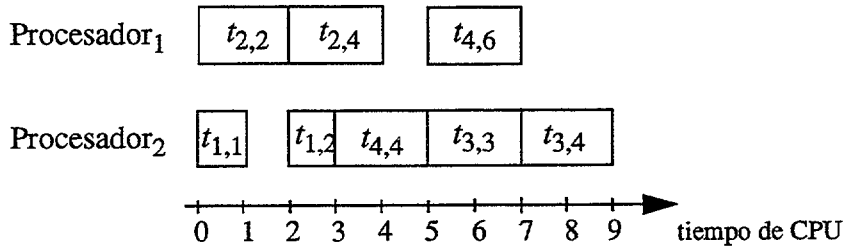


Figura 3.6. Simulación paralela sobre 2 procesadores

Al inicio de la simulación, las transiciones t_1 y t_2 son disparadas simultáneamente. Ambas transiciones son mutuamente independientes pues no existe camino que enlace ambas transiciones. Como resultado de los disparos se sensibilizan las transiciones t_1, t_2, t_3 y t_4 con $C(t_1)=2$, $C(t_2)=4$, $C(t_3)=3$ y $C(t_4)=4$.

Una relación de causalidad entre dos transiciones $t => t'$ no puede generarse si C(t)>C(t'). Esto restringe las combinaciones posibles entre transiciones para evaluar la expresión 2.7 a las siguientes

$C(t_1) + d(t_1, t_2) > C(t_2)$	No causal
$C(t_1) + d(t_1, t_3) < C(t_3)$	Causal
$C(t_1) + d(t_1, t_4) > C(t_4)$	No causal
$C(t_3) + d(t_3, t_2) > C(t_2)$	No causal
$C(t_3) + d(t_3, t_4) > C(t_4)$	No causal

Se observa una relación de causalidad $t_1 => t_3$, lo que descarta la posibilidad de disparar t_3 simultáneamente con t_1. Con el disparo de t_1, t_2 y t_4 la Red alcanza un nuevo estado con t_3, t_4 y t_5 sensibilizadas. Procediendo de igual forma, se consigue el grupo de

transiciones mutuamente independientes formado por t_3 y t_4.

Las diferencias existentes entre las secuencias de disparo indicadas en las expresiones 3.3 y 3.4 son debidas a que en ésta última la existencia de una relación de causalidad ha sido revelada mediante la evaluación de la expresión 2.7, mucho más restrictiva que la expresión 2.3 que es la que mejor se adapta al tipo de condición de lugar empleada en la Red de Petri de la figura 3.2.

La evaluación de la expresión 2.3 sobre las transiciones sensibilizadas en el segundo estado de la Red es

$C(t_1) + d(t_1, t_2) > S_p(t_2)$	No causal
$C(t_1) + d(t_1, t_3) > S_p(t_3)$	No causal
$C(t_1) + d(t_1, t_4) > S_p(t_4)$	No causal
$C(t_3) + d(t_3, t_2) > S_p(t_2)$	No causal
$C(t_3) + d(t_3, t_4) > S_p(t_4)$	No causal

donde no se refleja ninguna relación de causalidad, lo que derivaría en la secuencia de disparos de la expresión 3.3.

3.3. Algoritmo para la simulación paralela de Redes de Petri temporizadas

Partamos de una Red de Petri temporizada con un marcado conocido y por lo tanto un conjunto de transiciones sensibilizadas. Este conjunto de transiciones ordenadas por su tiempo de disparo compondrán la LFD.

Se intentará obtener un subconjunto de la LFD compuesto de transiciones sensibilizadas mutuamente independientes, y que por lo tanto pueden ser disparadas simultáneamente. Lo llamaremos TMI.

El conjunto TMI se iniciará con una copia del disparo inmediato de la LFD. Esta transición no se encuentra afectada por una relación causa-efecto, pues no existe otra transición con menor tiempo de disparo. En el caso de que el disparo inmediato conste de varias transiciones, se procederá de igual forma, pues ya se expuso en el lema 1 del capítulo 1 que dos transiciones con igual tiempo de disparo son mutuamente indepen-

dientes.

Ampliaremos el conjunto TMI copiándole las transiciones pertenecientes a la LFD que sean mutuamente independientes con las previamente existentes en TMI.

Para una mejor comprensión del algoritmo debe recordarse que el añadir una transición al conjunto TMI es una operación de copiado, que en ningún caso significa que la transición se elimine de la LFD.

Supongamos que se pretende estudiar la inclusión de una transición determinada *t'* de la LFD en el conjunto TMI. Para ello, se averiguará si *t'* se encuentra afectada por alguna relación de causalidad. La LFD se puede particionar en tres subconjuntos disjuntos: el subconjunto de transiciones $s_{min}(t')$ cuyos tiempos de disparo serán menores que el correspondiente de la transición *t'*, el subconjunto de transiciones $s_{eq}(t')$ cuyos tiempos de disparo son iguales que el de la transición *t'*, y otro subconjunto de transiciones $s_{max}(t')$ cuyos tiempos de disparo son mayores que el de la transición *t'*.

Siendo *t''* una transición perteneciente al conjunto $s_{max}(t')$ o $s_{eq}(t')$ es obvio que *t'* no se encuentra afectada por la transición *t''*.

Con respecto al conjunto $s_{min}(t')$ habrá que utilizar la expresión 2.7 para averiguar si existe alguna transición *t''* de este conjunto que pueda generar una relación de causalidad sobre *t'*, *t''* => *t'*.

Concluyendo, una transición cualquiera *t'* perteneciente a la LFD se añadirá al conjunto TMI si no existe una relación de causalidad entre ningún elemento de $s_{min}(t')$ hacia la transición en estudio *t'*.

El método descrito se seguirá hasta haber estudiado la inclusión de la totalidad de las transiciones que componen la LFD.

Como ya se comento anteriormente, el hecho de que el disparo de alguna transición del conjunto TMI pueda sensibilizar una transición que no se encontraba previamente en la LFD obliga a recomponer de nuevo la LFD con el objeto de añadir las nueva transiciones sensibilizadas.

A modo de resumen, los pasos a seguir en el algoritmo de simulación paralela propuesto son:

Paso 1 - Iniciar la LFD mediante la obtención del conjunto de transiciones sensibilizadas.

Paso 2 - Ordenar la LFD en orden ascendente según los tiempos de disparo de las transiciones que lo componen.

Paso 3 - Iniciar el conjunto TMI con la transición o transiciones que componen el disparo inmediato.

Paso 4 - Seleccionar la siguiente transición de la LFD y comprobar la existencia de relación de causalidad entre las transiciones de la LFD con un tiempo de disparo menor y la seleccionada. En caso afirmativo desechar la transición, y en caso contrario añadirla al conjunto TMI.

Paso 5 - Volver al paso 4 hasta haber seleccionado todos los elementos de la LFD.

Paso 6 - Avanzar el tiempo simulado al tiempo de disparo del disparo inmediato.

Paso 7 - Disparar simultáneamente todas las transiciones incluidas en TMI.

Paso 8 - Ir al paso 1.

La simulación paralela alterna entre fases donde su ejecución es secuencial y otras fases con una ejecución paralela.

En la figura 3.8 se indican los pasos del algoritmo de simulación paralela que englobaría cada fase.

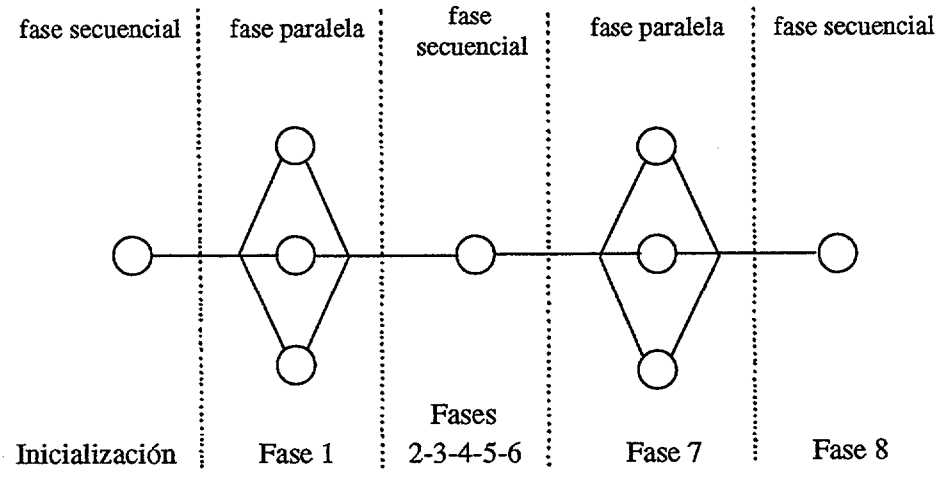
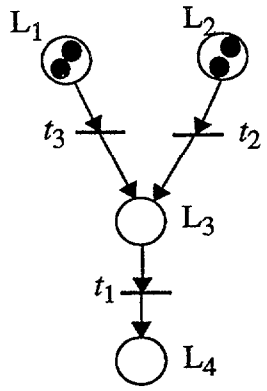


Figura 3.7. Fases del algoritmo de simulación paralela

3.4. Cumplimiento del axioma 3

En la ejecución de una Red STER puede ocurrir el caso particular de la existencia de una tupla sensibilizadora que sensibilice a una transición t con un instante de disparo menor que el último disparo realizado en la Red. En este caso, el axioma 3 impide que la transición t sea disparada y los token sensibilizadores son considerados como token muertos.

Un ejemplo se ilustra mediante la Red de Petri de la figura 3.7.



$t_1 = \{<L_3,L_4> \rightarrow L_4.chronos = max(L_3.chronos) + 5\}$
$t_2 = \{<L_2,L_3> \rightarrow L_3.chronos = min(L_2.chronos) + 2\}$
$t_3 = \{<L_1,L_3> \rightarrow L_3.chronos = min(L_1.chronos) + 10\}$
Marcado inicial
$L_1.chronos = 10$ y 30; $L_2.chronos = 0$ y 2

Figura 3.8. Red de Petri temporizada

La ejecución de la Red da lugar a la siguiente secuencia de disparos produciendo un token muerto en el lugar L_3

$$\{t_{2,2}, t_{2,4}, t_{1,9}, t_{3,20}, t_{1,25}, t_{3,40}, t_{1,45}\}$$

el token del lugar L_3 con chronos 2 está muerto

La simulación paralela mediante el algoritmo expuesto produce la siguiente secuencia de disparos

$$\{ <t_{2,2}, t_{3,20}>, <t_{2,4}, t_{1,25}>, t_{1,9}, t_{1,7}, t_{3,40}, t_{1,45} \}$$

en cuyo estado final no existe ningún token muerto.

Una Red ejecutada con el algoritmo de simulación paralela expuesto anteriormente no da lugar nunca a token muertos pues no mantiene la cronología de los disparos, lo que impide evaluar cuándo el disparo de una transición puede violar el axioma 3. Por lo general, esta situación sólo se presenta en transiciones en cuyo predicado se utiliza la condición de lugar "máximo del lugar".

La solución consiste en analizar el valor de chronos de los token almacenados en L_3, de manera que un token *tk* se considerará muerto si existe otro token *tk'* que cumpla la condición 3.5.

$$tk'.\text{chronos} < tk.\text{chronos} + \text{duración de } t_1 \qquad (3.5)$$

Con la solución adoptada la secuencia de disparos se corresponde con

$$\{ \langle t_{2,2}, t_{3,20}\rangle, \langle t_{2,4}, t_{3,40}\rangle, t_{1,45}, t_{1,25}, t_{1,9} \}$$

el token del lugar L_3 con chronos 2 se considera muerto

3.5. Corrección del algoritmo de simulación paralela

En este punto se demostrará que la ejecución de una Red de Petri mediante el algoritmo de simulación paralela propuesto en el punto 3.4 es equivalente a la secuencia de disparos ordenada en el tiempo que se obtendría de la misma Red de Petri con una ejecución centralizada y un modelo temporal fuerte, es decir, el estado final de la Red de Petri en ambos casos es igual.

Se comenzará mostrando que la secuencia de disparos obtenida mediante el algoritmo de simulación paralela puede ser transformada en una secuencia de disparos ordenada en el tiempo.

La demostración se realizará por inducción.

Sea *s'* la secuencia de disparos obtenida con el algoritmo de simulación paralela

propuesto, y s la secuencia de disparos cronológicamente ordenada.

Para el caso de que la secuencia de disparos s' conste de un sólo elemento es obvio que $s'=s$.

Supongamos una secuencia de disparos s' compuesta por n disparos que puede ser transformada en una secuencia de disparos ordenada en el tiempo, y analicemos el caso de una secuencia s' formada por $n+1$ elementos.

El subconjunto de los n primer elementos de la secuencia s' puede ser ordenado en el tiempo según su tiempo de disparo en virtud de la suposición realizada, obteniendo la secuencia de disparos s. Tanto si el disparo del elemento $n+1$ se ha realizado simultáneamente con el disparo del elemento de posición n como si no, se puede establecer un orden tal que el elemento de posición $n+1$ figure con posterioridad al elemento n, de manera que la secuencia de disparos aparezca como un vector

$$s' = \{t'_1, t'_2, \ldots, t'_n, t'_{n+1}\} = \{s, t'_{n+1}\} = \{t_1, t_2, \ldots, t_n, t'_{n+1}\}$$

Si se cumple que $C(t_n) \leq C(t'_{n+1})$ entonces s' es ordenada en el tiempo.

Si $C(t_n) > C(t'_{n+1})$, ambos elementos pueden ser ordenados en el tiempo con tan solo permutar la posición de los dos elementos. Esta operación es correcta siempre y cuando no exista una relación de causalidad entre ambas transiciones $t_n \Rightarrow t'_{n+1}$, que obligue a realizar el disparo de t_n antes que el de t'_{n+1}. Por contradicción se mostrará que la causalidad mencionada no puede ocurrir con el algoritmo de simulación paralela.

Supongamos que existe la relación de causalidad indicada $t_n \Rightarrow t'_{n+1}$, lo que significa que existe un camino entre ambas transiciones tal que

$$C(t_n) + d(t_n, t_{n+1}) < C(t'_{n+1}) \qquad (3.6)$$

La condición 3.6 sólo se satisface si el valor de la distancia temporal es negativo, lo que contradice la definición de distancia temporal dada en el capítulo 2.

Una vez permutadas las transiciones t_n y t'_{n+1} se obtendrá la secuencia de disparos s'' indicada a continuación.

$$s'' = \{t_1, t_2, \ldots, t_{n-1}, t'_{n+1}, t_n\}$$

donde se debe analizar la relación entre $C(t_{n-1})$ y $C(t'_{n+1})$.

Sucesivamente la transición t'_{n+1} se permutará con la transición que ocupe una posición anterior en la secuencia de disparos, hasta que se cumpla $C(t_w) \leq C(t'_{n+1})$, siendo t_w la transición que ocupa una posición anterior a la de t'_{n+1}. La secuencia de disparos resultante estará ordenada en el tiempo según el tiempo de disparo de los elementos que la componen.

La secuencia de disparos obtenida cumple los axiomas 1, 2 y 3 enunciados en el punto 1.1.4. A continuación se demostrará que esta secuencia de disparos se corresponde con la ejecución de la Red de Petri bajo un modelo temporal fuerte, salvo aquellas diferencias que puedan aparecer debidas a la indeterminación inherente en este tipo de Redes, que no pueden ser imputadas a los algoritmos de ejecución.

Supongamos que ejecutamos una Red de Petri con un modelo de tiempo fuerte hasta un tiempo simulado Z, de modo que la secuencia de disparos resultante no contendrá ninguna transición cuyo tiempo de disparo sea mayor que Z. La ejecución centralizada deriva en una secuencia de disparos $S_s=\{t_1, t_2, \ldots, t_{p-1}, t_p\}$ y la ejecución paralela deriva en una secuencia de disparos que una vez ordenada en el tiempo según el tiempo de disparo de sus elementos denotaremos como $S_p=\{t'_1, t'_2, \ldots, t'_{q-1}, t'_q\}$.

Por inducción se demostrará que ambas secuencias S_s y S_p son idénticas.

Supongamos que la transición t_1 perteneciente a S_s no existe en S_p. El estado de la Red antes del disparo de t_1 se corresponde con el estado inicial y éste es idéntico en ambas ejecuciones: centralizada y paralela. Luego la transición t_1 se encuentra sensibilizada en la ejecución paralela y es la de menor tiempo de disparo, lo que fuerza que t_1 sea la primera transición disparada, siendo $t_1 = t'_1$.

Como hipótesis inductiva se supondrá que las n-1 primeras transiciones de ambas secuencias coinciden, y se estudiará las transiciones de posición n de ambas secuencias de disparo S_s y S_p, y que denotaremos como t_k y t'_m respectivamente.

Supongamos que la transición t_k perteneciente a S_s no existe en S_p. Si ambas secuencias de disparo coinciden hasta el elemento n-1, el estado final obtenido en ambas ejecuciones será idéntico. En la ejecución paralela, la transición t_k será sensible en este estado y es la de menor tiempo de disparo, luego $t_k = t'_m$.

De igual forma, se puede demostrar que todas las transiciones pertenecientes a S_p están incluidas en S_s, obteniendo como conclusión que S_s y S_p tienen igual número de elementos p=q, y son idénticas.

3.6. Conclusiones

La simulación paralela de una Red de Petri se apoya en la característica de concurrencia que tienen estos grafos.

Se ha desarrollado una versión modificada del algoritmo Event Scheduling/ Time Advance para realizar simulaciones paralelas en Redes de Petri temporizadas. En líneas generales, el algoritmo obtenido sigue el mismo esquema que el original, salvo que incluye unos pasos que permiten adelantar el tiempo de simulación del disparo de una transición, aumentando de esta forma su eficacia.

El incremento del número de transiciones a disparar an cada estado de la Red favorece su realización utilizando un sistema multiprocesador.

El algoritmo hace uso de la expresión 2.7 para poner de manifiesto una posible causalidad. Esta falta de certeza en la existencia de la relación de causalidad fuerza, en algunos casos, la ejecución secuencial de los eventos cuando no es necesario, desaprovechando la concurrencia existente. Pero el tiempo de ejecución de la simulación con estas pautas es inferior al que se obtendría si hubiera que verificar realmente todas las relaciones de causalidad evaluando todas las condiciones que componen el predicado de una transición.

Capítulo 4: Algoritmos de distribución de procesos

4. Algoritmos de distribución de procesos

La mayoría de los algoritmos de distribución expuestos en el capítulo 1 constan de una función coste de comunicaciones o función objetivo que debe ser maximizada o minimizada. Las características de nuestro problema no permiten aplicar ninguno de estos algoritmos, convirtiéndose en una problema muy singular debido a la falta de restricciones que nos permita acotar o restringir el problema sin necesidad de recorrer todo el espacio de soluciones.

4.1. Exposición del problema

El problema comprende el estudio de un método de distribución de procesos en un entorno multiprocesador que minimice el tiempo de ejecución del conjunto de procesos.

Las características de nuestra sistema son:

Proceso

a.- Los procesos son indivisibles.
b.- Los procesos son independientes no existiendo relación de precedencia, ni comunicación entre ellos.
c.- Los procesos son generalmente de distinta duración.
d.- El tiempo de ejecución de cada proceso es pequeño con respecto al del algoritmo de distribución.
e.- Se supone conocida una estimación del tiempo de ejecución medio de cada proceso, referido como coste del proceso.

Procesador

f.- Los procesadores son iguales y están fuertemente acoplados (comparten la memoria).

La característica *d* es muy importante y deberá tenerse en cuenta en el momento

de evaluar posibles métodos que impliquen un coste computacional adicional. Por ejemplo, no se contempla la posibilidad de una distribución con expropiación pues se supone que el coste computacional de transferir un proceso ya iniciado en un procesador p a otro procesador p' es mayor que el de finalización del proceso en el procesador p.

En un sistema con las características descritas la única manera de conseguir una disminución del tiempo de ejecución del sistema es distribuir los procesos de manera que se igualen las cargas de cada uno de los procesadores. El problema se reduce a distribuir un conjunto de números (coste de los procesos) en tantos subconjuntos como procesadores existan, de tal forma que el máximo de las sumas de los elementos de cada subconjunto sea lo mínimo posible.

El problema expuesto es una generalización del problema de partición del conjunto, cuyo enunciado es:

Sea Z el conjunto de los números enteros. Dado un conjunto finito A de Z, encontrar un subconjunto de A (A') tal que cumpla:

$$\sum_{a \in A'} a = \sum_{b \in (A-A')} b \qquad (4.1)$$

El problema es NP-completo [CHR 89][PAP 90][SAR 89], siendo posible generar una solución sólo en casos muy concretos (2 procesadores y procesos de igual duración), lo que obliga a recorrer el espacio de soluciones entero para obtener una solución óptima.

El tamaño del espacio de soluciones (cantidad de combinaciones posibles) viene dado por la expresión 4.2, donde p y m representan el número de procesos y procesadores respectivamente.

No se considera como posible solución la derivada de una situación en la que exista un procesador sin carga y otro procesador distinto con más de un proceso asignado. Este tipo de soluciones no son óptimas siendo fácilmente mejorables desplazando algunos de los procesos a los procesadores inactivos, por lo que no se incluyen en el

espacio de soluciones del problema.

$$S(m,p) = \begin{cases} m^{p-m} + \sum_{i=0} m^{i} S(m-1,p-i-1) & \\ 1 & (m \leq p) \vee (m=1) \end{cases} \qquad (4.2)$$

La ecuación 4.2 es recursiva. Los casos triviales se dan cuando el número de procesos es igual o menor al número de procesadores (en cuyo caso la única solución es un proceso por procesador), y cuando sólo existe un sólo procesador (todos los procesos se asignan al procesador).

En la tabla 4.1 se muestra la cantidad de combinaciones posibles para diferentes valores de p y m.

Las cantidades indicadas en la tabla 4.1 serían menores en el caso de existir procesos con costes iguales. De cualquier forma, la cantidad de soluciones es enorme para valores superiores a p=9 y m=3, valores perfectamente alcanzables en una Red de Petri.

Existen diversas técnicas para localizar la solución óptima localmente en el espacio de soluciones del problema. Parten de unas restricciones que disminuyen el campo de búsqueda dentro del espacio de soluciones, lo que no garantiza que la solución encontrada sea la óptima de todo el espacio. Este tipo de técnicas reciben el nombre de métodos óptimos localmente. Los métodos se clasifican en:

- Métodos simulated annealing
- Métodos de programación matemática
- Métodos de búsqueda en el espacio de estados

Los métodos de simulated annealing [KIR 83] conducen a una solución óptima mediante los siguientes pasos:

1 - Realizar una distribución inicial
2 - Variar la distribución mediante pequeños cambios aleatorios
3 - Evaluar la nueva distribución
4 - Si no se consigue mejora -> FIN
5 - Ir al paso 2

Tabla 4.1. Número de combinaciones para distribuir p procesos entre m procesadores

p\m	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	3	1	1	1	1	1	1	1
4	7	6	1	1	1	1	1	1
5	15	25	10	1	1	1	1	1
6	31	90	65	15	1	1	1	1
7	63	301	350	140	21	1	1	1
8	127	966	1701	1050	266	28	1	1
9	255	3025	7770	6951	2646	462	36	1
10	511	9330	34105	42525	22827	5880	750	45
11	1023	28501	145750	246730	179487	63987	11880	1155
12	2047	86526	611501	1379400	1323650	627396	157027	22275
13	4095	261625	2532530	7508500	9321310	5715420	1899610	359502
14	8191	788970	10391700	40075000	63436400	49329300	20912300	5135130
15	16383	2375100	42356000	210767000	420693000	408741000	216628000	67128500

Los métodos de programación matemática [CHU 87] utilizan técnicas de programación para resolver (minimizar o maximizar) una función objetivo. Generalmente se utilizan restricciones que acotan el campo de aplicabilidad de la solución, pero garantizan que ésta exista.

Los métodos de búsqueda en el espacio de estados construyen un árbol con todas las soluciones [MA 82]. La búsqueda consiste en analizar todos los caminos posibles desechando aquellos que conduzcan a soluciones inaceptables.

Los métodos descritos tienen el inconveniente de ser muy complejos, lo que implica un gran tiempo de computación que seguramente no justifica la mejora obtenida con respecto a una solución escogida al azar.

Como alternativa a la generación y estudio del espacio de soluciones, se analizarán los algoritmos heurísticos.

4.2. Algoritmos de distribución heurísticos

El análisis se centrará en los algoritmos que asocian una prioridad a cada proceso. Su ejecución consta de los siguientes pasos:

1. Confeccionar una lista de los procesos, ordenadas de acuerdo a su prioridad
2. Asignar el primer elemento de la lista a un procesador
3. Retirar el elemento asignado de la lista
4. Si existen elementos en la lista, ir al paso 2

Los diferentes algoritmos se diferencian en el criterio seguido para asignar prioridades a los procesos y seleccionar el procesador.

4.2.1. Algoritmo Heavy Task First (HTF)

El presente algoritmo es una versión modificada del descrito en [SHI 90], donde es utilizado para la distribución de procesos con relaciones de precedencia.

A cada proceso se le asigna una prioridad que es función de su tiempo de ejecución, que se denominará coste del proceso.

El algoritmo asigna el proceso con el coste más alto al procesador con menor carga. Al comienzo, todos los procesadores se encuentran sin carga y un conjunto de procesos (los de coste más elevado) igual al número de procesadores son asignados, a razón de uno por procesador. Los diferentes pasos del algoritmo son:

1. Confeccionar lista de procesos
2. Identificar proceso de mayor coste
3. Si no existe --> FIN
4. Identificar procesador con menor carga
5. Asignar proceso a procesador

6. Retirar el proceso de la lista de procesos
7. Volver al paso 2

Ejemplo: Sean 8 procesos con coste={1,8,4,7,2,3,8,4} y 3 procesadores. La distribución obtenida con el algoritmo HTF es la mostrada en la tabla 4.2.

Tabla 4.2. Distribución de los procesos del ejemplo

Procesador	P_1	P_2	P_3
Procesos	8 4	8 3 2	7 4 1
Carga total	12	13	12

El algoritmo es dificilmente identificable con alguno de los mostrados en la figura 1.18 del capítulo 1. Las decisiones concernientes a qué procesador se le asigna un proceso son tomadas durante la ejecución del programa, lo que corresponde a una distribución dinámica. Sin embargo, una vez un proceso ha sido asignado a un procesador, no se contempla la posibilidad de transferirlo a otro procesador. Desde este punto de vista la distribución podría ser considerada estática.

Para evaluar el algoritmo se realizaron unos ensayos con la intención de medir el porcentaje de optimabilidad (la solución obtenida coincide con la óptima). En la tabla 4.3 se muestran los resultados obtenidos para diferentes números de procesos y procesadores.

Se realizaron 250 experimentos para cada valor de p y m. El rango de valores de p y m se limitó a 10 y 9 respectivamente debido al gran coste computacional. Por lo que el comportamiento en el rango de valores experimentado va a ser extrapolado a valores superiores.

En cada celda de la tabla 4.3 se indican los siguientes resultados: en la parte superior el porcentaje de aciertos y en la parte inferior la distancia media entre la solución óptima y la subóptima dada en tanto por ciento sobre el coste de la solución

óptima, como se indica en la expresión 4.3.

$$\frac{\text{distancia solución óptima - subóptima}}{\text{coste solución óptima}} \times 100 \qquad (4.3)$$

La distancia entre la solución óptima y la subóptima es la diferencia de la carga (suma de los costes de los procesos asignados) de los procesadores con mayor carga en cada solución.

Los costes de los procesos son números aleatorios comprendidos entre 3 y 20. Estos límites han sido escogidos de manera que sean proporcionales a los tiempos de ejecución de los procesos que se pretende distribuir. En el capítulo 5 se mostrará una Red de Petri donde los costes de los procesos están comprendidos entre 1,8 y 12,1 respectivamente, y cuya relación corresponde con 3 y 20.

$$\frac{12,1}{1,8} \approx \frac{20}{3}$$

Tabla 4.3. Resultado de los ensayos HTF

p\m	2	3	4	5	6	7	8	9
3	100 0							
4	100 0	100 0						
5	57.6 2.45	100 0	100 0		optimabilidad (%) distancia óptima-subóptima (%)			
6	64.8 1.43	96 0.19	100 0	100 0				
7	33.6 2.77	52.8 3.33	97.2 0.17	100 0	100 0			
8	53.2 1.3	32.4 3.25	91.6 0.43	99.2 0.04	100 0	100 0		
9	29.2 2.24	47.6 2.05	52.8 2.91	91.6 0.41	99.6 0.02	100 0	100 0	

Tabla 4.3. Resultado de los ensayos HTF

p\m	2	3	4	5	6	7	8	9
10	60.8 0.85	26.8 3.09	33.6 4.11	90.8 0.48	96 0.23	99.6 0.02	100 0	100 0

Una ventaja adicional de este algoritmo es que su implementación es distribuida, cada procesador toma la decisión del proceso que debe ejecutar. No se requiere de un recurso dedicado para la implementación del algoritmo.

4.2.2. Algorimo Light Task First (LTF)

Como variante del algoritmo anterior se analiza un algoritmo que asigna el proceso con el coste más bajo al procesador con menor carga. Los diferentes pasos del algoritmo son:

1. Confeccionar la lista de procesos
2. Identificar proceso de menor coste
3. Si no existe --> FIN
4. Identificar procesador con menor carga
5. Asignar proceso a procesador
6. Retirar proceso de la lista de procesos
7. Volver al paso 2

En la tabla 4.4 se muestran los resultados de los ensayos realizados para diferente número de procesos y procesadores. Los costes de los procesos estaban comprendidos entre 3 y 20.

Tabla 4.4. Resultado de los ensayos LTF

p\m	2	3	4	5	6	7	8	9
3	18 8.04							
4	38.8 0.29	2.4 13.21						

Tabla 4.4. Resultado de los ensayos LTF

p\m	2	3	4	5	6	7	8	9
5	0 11.12	5.2 7.75	0 18.07		optimabilidad (%) distancia óptima-subóptima (%)			
6	11.2 3.39	14 4.96	2 11.86	0 19.8				
7	0 10.71	0 16.48	2.4 7.76	0 13.88	0 22.1			
8	6.8 3.04	0.4 9.98	5.2 5.95	0 10.6	0 16.46	0 23.2		
9	0 9.48	2 5.16	0 19.85	0.4 8.63	0 12.96	0 17.55	0 23.08	
10	4.8 2.59	0 15.49	0 14.14	2 6.87	0 10.42	0 14.74	0 19.34	0 24.51

4.3. La función coste

En los algoritmos expuestos en el punto 4.2 se supone conocido el tiempo de ejecución de cada proceso o un orden entre ellos función de su tiempo de ejecución.

La estimación del tiempo de ejecución de este tipo de procesos es difícil por no decir imposible debido a las sentencias de control de flujo existentes (*while(condición)*, *if(condición)*, ...). El comportamiento del proceso no se puede predecir pues las condiciones que se evalúan son dependientes del estado de otros parámetros en el momento en el que el proceso es ejecutado.

El único dato con el que se puede trabajar es el tiempo de ejecución medio del proceso medido sobre varios ciclos de ejecución del proceso.

En este punto se tratará de encontrar un método que nos permita estimar el tiempo de ejecución de los procesos observando las tareas que acometen. Nos centraremos en el método denominado Coste Simbólico, que utiliza la información que proporcionan las expresiones simbólicas de los códigos de los programas. El tiempo de ejecución de un proceso será la suma de los costes asignados a cada expresión simbólica que componga el proceso [SAR 89]. Con este fin se distinguen dos tipos de procesos:

Tipo 1.- Sensibilización: Evalúan las condiciones del predicado y la condición temporal para determinar el token sensibilizador y el tiempo de disparo de una transición.

Tipo 2.- Disparo: Llevan a cabo las tareas indicadas en el resultado de la acción de una transición.

4.3.1. Procesos de sensibilización (tipo 1)

Se intentará obtener un orden entre los procesos observando la sintaxis del predicado que tiene cada transición.

Partimos de la suposición que cuanto más complicada sea la evaluación de la condición y cuantas más condiciones deban evaluarse mayor será el coste temporal del proceso. De esta forma, a cada tipo de condición se le asignará un coste temporal, siendo el coste del proceso la suma de los costes de las condiciones que lo forman.

El coste de la evaluación de las condiciones del predicado es función más del tipo de lugar de la Red de Petri sobre el que se evalúa que del tipo de condición. La evaluación de una condición sobre un lugar no seguro tiene un coste bastante superior al correspondiente sobre un lugar seguro.

Se supondrá que el diseñador del modelo de la Red de Petri tiene conocimiento de si un lugar es seguro o no, y esta información se proporcionará al simulador cuando se introduzca el grafo. Esta información permitirá obtener una estimación más acertada sobre el comportamiento del proceso. En caso de que no se pueda determinar si un lugar es seguro, se fijará como que no lo es.

En la tabla 4.5 se muestran las diferentes situaciones que pueden aparecer al evaluar las condiciones del predicado en una transición con más de un lugar de entrada. También se indica los diferentes tipos de condiciones temporales. En ningún caso se pretende limitar las condiciones a las allí listadas.

Tabla 4.5. Condiciones del predicado de una transición

Lugares de entrada	
(tipo A)	*todos los lugares seguros*
(tipo B)	*todos los lugares no seguros*
(tipo C)	*lugares de ambos tipos*
Condición temporal	
(tipo Z)	*L.chronos=L'.chronos+@*
(tipo X)	*L.chronos=min(L'.chronos)+@*
(tipo S)	*L.chronos=min(L'.chronos[subcondición])+@*
(tipo Y)	*L.chronos=max(L'.chronos,L''.chronos)+@*
(tipo T)	*L.chronos=max(L'.chronos,min(L''.chronos))+@*
(tipo R)	*L.chronos=L'.chronos+v.a.*

La condición temporal tipo S establece una subcondición en una condición mediante la siguiente sintaxis *condición[subcondición]*. La *condición* será evaluada sobre los elementos que validen la *subcondición*.

La evaluación de la condición temporal se realizará exclusivamente sobre los token que validan el predicado. La excepción la constituye la condición temporal tipo S cuya evaluación se realiza sobre los token que validan la subcondición. Ahora bien, esto sólo se realiza a efectos de determinar el chronos de los token de salida. Los token consumidos por el disparo de la transición (token de entrada) siempre serán aquellos que validen el predicado.

Como primera aproximación puede afirmarse que

$$\text{coste(A)} < \text{coste(B)} < \text{coste(C)} \tag{4.4}$$

$$\text{coste(Z)} < \text{coste(R)} < \text{coste(Y)} < \text{coste(X)} < \text{coste(S)} < \text{coste(Z)} \tag{4.5}$$

Con el objeto de verificar las expresiones 4.4 y 4.5 se han medido algunos procesos de sensibilización mediante el programa "Turbo Profiler" de Borland . Los resultados se muestran en la tabla 4.6. Los tiempos están dados en microsegundos.

En las dos primeras columnas se indica el proceso y su tiempo de ejecución medio medido.

En las dos siguientes columnas se muestra información relativa al predicado de la transición. En la primera de estas columnas se indica el tipo de lugar de entrada de la transición en referencia a los definidos en la tabla 4.5. En la segunda columna se indica el tiempo de ejecución medio para la evaluación de las condiciones del predicado sobre los lugares de entrada.

Tabla 4.6. Información de los procesos

Proceso	**tiempo ejecuc. medio (µs)**	**Predicado**		**Condición temporal**	
		Tipo	**tpo. ejecuc. medio (µs)**	**Tipo**	**tpo. ejecuc. medio (µs)**
1	5.7	C	5.67	Z	0.42
2	6.1	C	6.1	Z	0.42
3	4.6	C	4.6	Z	0.42
4	6.1	C	5.86	X	1.7
5	5	B	4.54	X	1.5
6	6.3	C	5.97	X	1.6
7	6.5	B	5.95	X	1.5
8	5.2	B	5.2	Z	0.42
9	1.8	A	1.78	Y	0.9
10	2.2	A	1.9	R	0.6
11	5.3	C	5.16	S	6.9
12	2.2	A	2.2	T	11.8
13	1.8	A	1.79	R	0.8

En las dos últimas columnas se muestra información relativa a la condición temporal presente en la acción de la transición. En la primera columna se indica el tipo de condición temporal de acuerdo a los definidos en la tabla 4.5. En la segunda columna se indica el tiempo de ejecución medio para evaluar la condición temporal.

La principal conclusión que obtenemos del estudio de la tabla es la poca incidencia del tiempo de evaluación de la condición temporal sobre el tiempo total de ejecución de la función. Esto es obvio puesto que la ejecución del proceso no implica la evaluación de la condición temporal, tan sólo se evalúa cuando la transición esta sensibilizada. Este es el motivo de la existencia de tiempos de evaluación de la condición temporal del orden de decenas de microsegundos en procesos cuya duración es inferior, y que ponen en evidencia la poca incidencia de este tiempo en el del proceso.

Considerando tan sólo la información referente al predicado se observa que la expresión 4.4 es una buena aproximación.

Con las expresiones 4.4 y 4.5 se cometen errores pues el tiempo de ejecución de los procesos 1, 3 y 11 del tipo C son menores que el proceso 7 del tipo B, pero a pesar de ello el orden obtenido parece adecuado a nuestros propósitos.

4.3.2. Procesos de disparo (tipo 2)

Se intentará obtener un orden entre los procesos observando las condiciones que componen el resultado de la acción de la transición a la que el proceso se encuentra ligado.

El coste de la evaluación de las condiciones del resultado son función del número y tipo de éstas, así como, al igual que los procesos de tipo 1, función del tipo de lugar de la Red de Petri sobre el que se evalúa. De esta forma, tenemos los siguientes condicionantes (se supone que los lugares de una Red de Petri son implementados mediante matrices de una dimensión):

Número de condiciones: cuanto mayor sea el número de tareas que debe realizar el disparo de una transición, mayor será su duración.

Condición temporal con variable aleatoria: su existencia implica la realización de otros procesos para calcular el valor de la variable aleatoria.

Lugar de salida no seguro y distinto del de entrada: implica buscar una posición para los token de salida. En el caso de que ambos lugares

coincidan (transición con el mismo lugar de entrada y salida), el token de salida se ubica en el lugar del token de entrada.

Lugar de entrada no seguro: se necesita localizar la tupla sensibilizadora.

Lugar de entrada distinto del de salida: al sacar el token del lugar de entrada se necesita asignar un valor al hueco dejado por el token que indique que ya no existe token.

En la tabla 4.7 se ilustran algunas características de una serie de procesos de disparo. L_e y L_s denotan los lugares de entrada y salida de la transición respectivamente.

Tabla 4.7. Información de los procesos de disparo

Proceso	tiempo ejecuc. medio (μs)	Número de condi-ciones	Cond. temp. tipo R	L_e ($\neq L_s$)	L_s no seguro ($\neq L_e$)	L_e no seguro	Suma
1	12.1	14			2	1	21
2	6.5	9			1	1	13
3	6.5	9			1	1	13
4	8.2	9			1	1	13
5	3	4				1	5
6	8.2	9			1	1	13
7	3	4				1	5
8	2.5	5				1	6
9	4.4	5		1	1		9
10	3	3	1	1			5
11	3.9	6				1	7
12	2.9	5		2		1	8
13	10.4	6	1	1	1		10

En las dos primeras columnas se indica el proceso y su tiempo de ejecución medio.

En las cinco siguientes columnas se muestra información relativa a las condiciones del resultado de la acción de la transición. En la primera de estas columnas se indica el número de condiciones de que consta el resultado. En la segunda se indica si el tiempo de disparo viene determinado por una variable aleatoria. En la tercera el número de lugares de entrada de cada transición que no son de salida. En la cuarta la cantidad de lugares de salida no seguros, y que no son al mismo tiempo lugar de entrada para la transición. Por último los lugares de entrada que no son seguros.

Finalmente, se suman los valores contenidos en las columnas 3 a 7. Para ello, la columna 6 tiene un peso de valor 3, que representa la cantidad de celdas que se han de recorrer en la matriz que implementa el lugar de la Red para encontrar el token deseado.

Con los valores obtenidos en la columna 8 se puede establecer el siguiente orden entre los procesos.

$$P\_1 > (P\_2=P\_3=P\_4=P\_6) > P\_13 > P\_9 > P\_12 > P\_11 > (P\_8 > P\_10=P\_5=P\_7)$$

donde P_m representa al proceso *m*.

Se observa un error en la posición que ocupa el proceso 13, debido a que este proceso recorre 9 celdas de la matriz para localizar un token (tres veces superior al peso otorgado a la columna 6). Lo que ocasiona que se le estime un tiempo de ejecución inferior al real.

4.4. Conclusiones

La distribución de los procesos con las características expuestas en 4.1 es un problema NP-completo. Como alternativa a realizar un recorrido por la totalidad del espacio de soluciones en busca de la solución óptima, existen técnicas para localizar una solución subóptima con un coste computacional muy razonable.

Las conclusiones obtenidas de los algoritmos de distribución ensayados son:

High Task First

- Tiene valores muy grandes de optimabilidad para un número de procesos menor o igual que el doble de la cantidad de procesadores, siendo la diferencia entre la solución óptima y la subóptima practicamente nula.
- La optimabilidad disminuye si aumentan los procesos para un número de procesadores fijo. Sin embargo, aparecen máximos intermedios cuando la cantidad de procesos es múltiplo del número de procesadores.
- La distancia entre la solución óptima y la subóptima aumenta según crece el número de procesos para una cantidad de procesadores determinada. Este comportamiento mejora cuando el número de procesos y procesadores son múltiplos.
- Aunque en casos muy concretos la optimabilidad toma valores bajos, la diferencia entre la solución óptima y subóptima es muy baja en todos los casos estudiados.

Light Task First

- Los valores de optimabilidad son bastante inferiores a los obtenidos con el algoritmo anterior.
- La distancia entre la solución óptima y la subóptima es superior a la obtenida en el mismo caso con el algoritmo anterior.
- El comportamiento de ambos parámetros cuando el número de procesos y procesadores varían es similar al expuesto con el algoritmo HTF.

Una ventaja adicional de ambos algoritmos es que su tiempo de computación es sólo el requerido para ordenar los procesos de acuerdo a una prioridad.

La estimación del tiempo de ejecución de este tipo de procesos es difícil debido a que su comportamiento depende del estado de la Red en el momento en el que el proceso es ejecutado. Se ha desarrollado un método para obtener un orden entre los procesos según el tipo de éstos:

Procesos de sensibilización

- Se ordenan según la relación indicada en 4.4.

Procesos de disparo

- Se ordenan según los valores proporcionados por la expresión 4.6.

$$\eta_1 + \eta_2 + \eta_3 + (3 \times \eta_4) + \eta_5 \qquad (4.6)$$

donde

η_1: es el número de condiciones que componen el resultado de la acción de la transición. Debe considerarse que alguna condición puede tener un mayor o menor peso en esta contabilidad.

η_2: toma valor 1 si la duración del disparo de la transición viene determinada por una variable aleatoria.

η_3: es el número de lugares de salida que no son seguros y además son distintos a los lugares de entrada.

η_4: es el número de lugares de entrada que no son seguros.

η_5: es el número de lugares de entrada que son distintos a los de salida.

Capítulo 5: Prestaciones del simulador paralelo

5. Prestaciones del simulador paralelo

El algoritmo de simulación paralela expuesto en el capítulo 3 fue utilizado en el desarrollo de un simulador de Redes de Petri temporizadas. El simulador fue ensayado con un modelo de una red local con protocolo CSMA/CD con un alto paralelismo y gran carga computacional que pusiera de relieve las mejoras obtenidas con respecto a una simulación centralizada.

5.1. Implementación de los modelos

Los modelos se describirán en función de las entidades que los componen y de las relaciones entre las entidades. Las entidades del modelo se implementan del modo descrito:

Token: se define una estructura denominada TOKEN compuesta por los identificadores que se definan.

Lugar: es implementado mediante la estructura denominada PLACE compuesta de un vector de tamaño fijo debido a la dificultad de manejar memoria dinámica. El tamaño viene determinado por el parámetro *maxtok_x_place*. Las celdas del vector PLACE se definen como estructuras tipo TOKEN.

Transiciones: se implementa mediante una estructura denominada TRANSICION compuesta por dos punteros a función, el campo chronosfire y el vector tupla_enabled.

Los punteros apuntan a las funciones Sensiblem y Disparam, que comprueba si la transición está sensibilizada y ejecuta el disparo respectivamente. El campo chronosfire almacena el tiempo de disparo de la transición. El vector tupla_enabled contiene la tupla sensibilizadora de la transición. Su tamaño viene determinado por el parámetro *maxtok_x_tran*.

Las relaciones entre las entidades se definen mediante las siguientes funciones:

Sensiblem comprueba si la transición t_m se encuentra sensibilizada mediante la evaluación del predicado de la transición. En caso afirmativo inicia cada uno de los campos de la estructura TRANSICION y añade la transición a la LFD.

Disparam realiza las tareas indicadas en el resultado de la transición t_m, utilizando la información contenida en los campos de la estructura TRANSICION.

Se define una función Sensible*m* y otra Dispara*m* por cada transición del modelo.

Los token de un mismo lugar se almacenarán de forma consecutiva en los índices inferiores del vector PLACE. Esta forma de proceder evita que se recorra la totalidad del vector en busca de un token en particular, estando limitada la búsqueda entre las filas 0 y aquella con el valor Notoken (Notoken es el valor con el que se inician las celdas e indica celda vacía).

La aparición de celdas vacías (con el valor Notoken) en el rango anterior debido al consumo de token como consecuencia de un disparo, puede conducir a resultados no correctos al interpretarse dicho hueco como el fin del conjunto de token existentes en el lugar. Para evitarlo, periódicamente se debe comprobar la existencia de celdas vacías en el vector, en cuyo caso se trasladarán los token existentes en celdas superiores a los huecos existentes. Esta tarea es realizada mediante la función denominada **Quitarhuecos**.

Si un lugar es seguro no existe la posibilidad de formación de huecos.

5.2. Simulador

El simulador ha sido implementado en lenguaje C, y su diagrama de flujo es mostrado en la figura 5.1.

El simulador comienza calculando las distancias temporales entre las distintas transiciones que componen el modelo, iniciando los punteros a función, asignando a la Red su marcado inicial e iniciando algunos parámetros del modelo como duración del disparo de cada una de las transiciones, número máximo de token por lugar y otros.

A continuación se inicia la LFD localizando el conjunto de transiciones sensibles del modelo mediante la función Sensible correspondiente (cada transición tiene un

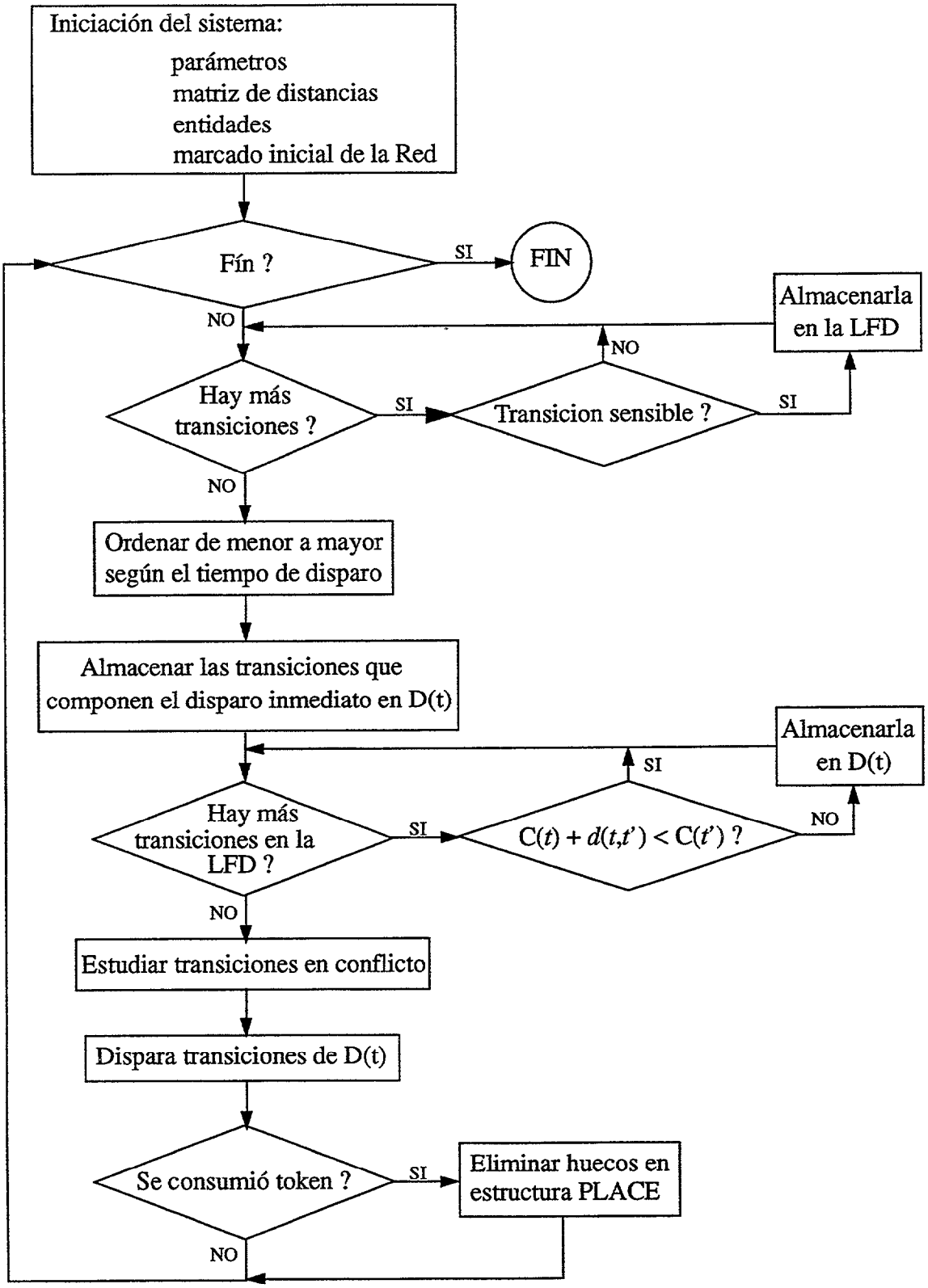


Figura 5.1. Diagrama de flujo del simulador

campo con un puntero iniciado a la función Sensible que le corresponde).

Para cada transición sensibilizada se almacena el número de la transición en la LFD, y al mismo tiempo se actualizan los campos de la estructura TRANSICION de la transición localizada con su tiempo de disparo y la tupla sensibilizadora.

El conjunto de transiciones sensibilizadas obtenido anteriormente se ordena de menor a mayor según su tiempo de disparo mediante un algoritmo de ordenación de complejidad $0(n^2)$.

Las transiciones mutuamente independientes se almacenarán en un vector denominado TRANDISPARABLES. Este conjunto se inicia con la transición o las transiciones con menor tiempo de disparo (primeros elementos de la LFD).

A continuación se estudian las relaciones de causalidad existentes entre las transiciones de la LFD evaluando la condición 2.7. De manera que una transición sensibilizada *t* se añadirá a TRANDISPARABLES si no existe ninguna transición con un tiempo de disparo menor que C(t) y con una relación causa-efecto dirigida hacia la transición en estudio *t*.

Obtenido el conjunto de transiciones mutuamente independientes, se comprueba si alguna de las transiciones se encuentra en conflicto. En caso de conflicto entre dos ó más transiciones, se debe disparar la transición con menor tiempo de disparo y el resto de transiciones se eliminan de TRANDISPARABLES.

Se procede al disparo de las transiciones que componen TRANDISPARABLES llamando a la función Disparo correspondiente a cada transición.

Finalmente, se eliminan los huecos en todos aquellos lugares donde hubo consumo de token mediante la función Quitarhuecos. A la función Quitarhuecos se le pasa como parámetro el lugar sobre el que debe actuar. Y el proceso vuelve a iniciarse localizando el conjunto de transiciones sensibles para recomponer la LFD.

5.2.1. Variables aleatorias

El simulador proporciona una serie de variables aleatorias que pueden ser empleadas por el modelo.

Para su generación se hace uso de la función de distribución F(x), la cual es continua y uniformemente distribuida de 0 a 1.

Se expondrá la obtención de la variable aleatoria con una distribución exponencial, utilizada en el modelo sobre el que el simulador fue ensayado. La distribución exponencial es la indicada en 5.1.

$$F(x) = 1 - e^{-x/\mu} \qquad x \geq 0 \qquad (5.1)$$

Si y=F(x), entonces obtenemos la expresión 5.2.

$$x = F^{-1}(y) = -\mu \cdot \ln(1-y) \qquad (5.2)$$

La expresión de x tiene una distribución exponencial y además (1-y) está uniformemente distribuida, luego la variable aleatoria X expresada mediante la relación 5.3 está distribuida exponencialmente con media μ.

$$X = -\mu \cdot \ln(U) \qquad (5.3)$$

Los valores para U se obtienen mediante un generador de números aleatorios uniformes entre 0 y 1 [KNU 81] que implementa un procedimiento de realimentación aditivo no lineal. Utiliza un vector de 55 enteros de 32 bits para proporcionar números seudoaleatorios en el rango 0 a 2^{32}-1. Estos números son luego divididos por el rango 2^{32}. El periodo del generador es aproximadamente $16*(2^{31}-1)$.

5.3. Simulación paralela de un modelo de protocolo CSMA/CD

El simulador fue ensayado con un modelo de red local con protocolo CSMA/CD representado en la figura 5.2. No se ha sido muy riguroso en el modelado del protocolo CSMA/CD, recogiendo tan sólo sus características más importantes.

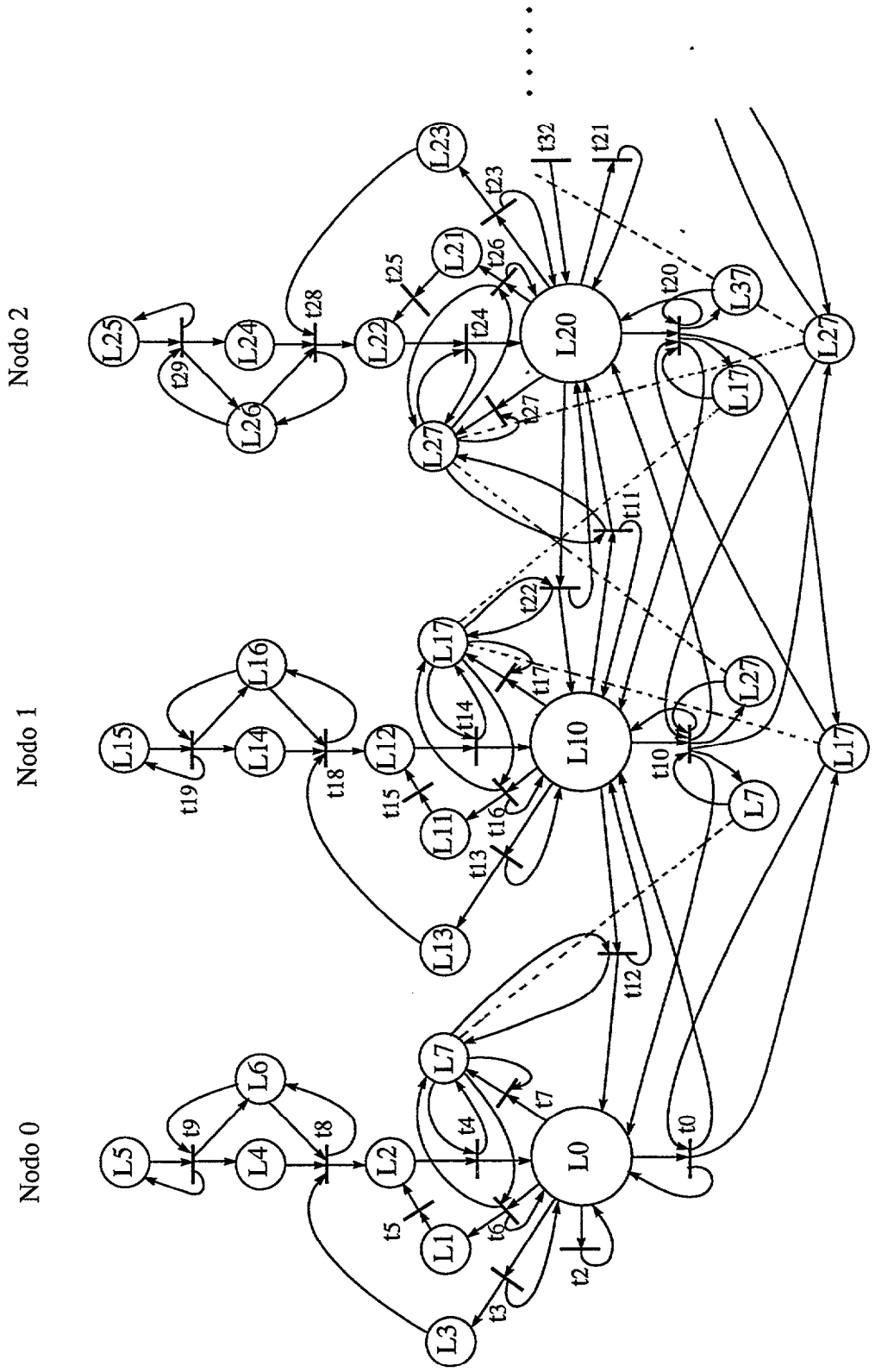


Figura 5.2. Modelo de una red local con protocolo CSMA/CD

Para el modelado se ha utilizado una Red de Petri de alto nivel temporizada con una ejecución que responda a un modelo de tiempo fuerte.

5.3.1. Descripción del modelo

El modelo de la red local comprende 10 nodos representando 10 estaciones ethernet interconectadas.

En cada nodo existe un usuario que genera mensajes con una cadencia determinada mediante una variable aleatoria exponencial.

Los parámetros del modelo se indican en la tabla 5.3.

Tabla 5.3. Parámetros del modelo

N° de estaciones	10
Distancia entre nodos	constante
Tamaño de la trama de transmisión	constante
Tamaño de las colas de estación	10 mensajes
Errores de transmisión	sólo por colisión
Límite de intentos	infinito
Límite algoritmo back-off	var. aleat. exponencial
Cadencia de generación de mensajes	var. aleat. exponencial

Toda la descripción del modelo se realiza en el tiempo. Sólo se ha tenido en cuenta las relaciones y magnitudes temporales del sistema. Las distancias entre estaciones se han transformado en retardos de propagación de la señal.

El modelo es muy simplificado para facilitar su implementación en el simulador, así como su representación gráfica.

Las estaciones generan y transmiten mensajes sin un destino determinado. La transmisión de un mensaje se considera correcta si no se detecta colisión antes de finalizar su transmisión. De esta forma no es necesario implementar un mecanismo para el

envío de un asentimiento por cada mensaje correctamente recibido y el modelo de la estación queda más reducido.

En la figura 5.4 se muestra el diagrama de estados de una estación en el momento de la contienda por el canal (se ha excluido la generación de mensajes).

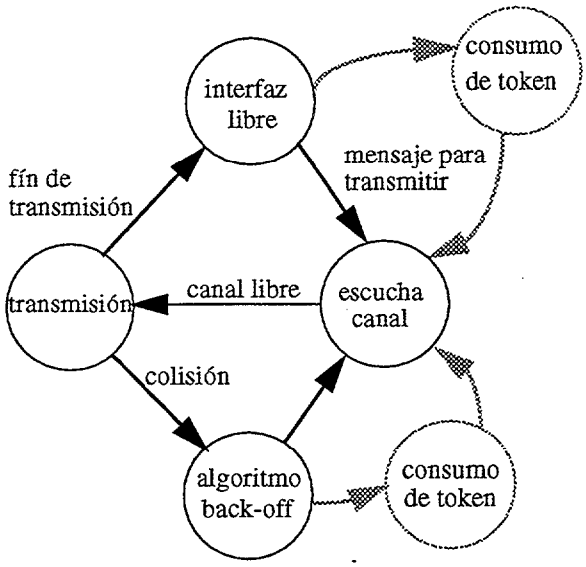


Figura 5.4. Diagrama de estados de la estación

Cuando la estación dispone de un mensaje para transmitir pasa al estado *escucha canal.* Si el canal está libre se inicia la transmisión pasando al estado *transmisión.* Una vez finalizada la transmisión y si ésta es correcta se pasa al estado *interfaz libre*, y al disponer de un mensaje para transmitir se vuelve al estado *escucha canal.*

Si durante la transmisión se detecta una colisión se aborta la transmisión y se pasa al estado *algoritmo back-off.* Finalizado el tiempo de espera del algoritmo se vuelve al estado *escucha canal* para volver a transmitir el mensaje.

La implementación realizada de la red mediante una Red de Petri obliga a pasar por un estado intermedio *consumo de token* antes de llegar al estado *escucha canal.* En este estado se limpia el canal mediante el consumo de todos los token existentes en el canal que han sido generados por la estación transmisora.

La Red de Petri de cada nodo se compone de 8 lugares y 10 transiciones, indica-

dos en la tabla 5.5 y 5.6 respectivamente.

Tabla 5.5. Relación de lugares en cada nodo (x=nodo)

L_{x0}	Transmisión - Canal
L_{x1}	Colisión - algoritmo de back-off
L_{x2}	Espera canal libre
L_{x3}	Interfaz libre para transmisión
L_{x4}	Pila de mensajes (máximo 10)
L_{x5}	Usuario
L_{x6}	Cuenta de mensajes en L_{x4}
L_{x7}	Cuenta de mensajes transmitiéndose en el canal

Tabla 5.6. Relación de transiciones en cada nodo (x=nodo)

t_{x0}	Propagación del mensaje a los nodos contiguos
t_{x1}	Propagación del mensaje al nodo derecho
t_{x2}	Propagación del mensaje al nodo izquierdo
t_{x3}	Fin de transmisión del mensaje
t_{x4}	Inicio de transmisión del mensaje
t_{x5}	Tiempo de espera algoritmo de back-off
t_{x6}	Detección de colisión
t_{x7}	Consumo de token
t_{x8}	Selección del mensaje a transmitir
t_{x9}	Nuevo mensaje

Los identificadores definidos para los token, con su rango de valores indicado son:

<función> : { STX, MTX, ETX, NUL }
<origen> : { 0, ... , nº de nodos-1 }
<cantidad>: $\{N^+ \cup 0\}$
<chronos> : $\{ \Re^+ \cup 0 \}$

El identificador <funcion> se utiliza para indicar la parte del mensaje que es objeto de la transmisión. STX, MTX y ETX hacen referencia a la cabecera, datos y fin del mensaje respectivamente. NUL es un valor de carácter general utilizado cuando el token no representa un mensaje.

El nodo generador del mensaje se indica mediante el identificador <origen>. El identificador <cantidad> almacena la cantidad de token en un lugar concreto. El identificador <chronos> indica el instante de tiempo de creación del token.

En el lugar L_{x5} siempre residirá un token que representa a un usuario que accede a la estación para transmitir un mensaje. Los mensajes a transmitir se almacenarán en el lugar L_{x4} que actuará como una cola con una política FIFO. Para evitar el crecimiento incontrolado del número de tokens en L_{x4}, y por otra parte para modelar las limitaciones de memoria de los terminales conectados a una red local, se ha impuesto un tamaño de la cola de 10 tokens.

En el lugar L_{x6} reside siempre un token con los identificadores <cantidad> y <chronos> que lleva la cuenta de token en el lugar L_{x4}. Cada nuevo mensaje que se incorpora a la cola aumenta en una unidad el valor del identificador <cantidad> del token que reside en L_{x6}. Esta cantidad se decrementa cuando se extrae un elemento de la cola. El identificador <cantidad> permite conocer el estado de la cola de mensajes. El identificador <chronos> se actualiza cada vez que se altera el valor de <cantidad> con el objeto de conocer el instante de tiempo en el que existe un hueco disponible en la cola.

De esta forma, se generan mensajes (disparo de la transición t_{x9}) siempre que la cola no esté llena (valor de <cantidad> menor que 10), y con un instante de tiempo correspondiente al máximo valor entre el instante de tiempo en el que se genera el nuevo mensaje para transmitir (valor de <chronos> del token de L_{x5}) y el instante de tiempo en el que existe un hueco en la cola (valor de <chronos> del token de L_{x6}).

Cuando el interfaz del nodo queda libre (token en el lugar L_{x3}) se selecciona el primer mensaje de la cola, y se escucha el canal L_{x0} en espera de que se encuentre vacío

para comenzar la transmisión y evitar las colisiones. Al igual que ocurre con el lugar L_{x6}, el lugar L_{x7} contiene un token con los identificadores <cantidad> y <chronos> que almacena la cantidad de token existentes en L_{x0}, es decir, el número de mensajes que el nodo *x* escucha en el canal. Tanto la transmisión de mensajes del nodo *x* como los mensajes originados por otros nodos de la red que alcanzan el lugar L_{x0}, hace aumentar el valor de <cantidad> del token en L_{x7}. El identificador <chronos> se actualiza cada vez que se altera el valor de <cantidad> con el objeto de conocer el instante de tiempo en el que el canal queda vacío.

Cuando el canal está vacío (identificador <cantidad> del token en L_{x7} con valor cero) se inicia la transmisión de la cabecera del mensaje (disparo de t_{x4}, depositando un token en L_{x0} con <funcion=STX>). El mensaje se propaga a ambos lados de la red (el disparo de la transición t_{x0} propaga la cabecera del mensaje hacia los nodos adyacentes *x+1* y *x-1*, y simultáneamente genera un token en L_{x0} con <función=MTX> indicando la transmisión de datos por el nodo *x*. Las transiciones $t_{(x+1)1}$ y $t_{(x-1)2}$ propagan el mensaje hacia el resto de la Red).

La finalización de la transmisión del mensaje puede venir dada como consecuencia de la finalización del mensaje (disparo de t_{x3}), o de manera anormal como consecuencia de una colisión (disparo de t_{x6}). De cualquier forma se finaliza la transmisión (el disparo de la transición genera un token en L_{x0} con <función=ETX>. El fin de mensaje es propagado del mismo modo que la cabecera del mensaje pero con la diferencia que el token generado en L_{x0} tiene <función=NUL>).

El token de L_{x0} con <función=NUL> es consumido por la transición t_{x7}, disminuyendo al mismo tiempo el valor de <cantidad> en una unidad. (La transición t_{x7} sería innecesaria si existiera un sistema paralelo al implementado mediante t_{x0}, t_{x1} y t_{x2}, que propagara los token con <función=ETX> a los nodos adyacentes sin generar un token en los lugares de entrada de las transiciones).

Para que el nodo *x* detecte una colisión se deben cumplir dos condiciones: el nodo *x* debe estar transmitiendo, y la transmisión de un mensaje por otro nodo distinto debe alcanzar nuestro nodo *x*, representado por el lugar L_{x0} (la duración de la transición t_{x3} -fin de transmisión del mensaje- es suficientemente grande comparado con el tiempo necesario para recorrer la Red, de manera que si el mensaje transmitido por un nodo colisiona, el nodo recibirá conocimiento de la colisión antes de finalizar la transmisión). En estas circunstancias el número de token en el lugar L_{x0} será mayor o igual que dos: un token con <función=MTX> y <origen=*x*>, y otros token con <función=MTX> y

<origen=y>, donde $y \neq x$ (la duración de la transición t_{x6} se ha elegido mayor que la correspondiente a t_{x0}, de manera que nunca se podrá detectar una colisión nada más comenzar la transmisión de la cabecera del mensaje, siempre se habrá comenzado a transmitir datos).

En el otro extremo el nodo *y* también detectará del mismo modo una colisión, finalizando su transmisión mediante el envío de un token con <función=ETX> al resto de los nodos. En consecuencia, en L_{x0} se observará un token con <función=ETX> y <origen=*y*> por cada token con <función=MTX> y <origen=*y*> previamente existente en el lugar. La existencia de esta pareja de token sensibiliza la transición t_{x7} y sus disparos se sucederán hasta la total limpieza del canal. Similar comportamiento tendrá el nodo *y*.

Detectada la colisión, el nodo aguarda un tiempo de espera (lugar L_{x1}) antes de volver a intentar transmitir el mensaje. Este ciclo es repetido hasta que el mensaje es correctamente transmitido.

El comportamiento de cada estación explicado hasta ahora se debe reflejar más formalmente en las acciones ligadas a cada transición.

t_0 = { <<L_0,L_{17}>,<L_0,L_{10},L_{17}>> | L_0.origen=0 & L_0.funcion=(STX || ETX) =>
L_{10}.funcion=L_0.funcion &
if L_0.funcion=STX then L'_0.funcion=MTX
if L_0.funcion=ETX then L'_0.funcion=NUL
L'_0.origen=L_{10}.origen=L_0.origen & L'_{17}.cant=L_{17}.cant+1 &
L'_0.chronos=L_{10}.chronos=L'_{17}.chronos=L_0.chronos+$chrt_0$ }

t_{x0} = { <<$L_{(x-1)7}$,L_{x0},$L_{(x+1)7}$>,<$L_{(x-1)7}$,$L_{(x-1)0}$,L_{x0},$L_{(x+1)0}$,$L_{(x+1)7}$>> |
L_{x0}.origen=x & L_{x0}.funcion=(STX || ETX) =>
$L_{(x-1)0}$.funcion=$L_{(x+1)20}$.funcion=L_{x0}.funcion &
if L_{x0}.funcion=STX then $L_{(x-1)0}$.funcion=$L_{(x+1)0}$.funcion=MTX
if L_{x0}.funcion=ETX then $L_{(x-1)0}$.funcion=$L_{(x+1)0}$.funcion=NUL
$L_{(x-1)0}$.origen=L'_{x0}.origen=$L_{(x+1)0}$.origen=L_{x0}.origen &
$L_{(x-1)0}$.chronos=L'_{x0}.chronos=$L_{(x+1)0}$.chronos=
=$L_{(x-1)7}$.chronos=$L_{(x+1)7}$.chronos=L_{x0}.chronos+$chrt_0$ &
$L'_{(x-1)7}$.cant=$L_{(x-1)7}$.cant+1 & $L'_{(x+1)7}$.cant=$L_{(x+1)7}$.cant+1 }

x ∈ {1,2,3,4,5,6,7,8}

t_{90} = { <<L_{87},L_{90}>,<L_{80},L_{87},L_{90}>> | L_{90}.origen=9 & L_{90}.funcion=(STX || ETX) =>
L_{80}.funcion=L_{90}.funcion &
if L_{90}.funcion=STX then L'_{90}.funcion=MTX
if L_{90}.funcion=ETX then L'_{90}.funcion=NUL
L_{80}.origen=L'_{90}.origen=L_{90}.origen & L'_{87}.cant=L_{87}.cant+1 &
L_{80}.chronos=L'_{90}.chronos=L'_{87}.cant=L_{90}.chronos+$chrt_0$ }

t_{x1} = { <L_{x0},$L_{(x+1)7}$>,<L_{x0},$L_{(x+1)0}$,$L_{(x+1)7}$>> | L_{x0}.origen<x & L_{x0}.funcion=(STX || ETX) =>
$L_{(x+1)0}$.funcion=L_{x0}.funcion &
if L_{x0}.funcion=STX then L'_{x0}.funcion=MTX
if L_{x0}.funcion=ETX then L'_{x0}.funcion=NUL
$L_{(x+1)0}$.origen=L'_{x0}.origen=L_{x0}.origen & $L'_{(x+1)7}$.cant=$L_{(x+1)7}$.cant+1 &
$L_{(x+1)0}$.chronos=L'_{x0}.chronos=$L'_{(x+1)7}$.chronos=min(L_{x0}.chronos)+$chrt_1$ }
x ∈ {1,2,3,4,5,6,7,8}

t_{91} = { <L_{90},L_{90}> | L_{90}.origen<9 & L_{90}.funcion=(STX || ETX) =>
if L_{90}.funcion=STX then L'_{90}.funcion=MTX
if L_{90}.funcion=ETX then L'_{90}.funcion=NUL
L'_{90}.origen=L_{90}.origen & L'_{90}.chronos=min(L_{90}.chronos)+$chrt_1$ }

t_2 = { <L_0,L_0> | L_0.origen>0 & L_0.funcion=(STX || ETX) =>
if L_0.funcion=STX then L'_0.funcion=MTX
if L_0.funcion=ETX then L'_0.funcion=NUL
L'_0.origen=L_0.origen & L'_0.chronos=min(L_0.chronos)+$chrt_2$ }

t_{x2} = { <<$L_{(x-1)7}$,L_{x0}>,<$L_{(x-1)0}$,$L_{(x-1)7}$,L_{x0}>> | L_{x0}.origen>x & L_{x0}.funcion=(STX || ETX) =>
$L_{(x-1)0}$.funcion=L_{x0}.funcion &
if L_{x0}.funcion=STX then L'_{x0}.funcion=MTX
if L_{x0}.funcion=ETX then L'_{x0}.funcion=NUL
$L_{(x-1)0}$.origen=L'_{x0}.origen=L_{x0}.origen & $L'_{(x-1)7}$.cant=$L_{(x-1)7}$.cant+1 &
$L_{(x-1)0}$.chronos=L'_{x0}.chronos=$L'_{(x-1)7}$.chronos=min(L_{x0}.chronos)+$chrt_2$ }
x ∈ {1,2,3,4,5,6,7,8}

t_{x3} = { <L_{x0},<L_{x0},L_{x3}>> | L_{x0}.origen=x & L_{x0}.funcion=MTX =>
L'_{x0}.funcion=ETX & L_{x3}.funcion=NUL &
L'_{x0}.origen=L_{x3}.origen=L_{x0}.origen &
L'_{x0}.chronos=L_{x3}.chronos=L_{x0}.chronos+$chrt_3$ }
x ∈ {0,1,2,3,4,5,6,7,8,9}

t_{x4} = { <<L_{x2},L_{x7}>,<L_{x0},L_{x7}> | L_{x7}.cant=0 =>
L_{x0}.funcion=L_{x2}.funcion & L_{x0}.origen=L_{x2}.origen & L'_{x7}.cant=L_{x7}.cant+1 &
L_{x0}.chronos=L'_7.chronos=max(L_{x2}.chronos,L_{x7}.chronos)+$chrt_4$ }
x ∈ {0,1,2,3,4,5,6,7,8,9}

t_{x5} = { <L_{x1},L_{x2}> | =>
L_{x2}.funcion=STX & L_{x2}.origen=L_{x1}.origen &
L_{x2}.chronos=L_{x1}.chronos+var.aleat.expon. }
x ∈ {0,1,2,3,4,5,6,7,8,9}

t_{x6} = { <<L_{x0},L_{x7}>,<L_{x0},L_{x1}>> | L_7.cant>1 & L_{x0}.origen=x & L_{x0}.funcion!=(ETX&&NUL) =>
L_{x1}.funcion=NUL & L'$_{x0}$.funcion=ETX & L'$_{x0}$.origen=L_{x0}.origen &
L_{x1}.chronos=L'$_{x0}$.chronos=mín(L_{x0}.chronos[L_{x0}.origen!=x])+chrt$_6$}
x ∈ {0,1,2,3,4,5,6,7,8,9}

t_{x7} = { <<L_{x0},L_{x7}>,L_{x7}> | L_{x0}.funcion=NUL ||
L_{x0}.origen[L_{x0}.funcion=MTX]=L_{x0}.origen[L_{x0}.funcion=NUL] =>
if L_{x0}.origen[L_{x0}.funcion=MTX]=L_{x0}.origen[L_{x0}.funcion=NUL] then
L'$_{x7}$.cant=L_{x7}.cant-2 else L'$_{x7}$.cant=L_{x7}.cant-1 &
L'$_{x7}$.chronos=min(L_{x0}.chronos[L_{x0}.funcion=NUL])+chrt$_7$}
x ∈ {0,1,2,3,4,5,6,7,8,9}

t_{x8} = { <<L_{x3},L_{x4},L_{x6}>,<L_{x2},L_{x6}>> | =>
L_{x2}.funcion=STX & L_{x2}.origen=x & L'$_{x6}$.cant=L_{x6}.cant-1 &
L_{x2}.chronos=L_{x6}.chronos=max(L_{x3}.chronos,min(L_{x4}.chronos))+chrt$_8$ }
x ∈ {0,1,2,3,4,5,6,7,8,9}

t_{x9} = { <<L_{x5},L_{x6}>,<L_{x4},L_{x5},L_{x6}>> | L_{x6}.cant<10 =>
L_{x4}.funcion=L'$_{x5}$.funcion=L_{x5}.funcion & L'$_{x6}$.cant=L_{x6}.cant+1 &
L_{x4}.chronos=L'$_{x5}$.chronos=L_{x6}.chronos=
max(L_{x5}.chronos,L_{x2}.chronos)+var.aleat.expon. }
x ∈ {0,1,2,3,4,5,6,7,8,9}

Las constantes $chrt_n$ indicadas en las acciones se corresponden con la duración del disparo de la transición t_{xn}. Deben satisfacer las siguientes condiciones para garantizar un comportamiento del modelo de acuerdo al ya descrito en el punto 5.3.1.

$$\begin{aligned} &chrt_0 = chrt_1 = chrt_2 \\ &chrt_3 >> 2*chrt_0*(\text{n° de nodos}) \\ &chrt_6 > chrt_0 \end{aligned} \qquad (5.4)$$

5.3.1.1. Estudio de las transiciones en conflicto del modelo

Aunque el modelo presenta un elevado número de transiciones en conflicto estático, sólo las transiciones t_3 y t_6 de un mismo nodo pudieran verse afectadas por un conflicto dinámico.

Las condiciones exigidas en 5.4 garantizan que en el caso de estar sensibilizadas con la misma tupla sensibilizadora nunca podrán ser disparadas simultáneamente como se demuestra a continuación.

Supongamos que el nodo *m* del modelo detecta una colisión entre una transmisión del propio nodo y otra del nodo *n*. En el peor de los casos, el nodo *n* habría iniciado su transmisión en el mismo instante de tiempo que escucha la transmisión del nodo *m* (más tarde sería imposible pues el nodo *n* detectaría una transmisión en el canal), y este instante de tiempo viene dado por el inicio de la transmisión del mensaje por el nodo *m* más el tiempo de propagación del mensaje hasta el nodo *n*.

$$C(t_{n4}) = C(t_{m4}) + chrt_0 + chrt_1 * L \qquad (5.5)$$

siendo L el número de nodos existentes entre *m* y *n*, excluidos éstos.

El mensaje transmitido por el nodo *n* alcanzará el nodo *m* en el instante de tiempo

$$C(t_{n4}) + chrt_0 + chrt_2 * L \qquad (5.6)$$

Sustituyendo 5.5 en 5.6 obtenemos el momento en el que el nodo *m* detecta la colisión.

$$C(t_{m4}) + chrt_1 * L + chrt_2 * L + 2 * chrt_0 \qquad (5.7)$$

Para que las transiciones t_{m3} y t_{m6} sean disparadas simultáneamente deben tener el mismo tiempo de disparo.

$$C(t_{x3}) = C(t_{x6}) \qquad (5.8)$$

Introduciendo 5.7 en la expresión 5.8 se obtiene

$$C(t_{m4}) + chrt_0 + chrt_3 = C(t_{m4}) + chrt_1*L + chrt_2*L + 2*chrt_0 + chrt_6$$

teniendo en cuenta la primera condición de 5.4, la expresión anterior queda reducida a la expresión 5.9.

$$chrt_3 = chrt_0*(1+2*L) + chrt_6 \qquad (5.9)$$

condición incompatible con las enumeradas en 5.4 salvo para valores de $chrt_6$ muy elevados, caso absurdo en las estaciones reales. Siendo por tanto imposible que ambas transiciones se encuentren en conflicto dinámico.

5.3.2. Test del simulador

En la tabla 5.7 se indican los valores de los parámetros utilizados en las simulaciones.

Tabla 5.7. Valor de los parámetros

maxtok_x_place	15
maxtok_x_tran	5
media de la v.a.exp.	150
chrt0	5
chrt1	5
chrt2	5
chrt3	150
chrt4	3
chrt5	variable
chrt6	6
chrt7	1
chrt8	2
chrt9	variable

El modelo parte del marcado inicial indicado en la tabla 5.8, en el que el interfaz de cada nodo está libre, el canal y los buffer de entrada vacíos, y existe un usuario en cada nodo.

Para comprobar la validez del simulador se compararán los resultados obtenidos con los correspondientes a una simulación centralizada que siga las pautas del algoritmo Event Scheduling/Time Advance.

Tabla 5.8. Marcado inicial del modelo (x=n° de nodo)

Place	Función	Origen	Cantidad	Chronos
L_{x3}	NUL	0	-	0.0
L_{x5}	NUL	0	-	0.0
L_{x6}	-	-	0	0.0
L_{x7}	-	-	0	0.0

En el simulador se han insertado sentencias de código para capturar la siguiente información durante la simulación:

- En cada estado de la simulación se obtienen las transiciones sensibilizadas con su tiempo de disparo, las transiciones disparadas y el marcado de la Red después de realizados los disparos.

- Se almacena el valor proporcionado por la variable aleatoria junto con la transición que lo solicitó. Esta información es necesaria porque ambas simulaciones (centralizada y paralela) proporcionan valores de la variable aleatoria diferentes debido a la distinta ejecución del modelo. Para realizar comparaciones entre los dos métodos de simulación, los valores que toma la variable aleatoria en la simulación centralizada serán proporcionados por el usuario y coincidirán con los obtenidos en la simulación paralela.

El estado final obtenido con ambos métodos de simulación han sido los mismos. Sólo cabe destacar que las comparaciones han sido realizadas a nivel de comportamiento del modelo y no a nivel de estados, pues el método de simulación paralela implementado presenta como inconveniente la posibilidad de alcanzar estados inconsistentes aunque sus resultados finales sean correctos. Esta situación será comentada en el

capítulo 6.

5.3.3. Tiempo de ejecución de la simulación paralela

En [WON 95] [SOL 95] y [SIN 94] se muestran técnicas para analizar el paralelismo de un programa y estimar su rendimiento sobre una arquitectura multiprocesadora. La ejecución del programa se realiza simulando la existencia de N procesadores sobre una máquina monoprocesadora mediante la inserción de instrucciones en el programa que contabilizan el tiempo de ejecución paralela.

Se supone que el programa de simulación está formado por tareas independientes que pueden ser mapeadas directamente a los procesadores en una simulación paralela.

Cada procesador es implementado mediante una cola de entrada y otra de salida. Al mismo tiempo, cada cola mantiene una variable t_i que contabiliza el tiempo de actividad del procesador *i*. Las tareas son insertadas y extraídas de las colas de acuerdo a las reglas de espera del paradigma de Chandy-Misra.

Cuando la simulación ha finalizado, el tiempo de ejecución de la simulación paralela se corresponde con $T_p = \max(t_i)$ con $1 \leq i \leq N$, donde N es el número de procesadores simulados.

Para estimar el tiempo de ejecución de la simulación paralela de nuestro modelo se ha utilizado el mismo método pero adaptado a las particularidades de nuestro sistema y algoritmo. En los dos siguientes puntos se describirán estas particularidades.

5.3.3.1. Sistema multiprocesador

Se supone que el sistema multiprocesador tiene las siguientes características:

- N procesadores homogéneos: procesadores iguales e independientes
- Memoria central y compartida: la memoria no se encuentra distribuida y su espacio de direcciones puede ser referenciado por cualquier procesador.

- Tiempo de acceso a memoria uniforme: el tiempo de acceso a memoria es igual para todos los procesadores.

Se distinguen dos tipos de procesadores: maestro y esclavo. El procesador maestro controla el sistema y determina las tareas que deben ser realizadas por el resto de procesadores esclavos [SIM 84][CHA 88].

El procesador maestro crea un Bloque de Control de Tarea (BCT) por cada tarea que debe ser realizada por los procesadores esclavos. Los BCT creados se almacenan en una pila FIFO en la memoria compartida y contienen, al menos, la siguiente información:

- Identificativo de la tarea
- Estado de la tarea
- Puntero al código de programa de la tarea
- Registros del sistema

Cada vez que se deposita un BCT en la pila de tareas, el procesador maestro manda una señal de control *start* a los procesadores esclavos indicándoles la existencia de tareas pendientes de realización. Una vez recibida la señal *start* por un procesador esclavo, éste solicita el primer BCT de la lista de tareas.

La inexistencia de BCT en la pila de tareas es comunicada al procesador maestro mediante una señal de control *end* por cada uno de los procesadores esclavo que acceden a ella en busca de una tarea.

Finalizada la creación de los BCT, el procesador maestro puede participar junto con los procesadores esclavos en la ejecución de las tareas pendientes de realización.

Con el objeto de prevenir que dos procesadores accedan simultáneamente a la pila de tareas y ejecuten la misma tarea, el sistema operativo implementa un sistema de semáforos.

Los semáforos también serán empleados para garantizar el cumplimiento de las condiciones de concurrencia indicadas en 5.10 en las tareas con secciones críticas (com-

partición de variables por varios procesadores).

$$\begin{aligned} R(a) \cap W(b) &= \varnothing \\ W(a) \cap R(b) &= \varnothing \\ W(a) \cap W(b) &= \varnothing \end{aligned} \qquad (5.10)$$

donde R(a) es el conjunto de variables referenciadas por la tarea *a*, y W(a) es el conjunto de variables cuyos valores son modificados por la tarea *a*.

5.3.3.2. Relaciones de precedencia y concurrencia de la simulación

La ejecución del simulador consiste de ciclos de computación donde se realizan las tareas indicadas en la figura 5.1. Dentro de cada ciclo de computación existen tareas que son ejecutadas secuencialmente y otras en paralelo. Para la correcta realización de cada ciclo de computación se deben respetar las relaciones de precedencia existentes.

En la figura 5.9 se muestra el grafo de precedencia del simulador, que viene a ser una compactación del mostrado en la figura 5.1. En él se muestra la existencia de tres nodos que corresponden a tres actividades que deben ser realizadas en cada ciclo de computación según el orden establecido en el grafo.

La concurrencia existente durante la ejecución del simulador debe respetar las siguientes condiciones:

1. Las condiciones de concurrencia expresadas en 5.10.
2. Las relaciones de precedencia de la figura 5.9.

La segunda condición impone la necesidad de que el procesador maestro espere a la finalización de las tareas pertenecientes a una actividad antes de iniciar la siguiente actividad.

La finalización de las tareas es indicada por los procesadores esclavos mediante la señal de control *end* dirigida al procesador maestro. Esta señal de control se origina cuando un procesador accede a la pila de tareas y ésta se encuentra vacía.

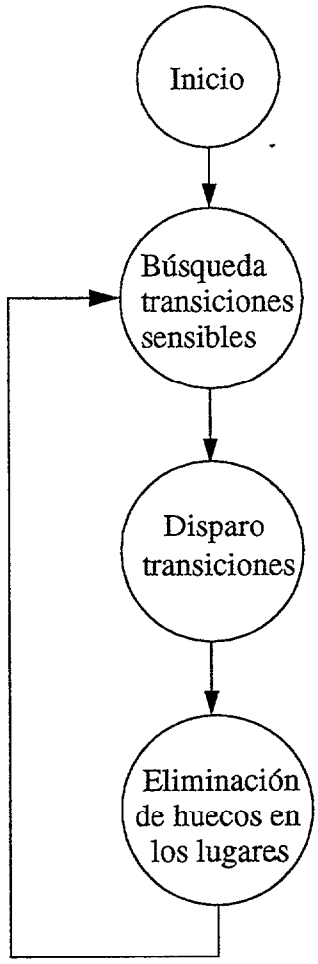


Figura 5.9. Grafo de precedencia del simulador paralelo

El procesador maestro da por finalizada la ejecución de una actividad cuando recibe tantas señales de control *end* como procesadores esclavos existen.

El cumplimiento de la primera condición está garantizado mediante la implementación de semáforos, que evitarán que dos procesadores se encuentren simultáneamente en la misma sección crítica.

En la figura 5.10 se muestra la forma de trabajo del sistema multiprocesador, recogiendo todos los aspectos previamente comentados. Se muestran dos formas de trabajo marcadas como A y B. En la zona A la tareas que deben ser ejecutadas por cada

procesador están predefinidas de antemano, por lo que el procesador maestro actúa como un procesador esclavo.

En la zona B los procesadores esclavos esperan a que el procesador maestro determine las tareas que deben ser ejecutadas. Según se van definiendo, comienza su ejecución por los procesadores esclavos. Una vez definidas todas las tareas a ejecutar, el procesador maestro puede participar en la ejecución de las tareas actuando como un procesador esclavo más.

5.3.3.3. Contabilización del tiempo de ejecución de la simulación

En el programa de simulación se insertan unas instrucciones que contabilicen el tiempo de ejecución de la simulación paralela correspondiente a N procesadores [LIN 93][WON 95].

El método implementa N relojes $\{t_1, \ldots, t_N\}$ que contabilizan los tiempos de ejecución correspondientes a cada uno de los N procesadores simulados. El tiempo de ejecución de las tareas que realiza cada procesador es añadido a su reloj correspondiente incrementando su valor.

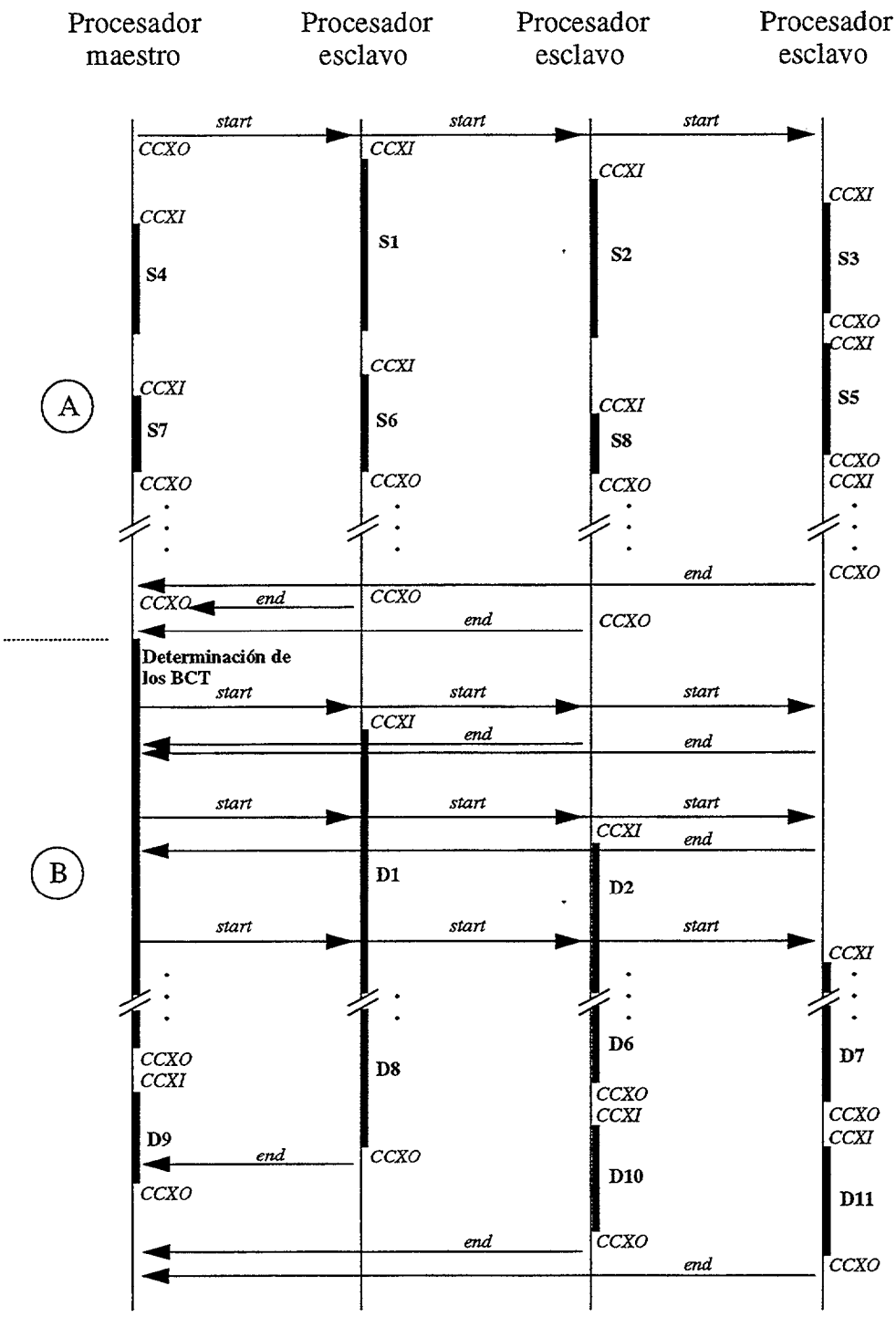
Las relaciones de precedencia obligan a que previamente a realizar una actividad los relojes sean incrementados al valor max (t_i) $i=\{1, \ldots, N\}$, reflejando de esta manera la espera que deben realizar algunos procesadores hasta que se hayan ejecutado todas las tareas pertenecientes a una misma actividad.

Como paso previo se necesita medir los tiempos de ejecución de las tareas que se pretende contabilizar, que son:

Función Sensible
Función Dispara
Función Quitarhuecos

Para medir el tiempo de ejecución de las funciones se empleó el programa "Turbo Profiler" de Borland [TPROF].

Para obtener valores significativos, la medición se ha realizado sobre varios ciclos de ejecución de la misma función, obteniendo de esta forma el valor medio del



CCXO: cambio de contexto de salida (salva registros)
CCXI: cambio de contexto de entrada (carga registros)
start: señal de control start
end: señal de control end

Figura 5.10. Modo de operación de los procesadores

tiempo de ejecución.

Los tiempos de ejecución han sido medidos sobre un ordenador personal 486-66 Mhz y se listan en la tabla 5.11. En el nombre de la función se indica el tipo de tarea medida (Sensible, Disparo, ...) y la transición a la que corresponde, por ejemplo Sensible0 es la tarea que evalúa el predicado de las transiciones t_{x0} con $1 \le x \le 8$. Las tareas correspondientes a las transiciones t_0 y t_{90} son Sensible00 y Sensible000 respectivamente, debido a que su predicado es diferente del resto de transiciones.

Tabla 5.11. Tiempo de ejecución medio de cada función del programa

Función	**Tpo. medio ejecución (µs)**	**Función**	**Tpo. medio ejecución (µs)**
Sensible0	6.0	Dispara0	16.3
Sensible00 (transición t_0)	6.9	Dispara00 (transición t_0)	9.8
Sensible000 (transición t_{90})	4.9	Dispara000 (transición t_{90})	9.8
Sensible1	6.5	Dispara1	10.2
Sensible11 (transición t_{91})	5.2	Dispara11 (transición t_{91})	4.7
Sensible2	6.8	Dispara2	10.8
Sensible22 (transición t_2)	7.9	Dispara22 (transición t_2)	4.7
Sensible3	5.5	Dispara3	3.4
Sensible4	1.6	Dispara4	6.3
Sensible5	2.1	Dispara5	4.7
Sensible6	5.7	Dispara6	5.7
Sensible7	6.9	Dispara7	5.8
Sensible8	2.1	Dispara8	3.9
Sensible9	1.6	Dispara9	13.0
Quitarhuecos	14.6		

Para contabilizar el tiempo de ejecución del procesador maestro necesitamos conocer el tiempo de ejecución medio de una instrucción del programa. El valor obte-

nido mediante el programa Turbo Profiler es 0,53 μs.

Se supondrá que el cambio de contexto (obtención del BCT y finalización de la tarea) tiene una duración equivalente a una instrucción, es decir 0,53 μs.

La LFD se reconstruye en cada nuevo estado de la Red, por lo que las tareas que componen la actividad correspondiente a la búsqueda de las transiciones sensibilizadas del modelo son siempre las mismas, y pueden ser predefinidas de antemano. Para ello, la pila de tareas se iniciará con los BCT correspondientes a estas tareas pero ordenados según su tiempo de ejecución, de manera que la distribución de tareas entre los procesadores se realice mediante el algoritmo Heavy Task First.

En el resto de los casos, en la pila de tareas se almacenarán los BCT según vayan siendo generados, y las tareas serán ejecutadas según este orden.

Para contabilizar el tiempo de ejecución de la simulación paralela, el modelo ha sido simulado hasta haber disparado 50.000 transiciones. Los valores obtenidos se indican en la tabla 5.12 y se representan gráficamente en la figura 5.13 junto con la ganancia de procesamiento G y la eficacia Ef que responden a las siguientes expresiones [SUN 95]:

$$G_k = \frac{T_{p=1}}{T_{p=k}} \tag{5.11}$$

$$Ef_k = \frac{G_k}{k} \tag{5.12}$$

donde k es el número de procesadores.

Tabla 5.12. Resultados de la ejecución paralela para diferente número de procesadores

Número de procesadores	T_p **(seg.)**	**Ganancia**	**Eficacia**
1	9.881	1.0	1.0
2	5.301	1.86	0.93
3	3.935	2.51	0.837

Tabla 5.12. Resultados de la ejecución paralela para diferente número de procesadores

4	3.262	3.03	0.757
5	2.887	3.42	0.684
6	2.637	3.75	0.625
7	2.482	3.98	0.569
8	2.370	4.17	0.521
9	2.291	4.31	0.479
10	2.221	4.45	0.445

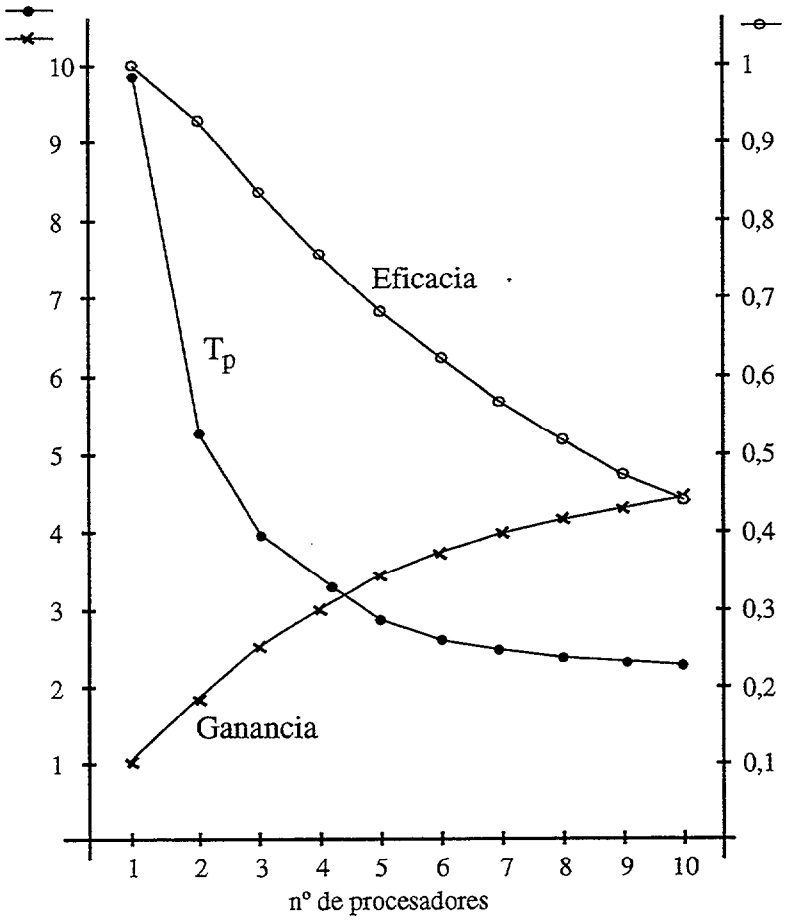


Figura 5.13. Resultados de la ejecución paralela para diferente número de procesadores

Durante la simulación también se obtuvieron las siguientes estadísticas:

Fdp del número de transiciones disparadas en cada estado

El número medio de transiciones disparadas en cada estado de la Red ha sido de 3.3389, siendo la función densidad de probabilidad la indicada en la tabla 5.14 representada gráficamente en la figura 5.15.

La función densidad de probabilidad para valores superiores a 10 es nula, debido fundamentalmente a que sólo se simulan 10 estaciones ethernet y a la existencia de posibles relaciones de causalidad que han impedido que éste número sea mayor.

Se observa como la ganancia llega a ser mayor que el número medio de transiciones disparadas en cada estado. esto podría parecer contradictorio, pero debe recordarse que el simulador comprende otras actividades además del disparo de transiciones, como la búsqueda de transiciones sensibles, la evaluación de relaciones de causalidad y otros. La ganancia mostrada en la tabla 5.12 es el resultado de las ganancias de cada una de estas actividades.

Tabla 5.14. Fdp del número de transiciones disparadas en cada estado

x	Fdp(x)
0	0
1	0.113456
2	0.182237
3	0.265643
4	0.230384
5	0.130618
6	0.0578297
7	0.0168948
8	$2.33723*10^{-3}$
9	$5.34224*10^{-4}$
10	$6.6778*10^{-5}$
11	0

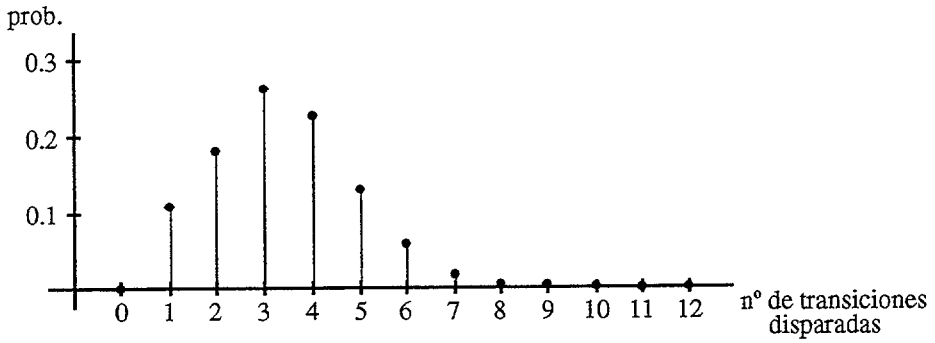


Figura 5.15. Función densidad de probabilidad

Fdp del número de ejecuciones de Quitarhuecos en cada estado

El número medio de ejecuciones de la función Quitarhuecos en cada estado de la Red fue de 1.01863, siendo la función densidad de probabilidad la indicada en la tabla 5.16 representada en la figura 5.17.

El bajo número de ejecuciones de la función Quitarhuecos es debido a que la mayoría de los lugares del modelo son seguros.

Tabla 5.16. Fdp del número de ejecuciones de Quitarhuecos en cada estado

x	Fdp(x)
0	0.319599
1	0.407012
2	0.217763
3	0.0471452
4	0.0078798
5	$5.34224*10^{-4}$
6	0
7	$6.6778*10^{-5}$
8	0

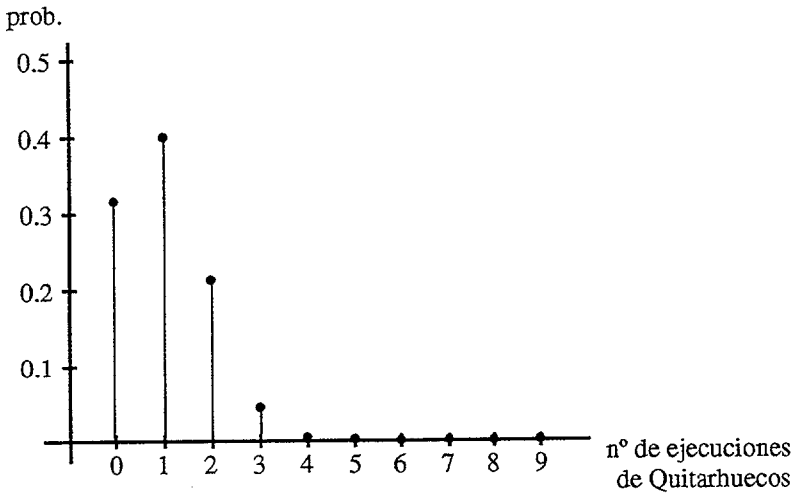


Figura 5.17. Función densidad de probabilidad

Número de ciclos de computación realizados durante la simulación

Se interpreta como ciclo de computación el conjunto de tareas mostrado en la figura 5.1, y que son realizadas ciclicamente hasta la finalización de la simulación.

El número de ciclos de computación realizados en la simulación paralela del modelo durante los 50000 disparos de transiciones fue 14975.

En el caso de una simulación centralizada donde no hubieran aparecido transiciones sensibles con el mismo tiempo de disparo, el número de ciclos de computación hubiera sido 50000.

La disminución de la cantidad de ciclos de computación realizados no implica necesariamente una disminución del tiempo de ejecución de la simulación. La disminución de esta cantidad es consecuencia de agrupar el disparo de varias transiciones en un mismo ciclo de computación. Esto redundará en una disminución del tiempo de ejecución de la simulación siempre que el tiempo de ejecución de las tareas que seleccionan las transiciones que son disparadas en paralelo sea menor que el tiempo empleado para obtener una nueva LFD.

5.4. Comportamiento del simulador

En este punto se trata se estudiar las variaciones que sufre la ganancia de procesamiento bajo diferentes condiciones del modelo que se pretende simular.

El modelo utilizado es mostrado en la figura 5.18. Consiste en un conjunto de fuentes independientes generadoras de token, lo que posibilita el control sobre el número de transiciones del modelo así como su tiempo de disparo.

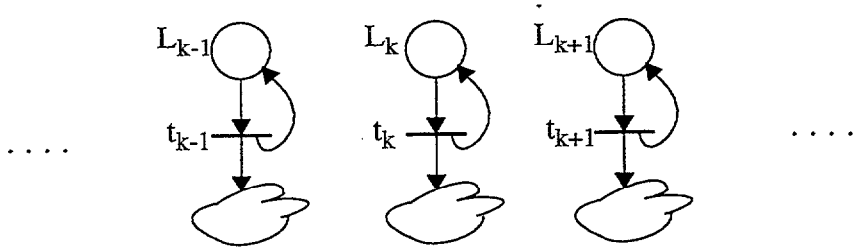


Figura 5.18. Modelo utilizado

Los token sólo contendrán el identificador chronos. En el marcado inicial del modelo cada lugar contendrá dos token con un valor de chronos igual a 0 y 1 respectivamente.

La distancia temporal entre cualesquiera dos transiciones es infinito, con lo que todas las transiciones sensibilizadas en cada estado de la Red serán disparadas. Y evidentemente el número de transiciones sensibles coincidirá con el número de fuentes presentes en el modelo.

5.4.1. Transiciones de igual duración

En un primer experimento se va a simular el modelo variando el número de fuentes de 1 a 50, obteniendo el tiempo de ejecución de la simulación para un número de procesadores comprendido entre 1 y 10. La duración del disparo de las transiciones es igual y por lo tanto las transiciones sensibilizadas tendrán el mismo tiempo de disparo.

Los resultados se muestran en la tabla 5.19 donde la ganancia de procesamiento

es indicada dentro de paréntesis. En filas se indica el número de fuentes que componen el modelo y en columnas el número de procesadores bajo los que se ejecuta la simulación.

Observando la tabla 5.19 se pueden obtener las siguientes conclusiones:

1- Incrementando el número de procesadores sobre un conjunto de fuentes constante aumenta la ganancia de procesamiento hasta un número de procesadores igual al de fuentes del modelo más uno, a partir del cual permanece constante.

   Un análisis más en profundidad muestra que la observación anterior sólo es válida si las transiciones del modelo son disparables en cualquier estado de la Red. En estas circunstancias el número máximo de procesadores viene determinado en el momento del disparo de las transiciones, necesitando un procesador por cada disparo de transición además del procesador maestro. Más procesadores no proporcionan ningún valor añadido.

   Si no todas las transiciones del modelo son sensibles en cada estado de la Red, la ganancia de procesamiento permanece constante a partir de un número de procesadores igual al de transiciones del modelo. El número máximo de procesadores vendrá determinado en el momento de comprobar la sensibilización de las transiciones, necesitando un procesador por cada transición del modelo. En esta actividad el procesador maestro no ejerce como tal.

2- Se observa un incremento del tiempo de ejecución de la simulación (disminuyendo la ganancia de procesamiento) cuando el número de fuentes es una potencia de 2.

   La causa de tal comportamiento radica en el método de ordenación de las transiciones sensibles según su tiempo de disparo. Este es implementado mediante una serie de ciclos dentro de los cuales se ordena un conjunto de transiciones. Cuando el número de transiciones sensibles (igual al de fuentes) es una potencia de 2, se añade un nuevo ciclo aumentando el tiempo de ejecución del algoritmo de ordenación y por lo tanto de la simulación.

Tabla 5.19. Tiempo de ejecución y ganancia de procesamiento de la simulación

	1	2	3	4	5	6	7	8	9	10
1	2.081 (1)	1.981 (1.05)	1.981 (1.05)	1.981 (1.05)	1.981 (1.05)	1.981 (1.05)	1.981 (1.05)	1.981 (1.05)	1.981 (1.05)	1.981 (1.05)
2	2.006 (1)	1.141 (1.76)	1.091 (1.84)	1.091 (1.84)	1.091 (1.84)	1.091 (1.84)	1.091 (1.84)	1.091 (1.84)	1.091 (1.84)	1.091 (1.84)
3	1.973 (1)	1.27 (1.55)	0.8202 (2.4)	0.7866 (2.51)	0.7866 (2.51)	0.7866 (2.51)	0.7866 (2.51)	0.7866 (2.51)	0.7866 (2.51)	0.7866 (2.51)
4	1.974 (1)	1.103 (1.79)	0.9777 (2.02)	0.6788 (2.91)	0.6535 (3.02)	0.6535 (3.02)	0.6535 (3.02)	0.6535 (3.02)	0.6535 (3.02)	0.6535 (3.02)
5	1.966 (1)	1.143 (1.72)	0.8227 (2.39)	0.7925 (2.48)	0.5839 (3.37)	0.5637 (3.49)	0.5637 (3.49)	0.5637 (3.49)	0.5637 (3.49)	0.5637 (3.49)
6	1.956 (1)	1.083 (1.81)	0.795 (2.46)	0.6901 (2.83)	0.6649 (2.94)	0.5164 (3.79)	0.4995 (3.92)	0.4995 (3.92)	0.4995 (3.92)	0.4995 (3.92)
7	1.952 (1)	1.078 (1.81)	0.8276 (2.36)	0.6205 (3.15)	0.5989 (3.26)	0.5775 (3.38)	0.4719 (4.14)	0.4574 (4.27)	0.4574 (4.27)	0.4574 (4.27)
8	1.968 (1)	1.091 (1.8)	0.7683 (2.56)	0.6599 (2.98)	0.5683 (3.46)	0.5496 (3.58)	0.5339 (3.69)	0.4541 (4.33)	0.4446 (4.43)	0.4446 (4.43)
9	1.967 (1)	1.072 (1.83)	0.7952 (2.47)	0.6802 (2.89)	0.5307 (3.71)	0.514 (3.83)	0.5 (3.93)	0.4857 (4.05)	0.4238 (4.64)	0.4208 (4.67)
10	1.963 (1)	1.076 (1.82)	0.7924 (2.48)	0.6324 (3.1)	0.5586 (3.51)	0.483 (4.06)	0.4703 (4.17)	0.4575 (4.29)	0.4425 (4.44)	0.4017 (4.89)
11	1.963 (1)	1.057 (1.86)	0.7478 (2.62)	0.5979 (3.28)	0.5683 (3.45)	0.4599 (4.27)	0.4485 (4.38)	0.4367 (4.49)	0.4232 (4.64)	0.4137 (4.74)
12	1.958 (1)	1.064 (1.84)	0.7741 (2.53)	0.6304 (3.11)	0.5357 (3.65)	0.489 (4)	0.426 (4.6)	0.4152 (4.71)	0.4028 (4.86)	0.3942 (4.97)
13	1.958 (1)	1.046 (1.87)	0.7593 (2.58)	0.6247 (3.13)	0.5122 (3.82)	0.4902 (3.99)	0.4109 (4.76)	0.401 (4.88)	0.3895 (5.03)	0.3816 (5.13)
14	1.956 (1)	1.052 (1.86)	0.7356 (2.66)	0.5946 (3.29)	0.4991 (3.92)	0.4697 (4.16)	0.4359 (4.49)	0.3869 (5.05)	0.3763 (5.2)	0.3689 (5.3)
15	1.956 (1)	1.047 (1.87)	0.7556 (2.59)	0.5852 (3.34)	0.5256 (3.72)	0.4536 (4.31)	0.4373 (4.47)	0.3782 (5.17)	0.3665 (5.34)	0.3596 (5.44)
16	1.976 (1)	1.062 (1.86)	0.7635 (2.59)	0.6273 (3.15)	0.5448 (3.63)	0.46 (4.29)	0.4447 (4.44)	0.415 (4.76)	0.3784 (5.22)	0.372 (5.31)
17	1.977 (1)	1.067 (1.85)	0.7598 (2.6)	0.6233 (3.17)	0.5279 (3.74)	0.457 (4.32)	0.4336 (4.56)	0.4176 (4.73)	0.3714 (5.32)	0.3652 (5.41)
18	1.975 (1)	1.056 (1.87)	0.7619 (2.59)	0.6013 (3.28)	0.5111 (3.86)	0.4744 (4.16)	0.4222 (4.68)	0.407 (4.85)	0.3818 (5.17)	0.3577 (5.52)
19	1.975 (1)	1.064 (1.86)	0.7562 (2.61)	0.6065 (3.26)	0.5158 (3.83)	0.4761 (4.15)	0.4134 (4.78)	0.399 (4.95)	0.3844 (5.14)	0.3523 (5.61)
20	1.973 (1)	1.051 (1.88)	0.7561 (2.61)	0.6071 (3.25)	0.5209 (3.79)	0.4625 (4.27)	0.4116 (4.79)	0.3892 (5.07)	0.3753 (5.26)	0.3575 (5.52)
21	1.973 (1)	1.061 (1.86)	0.7486 (2.64)	0.6047 (3.26)	0.5214 (3.78)	0.4526 (4.36)	0.4201 (4.7)	0.3829 (5.15)	0.3696 (5.34)	0.3598 (5.48)
22	1.973 (1)	1.048 (1.88)	0.75 (2.63)	0.5956 (3.31)	0.5082 (3.88)	0.4459 (4.42)	0.4217 (4.68)	0.3759 (5.25)	0.3633 (5.43)	0.354(5.57)
23	1.973 (1)	1.059 (1.86)	0.7559 (2.61)	0.6049 (3.26)	0.5047 (3.91)	0.4553 (4.33)	0.4144 (4.76)	0.3817 (5.17)	0.3586 (5.5)	0.3496 (5.64)
24	1.971 (1)	1.042 (1.89)	0.7369 (2.67)	0.5921 (3.33)	0.5134 (3.84)	0.4484 (4.4)	0.4047 (4.87)	0.3777 (5.22)	0.3512 (5.61)	0.3426 (5.75)
25	1.97 (1)	1.058 (1.86)	0.7425 (2.65)	0.5905 (3.34)	0.5046 (3.9)	0.4506 (4.37)	0.3986 (4.94)	0.3807 (5.18)	0.3473 (5.67)	0.3391 (5.81)
26	1.971 (1)	1.038 (1.9)	0.7534 (2.62)	0.5864 (3.36)	0.5049 (3.9)	0.4423 (4.46)	0.3962 (4.97)	0.3751 (5.25)	0.3556 (5.54)	0.335 (5.88)
27	1.97 (1)	1.057 (1.86)	0.7371 (2.67)	0.6003 (3.28)	0.4955 (3.98)	0.4351 (4.53)	0.4087 (4.82)	0.3705 (5.32)	0.3518 (5.6)	0.332 (5.94)
28	1.969 (1)	1.037 (1.9)	0.7377 (2.67)	0.5862 (3.36)	0.4912 (4.01)	0.4321 (4.56)	0.4014 (4.91)	0.3646 (5.4)	0.3528 (5.58)	0.3275 (6.01)
29	1.97 (1)	1.053 (1.87)	0.7447 (2.65)	0.5895 (3.34)	0.5049 (3.9)	0.4454 (4.42)	0.4043 (4.87)	0.3609 (5.46)	0.3496 (5.64)	0.3338 (5.9)
30	1.969 (1)	1.037 (1.9)	0.7346 (2.68)	0.5838 (3.37)	0.4956 (3.97)	0.4381 (4.49)	0.3984 (4.94)	0.3578 (5.5)	0.3455 (5.7)	0.3302 (5.96)
31	1.969 (1)	1.051 (1.87)	0.7393 (2.66)	0.59 (3.34)	0.4967 (3.96)	0.4402 (4.47)	0.3937 (5)	0.3676 (5.36)	0.3425 (5.75)	0.3326 (5.92)
32	1.992 (1)	1.061 (1.88)	0.7605 (2.62)	0.6017 (3.31)	0.5177 (3.85)	0.4567 (4.36)	0.4147 (4.8)	0.3863 (5.16)	0.362 (5.5)	0.3524 (5.65)
33	1.993 (1)	1.07 (1.86)	0.7551 (2.64)	0.609 (3.27)	0.5187 (3.84)	0.456 (4.37)	0.4144 (4.81)	0.3893 (5.12)	0.3597 (5.54)	0.3503 (5.69)
34	1.992 (1)	1.064 (1.87)	0.7641 (2.61)	0.6106 (3.26)	0.5194 (3.84)	0.4579 (4.35)	0.418 (4.77)	0.3852 (5.17)	0.358 (5.56)	0.3474(5.73)
35	1.993 (1)	1.065 (1.87)	0.7586 (2.63)	0.6075 (3.28)	0.5126 (3.89)	0.459 (4.34)	0.4141 (4.81)	0.3822 (5.21)	0.3616 (5.51)	0.3455 (5.77)
36	1.991 (1)	1.066 (1.87)	0.749 (2.66)	0.5987 (3.33)	0.5129 (3.88)	0.4525 (4.4)	0.4152 (4.8)	0.3779 (5.27)	0.3579 (5.56)	0.3423 (5.82)
37	1.992 (1)	1.061 (1.88)	0.7648 (2.61)	0.6022 (3.31)	0.5103 (3.9)	0.4548 (4.38)	0.4117 (4.84)	0.3768 (5.29)	0.3607 (5.52)	0.3407 (5.85)
38	1.991 (1)	1.068 (1.86)	0.7572 (2.63)	0.6112 (3.26)	0.5198 (3.83)	0.4514 (4.41)	0.4081 (4.88)	0.3839 (5.19)	0.3578 (5.57)	0.3449 (5.77)
39	1.993 (1)	1.059 (1.88)	0.7465 (2.67)	0.603 (3.3)	0.514 (3.88)	0.4482 (4.45)	0.4059 (4.91)	0.3815 (5.22)	0.356 (5.6)	0.3434 (5.8)
40	1.99 (1)	1.066 (1.87)	0.7599 (2.62)	0.5989 (3.32)	0.5058 (3.93)	0.4557 (4.37)	0.4127 (4.82)	0.3768 (5.28)	0.352 (5.65)	0.3397 (5.86)
41	1.991 (1)	1.059 (1.88)	0.7558 (2.63)	0.5992 (3.32)	0.5122 (3.89)	0.4516 (4.41)	0.4096 (4.86)	0.3795 (5.25)	0.3504 (5.68)	0.3421 (5.82)
42	1.991 (1)	1.063 (1.87)	0.7474 (2.66)	0.606 (3.29)	0.5067 (3.93)	0.4469 (4.45)	0.4059 (4.9)	0.3765 (5.29)	0.3482 (5.72)	0.34 (5.86)
43	1.991 (1)	1.062 (1.88)	0.7585 (2.62)	0.5987 (3.32)	0.5141 (3.87)	0.4488 (4.44)	0.4082 (4.88)	0.3742 (5.32)	0.3542 (5.62)	0.3386 (5.88)
44	1.991 (1)	1.059 (1.88)	0.7511 (2.65)	0.5948 (3.35)	0.5078 (3.92)	0.4472 (4.45)	0.4043 (4.92)	0.3722 (5.35)	0.3515 (5.66)	0.3362 (5.92)
45	1.992 (1)	1.064 (1.87)	0.7492 (2.66)	0.6015 (3.31)	0.5044 (3.95)	0.4473 (4.45)	0.4036 (4.93)	0.3709 (5.37)	0.35 (5.69)	0.3351 (5.94)
46	1.99 (1)	1.055 (1.89)	0.7569 (2.63)	0.6007 (3.31)	0.5064 (3.93)	0.4497 (4.42)	0.4038 (4.93)	0.3748 (5.31)	0.3515 (5.66)	0.3331 (5.97)
47	1.99 (1)	1.066 (1.87)	0.7479 (2.66)	0.5985 (3.33)	0.5099 (3.9)	0.4461 (4.46)	0.4062 (4.9)	0.3729 (5.34)	0.35 (5.69)	0.3328 (5.98)
48	1.989 (1)	1.052 (1.89)	0.7487 (2.66)	0.5901 (3.37)	0.5066 (3.93)	0.4406 (4.51)	0.4016 (4.95)	0.3689 (5.39)	0.3465 (5.74)	0.3327 (5.98)
49	1.99 (1)	1.066 (1.87)	0.7544 (2.64)	0.6026 (3.3)	0.5021 (3.96)	0.4457 (4.46)	0.3992 (4.98)	0.3713 (5.36)	0.3453 (5.76)	0.3317 (6)
50	1.989 (1)	1.051 (1.89)	0.7439 (2.67)	0.5961 (3.34)	0.501 (3.97)	0.4416 (4.5)	0.4006 (4.96)	0.3687 (5.39)	0.3433 (5.79)	0.33 (6.03)

3- Cuando se cumple la condición 5.14

$$(\text{n° de fuentes } \textit{mod} \text{ n° de procesadores}) = 1 \qquad (5.14)$$

se observa un incremento del tiempo de ejecución de la simulación con respecto al correspondiente a un número de fuentes inmediatamente inferior, debido a un desequilibrio en la carga asignada a los procesadores cuando se distribuyen las tareas para comprobar la sensibilización de las transiciones. Este hecho se refleja gráficamente en la figura 5.20 para 7 fuentes y 3 procesadores.

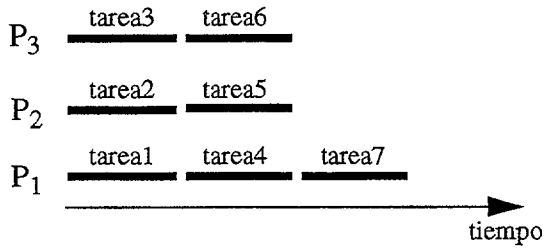


Figura 5.20. Distribución de tareas de sensibilización

Este desequilibrio es dependiente del tiempo de ejecución de las tareas que se distribuyen siendo más acentuado cuando éstas son similares.

4- El comportamiento descrito en la conclusión 3 puede observarse también en la distribución de las tareas de disparo y de la función Quitarhuecos pero con la diferencia de que existe un procesador maestro que no se puede predecir si participará en la distribución y cuando, no pudiendo establecer una condición que determine cuando ocurriría el desequilibrio en la carga.

En la tabla 5.19 se observa que con cantidades pequeñas de fuentes múltiplo del número de procesadores se incrementa el tiempo de ejecución de la simulación. Esta conducta desaparece con un número de fuentes mayor. Dos situaciones distintas son mostradas en la figura 5.21 para 3 procesadores.

5.4.2. Transiciones de distinta duración

En un siguiente experimento se fijaron duraciones distintas para las transiciones

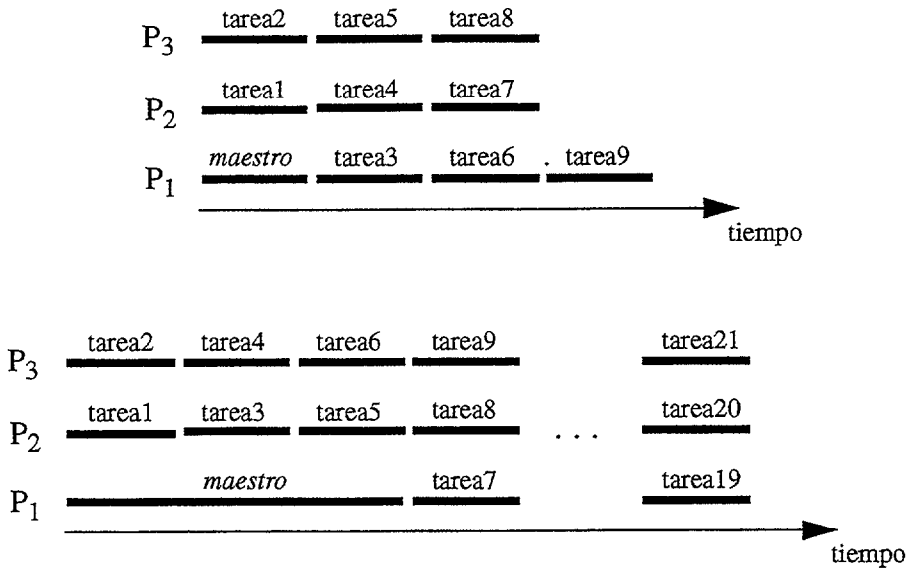


Figura 5.21. Distribución de tareas de disparo

dando lugar a transiciones sensibles con distinto instante de disparo. El resto de características del modelo permanece igual que en el punto anterior.

La consecuencia inmediata de la existencia de distintos instantes de disparo es la necesidad de ordenar las transiciones según su instante de disparo. De forma que en este experimento se comprobará la incidencia de la rutina de ordenación llevada a cabo por el procesador maestro sobre el tiempo de ejecución de la simulación. Los resultados son mostrados en la tabla 5.22.

Es de destacar la fuerte disminución de la ganancia de procesamiento con respecto a la obtenida en la tabla 5.19, debido a la realización de actividades que exigen ser realizadas de manera secuencial por el procesador maestro y que penalizan el tiempo de procesamiento. Estas son: la ordenación de las transiciones por su tiempo de disparo y la evaluación de la condición 2.17 para decidir si dos transiciones son mutuamente independientes.

Por otra parte las fluctuaciones del tiempo de ejecución de la rutina de ordenación (complejidad $0(n^2)$) de las transiciones inciden directamente en fluctuaciones en la ganancia de procesamiento.

Tabla 5.22. Tiempo de ejecución y ganancia de procesamiento de la simulación

	1	2	3	4	5	6	7	8	9	10
1	1.988 (1)	1.879 (1.06)	1.879 (1.06)	1.879 (1.06)	1.879 (1.06)	1.879 (1.06)	1.879 (1.06)	1.879 (1.06)	1.879 (1.06)	1.879 (1.06)
2	2.001 (1)	1.206 (1.66)	1.193 (1.68)	1.193 (1.68)	1.193 (1.68)	1.193 (1.68)	1.193 (1.68)	1.193 (1.68)	1.193 (1.68)	1.193 (1.68)
3	1.967 (1)	1.275 (1.54)	0.9097 (2.16)	0.9004 (2.18)	0.9004 (2.18)	0.9004 (2.18)	0.9004 (2.18)	0.9004 (2.18)	0.9004 (2.18)	0.9004 (2.18)
4	2.012 (1)	1.158 (1.74)	1.071 (1.88)	0.8228 (2.45)	0.8158 (2.47)	0.8158 (2.47)	0.8158 (2.47)	0.8158 (2.47)	0.8158 (2.47)	0.8158 (2.47)
5	2.006 (1)	1.202 (1.67)	0.9355 (2.14)	0.9131 (2.2)	0.7369 (2.72)	0.7313 (2.74)	0.7313 (2.74)	0.7313 (2.74)	0.7313 (2.74)	0.7313 (2.74)
6	1.966 (1)	1.114 (1.77)	0.8547 (2.3)	0.7891 (2.49)	0.7703 (2.55)	0.6423 (3.06)	0.6376 (3.08)	0.6376 (3.08)	0.6376 (3.08)	0.6376 (3.08)
7	1.975 (1)	1.127 (1.75)	0.9025 (2.19)	0.7406 (2.67)	0.7245 (2.73)	0.7084 (2.79)	0.6147 (3.21)	0.6107 (3.23)	0.6107 (3.23)	0.6107 (3.23)
8	2.1 (1)	1.231 (1.71)	0.9653 (2.18)	0.8692 (2.42)	0.8095 (2.59)	0.7954 (2.64)	0.7813 (2.69)	0.7133 (2.94)	0.7098 (2.96)	0.7098 (2.96)
9	2.102 (1)	1.22 (1.72)	0.9767 (2.15)	0.8924 (2.35)	0.7789 (2.7)	0.7664 (2.74)	0.7539 (2.79)	0.7414 (2.83)	0.6935 (3.03)	0.6903 (3.04)
10	2.009 (1)	1.128 (1.78)	0.8855 (2.27)	0.761 (2.64)	0.7039 (2.85)	0.6476 (3.1)	0.6364 (3.16)	0.6252 (3.21)	0.6139 (3.27)	0.582 (3.45)
11	2.023 (1)	1.115 (1.81)	0.8588 (2.36)	0.7455 (2.71)	0.7251 (2.79)	0.6425 (3.15)	0.6323 (3.2)	0.6221 (3.25)	0.6118 (3.31)	0.6016 (3.36)
12	2.054 (1)	1.163 (1.77)	0.92 (2.23)	0.8091 (2.54)	0.7326 (2.8)	0.7006 (2.93)	0.6474 (3.17)	0.6381 (3.22)	0.6287 (3.27)	0.6193 (3.32)
13	2.067 (1)	1.14 (1.81)	0.9242 (2.24)	0.8197 (2.52)	0.726 (2.85)	0.7152 (2.89)	0.6475 (3.19)	0.6388 (3.24)	0.6302 (3.28)	0.6215 (3.33)
14	2.039 (1)	1.122 (1.82)	0.8649 (2.36)	0.7671 (2.66)	0.6801 (3)	0.6701 (3.04)	0.6387 (3.19)	0.5992 (3.4)	0.5912 (3.45)	0.5831 (3.5)
15	2.054 (1)	1.12 (1.83)	0.8998 (2.28)	0.7635 (2.69)	0.7223 (2.84)	0.6723 (3.06)	0.6574 (3.12)	0.6061 (3.39)	0.5986 (3.43)	0.5911 (3.47)
16	2.298 (1)	1.362 (1.69)	1.138 (2.02)	1.031 (2.23)	0.9714 (2.37)	0.9075 (2.53)	0.8936 (2.57)	0.8678 (2.65)	0.8385 (2.74)	0.8314 (2.76)
17	2.302 (1)	1.363 (1.69)	1.132 (2.03)	1.034 (2.23)	0.9608 (2.4)	0.9012 (2.55)	0.8875 (2.59)	0.8743 (2.63)	0.8357 (2.75)	0.8291 (2.78)
18	2.097 (1)	1.153 (1.82)	0.9313 (2.25)	0.8111 (2.59)	0.7419 (2.83)	0.7096 (2.96)	0.6726 (3.12)	0.6602 (3.18)	0.6388 (3.28)	0.6175 (3.4)
19	2.113 (1)	1.18 (1.79)	0.944 (2.24)	0.8311 (2.54)	0.7611 (2.78)	0.7295 (2.9)	0.6817 (3.1)	0.6698 (3.15)	0.658 (3.21)	0.6294 (3.36)
20	2.145 (1)	1.193 (1.8)	0.9814 (2.19)	0.8615 (2.49)	0.7951 (2.7)	0.7519 (2.85)	0.7153 (3)	0.6952 (3.09)	0.684 (3.14)	0.6662 (3.22)
21	2.16 (1)	1.226 (1.76)	0.9861 (2.19)	0.8776 (2.46)	0.813 (2.66)	0.759 (2.85)	0.7327 (2.95)	0.705 (3.06)	0.6942 (3.11)	0.6836 (3.16)
22	2.133 (1)	1.177 (1.81)	0.9562 (2.23)	0.8378 (2.55)	0.7761 (2.75)	0.7246 (2.94)	0.7092 (3.01)	0.673 (3.17)	0.6628 (3.22)	0.6526 (3.27)
23	2.149 (1)	1.217 (1.77)	0.9895 (2.17)	0.8726 (2.46)	0.7839 (2.74)	0.757 (2.84)	0.7199 (2.99)	0.7003 (3.07)	0.6755 (3.18)	0.6657 (3.23)
24	2.241 (1)	1.281 (1.75)	1.064 (2.11)	0.9524 (2.35)	0.8927 (2.51)	0.8415 (2.66)	0.806 (2.78)	0.7872 (2.85)	0.7634 (2.94)	0.754 (2.97)
25	2.253 (1)	1.314 (1.72)	1.074 (2.1)	0.9665 (2.33)	0.8972 (2.51)	0.8578 (2.63)	0.814 (2.77)	0.8034 (2.8)	0.7731 (2.91)	0.7641 (2.95)
26	2.17 (1)	1.213 (1.79)	1.003 (2.16)	0.8714 (2.49)	0.8152 (2.66)	0.7669 (2.83)	0.7247 (2.99)	0.7146 (3.04)	0.6959 (3.12)	0.6767 (3.21)
27	2.186 (1)	1.243 (1.76)	1.005 (2.17)	0.9013 (2.42)	0.8242 (2.65)	0.7777 (2.81)	0.7535 (2.9)	0.7274 (3)	0.7094 (3.08)	0.691 (3.16)
28	2.218 (1)	1.262 (1.76)	1.044 (2.12)	0.9234 (2.4)	0.8517 (2.6)	0.8042 (2.76)	0.7808 (2.84)	0.7557 (2.93)	0.744 (2.98)	0.7206 (3.08)
29	2.234 (1)	1.286 (1.74)	1.062 (2.1)	0.9409 (2.37)	0.877 (2.55)	0.8311 (2.69)	0.8004 (2.79)	0.7687 (2.91)	0.7574 (2.95)	0.7412 (3.01)
30	2.207 (1)	1.254 (1.76)	1.023 (2.16)	0.9165 (2.41)	0.8438 (2.62)	0.7994 (2.76)	0.7697 (2.87)	0.7391 (2.99)	0.7282 (3.03)	0.7126 (3.1)
31	2.223 (1)	1.273 (1.75)	1.054 (2.11)	0.934 (2.38)	0.8633 (2.58)	0.8194 (2.71)	0.7829 (2.84)	0.7624 (2.92)	0.7427 (2.99)	0.7318 (3.04)
32	2.694 (1)	1.734 (1.55)	1.507 (1.79)	1.388 (1.94)	1.321 (2.04)	1.28 (2.1)	1.246 (2.16)	1.227 (2.2)	1.209 (2.23)	1.199 (2.25)
33	2.698 (1)	1.736 (1.55)	1.499 (1.8)	1.392 (1.94)	1.332 (2.03)	1.28 (2.11)	1.255 (2.15)	1.233 (2.19)	1.21 (2.23)	1.2 (2.25)
34	2.27 (1)	1.32 (1.72)	1.103 (2.06)	0.9877 (2.3)	0.9103 (2.49)	0.8694 (2.61)	0.832 (2.73)	0.8083 (2.81)	0.792 (2.87)	0.7749 (2.93)
35	2.287 (1)	1.331 (1.72)	1.109 (2.06)	0.9972 (2.29)	0.922 (2.48)	0.8824 (2.59)	0.846 (2.7)	0.823 (2.78)	0.8071 (2.83)	0.7905 (2.89)
36	2.318 (1)	1.366 (1.7)	1.129 (2.05)	1.022 (2.27)	0.956 (2.42)	0.91 (2.55)	0.8806 (2.63)	0.8522 (2.72)	0.8368 (2.77)	0.8207 (2.82)
37	2.334 (1)	1.378 (1.69)	1.163 (2.01)	1.04 (2.25)	0.9678 (2.41)	0.9296 (2.51)	0.8944 (2.61)	0.8667 (2.69)	0.8561 (2.73)	0.8361 (2.79)
38	2.308 (1)	1.355 (1.7)	1.128 (2.05)	1.024 (2.25)	0.9529 (2.42)	0.8996 (2.57)	0.8654 (2.67)	0.8479 (2.72)	0.8281 (2.79)	0.8136 (2.84)
39	2.326 (1)	1.371 (1.7)	1.141 (2.04)	1.035 (2.25)	0.9658 (2.41)	0.9139 (2.55)	0.8805 (2.64)	0.8634 (2.69)	0.8442 (2.76)	0.83 (2.8)
40	2.413 (1)	1.456 (1.66)	1.237 (1.95)	1.114 (2.17)	1.05 (2.3)	1.012 (2.39)	0.9784 (2.47)	0.9508 (2.54)	0.932 (2.59)	0.9182 (2.63)
41	2.429 (1)	1.473 (1.65)	1.242 (1.95)	1.136 (2.14)	1.067 (2.28)	1.024 (2.37)	0.9913 (2.45)	0.9689 (2.51)	0.946 (2.57)	0.9358 (2.6)
42	2.347 (1)	1.388 (1.69)	1.165 (2.01)	1.059 (2.22)	0.9881 (2.37)	0.9372 (2.5)	0.9055 (2.59)	0.8836 (2.66)	0.8613 (2.72)	0.8513 (2.76)
43	2.362 (1)	1.411 (1.67)	1.185 (1.99)	1.07 (2.21)	1.006 (2.35)	0.9568 (2.47)	0.9252 (2.55)	0.8988 (2.63)	0.8821 (2.68)	0.8673 (2.72)
44	2.395 (1)	1.434 (1.67)	1.208 (1.98)	1.094 (2.19)	1.034 (2.32)	0.9859 (2.43)	0.955 (2.51)	0.9292 (2.58)	0.9128 (2.62)	0.8983 (2.67)
45	2.411 (1)	1.459 (1.65)	1.23 (1.96)	1.124 (2.14)	1.047 (2.3)	1.01 (2.39)	0.9697 (2.49)	0.9513 (2.53)	0.9284 (2.6)	0.9142 (2.64)
46	2.384 (1)	1.423 (1.68)	1.205 (1.98)	1.095 (2.18)	1.022 (2.33)	0.9811 (2.43)	0.9507 (2.51)	0.9232 (2.58)	0.9043 (2.64)	0.887 (2.69)
47	2.401 (1)	1.449 (1.66)	1.214 (1.98)	1.106 (2.17)	1.049 (2.29)	0.9956 (2.41)	0.9659 (2.49)	0.939 (2.56)	0.9204 (2.61)	0.9075 (2.65)
48	2.605 (1)	1.64 (1.59)	1.418 (1.84)	1.296 (2.01)	1.239 (2.1)	1.19 (2.19)	1.162 (2.24)	1.137 (2.29)	1.12 (2.32)	1.109 (2.35)
49	2.618 (1)	1.661 (1.58)	1.43 (1.83)	1.322 (1.98)	1.248 (2.1)	1.204 (2.17)	1.173 (2.23)	1.152 (2.27)	1.132 (2.31)	1.12 (2.34)
50	2.422 (1)	1.461 (1.66)	1.237 (1.96)	1.13 (2.14)	1.059 (2.29)	1.014 (2.39)	0.9858 (2.46)	0.9599 (2.52)	0.9387 (2.58)	0.9266 (2.61)

Por lo demás se observa el mismo comportamiento que el descrito en el punto 5.4.1, aunque un poco enmascarado debido a las fluctuaciones indicadas anteriormente.

5.5. Resultados y conclusiones

En la tabla 5.12 se muestra el tiempo de ejecución de la simulación paralela estimado y la ganancia de procesamiento con respecto a la misma simulación con un único procesador.

Se observa una disminución del tiempo de simulación con el aumento del número de procesadores como era de esperar, aunque esta relación no es proporcional.

El motivo por el cual la ganancia de procesamiento dista de la ideal es que aún siendo la simulación paralela, el simulador cuenta con una parte de código que debe ser ejecutado secuencialmente por el procesador maestro (ordenar las transiciones por su tiempo de disparo, iniciación de variables, iniciar los BCT y otros). Esto puede ser considerado como el límite inferior para T_p y tiene un valor de 1.0947 segundos. Aún así este límite inferior es inalcanzable puesto que habría que añadirle los tiempos de espera debidos a las relaciones de precedencia y la participación del procesador maestro en la realización de tareas.

La relación entre la disminución del tiempo de ejecución de la simulación y el coste de los recursos utilizados se indica mediante el parámetro eficacia que también es mostrado en la tabla 5.12. Será labor del usuario del simulador decidir un umbral mínimo para este parámetro.

Como primera aproximación se podría concluir que el número de procesadores ideal para realizar la simulación viene dado por la expresión

$$\text{int}(n^o \textit{ medio de transiciones disparadas}) + 1$$

El motivo del bajo número de transiciones disparadas en paralelo en cada estado radica en el modelo que, aún pareciendo altamente concurrente, está obligado a satisfacer unas relaciones temporales muy restrictivas debido a la existencia del canal que intercomunica las diferentes estaciones.

Además de la mejora obtenida con el aumento del número de procesadores, es

obligado constatar la disminución del tiempo de ejecución de la simulación paralela realizada sobre un sólo procesador con respecto a la simulación centralizada llevada a cabo con el método descrito en el punto 3.1. Este resultado es consecuencia del adelanto de la ejecución de aquellos eventos futuros que no mantienen una relación de causalidad con los eventos que van a ser ejecutados en un instante de ejecución dado. El tiempo de ejecución de la simulación centralizada del modelo durante 50000 disparos de transiciones realizada con este método es de 18.2227 segundos. Comparando este valor con el mostrada en la tabla 5.12 para un procesador se concluye que la simulación paralela realizada sobre un sólo procesador consume un 45.78% menos de tiempo de ejecución.

Capítulo 6: Conclusiones y líneas abiertas

6. Conclusiones y líneas abiertas

6.1. Conclusiones

Se ha desarrollado un algoritmo para la simulación paralela de Redes de Petri temporizadas sobre una arquitectura multiprocesadora genérica.

Las simulaciones realizadas mediante este algoritmo muestran una disminución considerable de su tiempo de ejecución con respecto al correspondiente a la simulación centralizada, dejando al usuario la tarea de situar el punto de trabajo de la simulación (coste de los procesadores versus magnitud de la disminución del tiempo de ejecución).

En el caso de utilizar condiciones de lugar del tipo "máximo del lugar" se necesita evaluar localmente a nivel de transición el cumplimiento del axioma 3 con objeto de evitar la desaparición de los token muertos que pudieran surgir como consecuencia de la ejecución de la Red de Petri.

Finalmente, el análisis del comportamiento del simulador bajo diferentes situaciones ha mostrado la relación entre el tiempo de ejecución de la simulación realizada con el algoritmo obtenido y algunas características del modelo (transiciones existentes en el modelo, transiciones sensibles en cada estado y otros).

6.2. Líneas abiertas

Concluido el estudio han quedado áreas que merecen un análisis más en profundidad, así como futuros campos de investigación.

1 - Relaciones de causalidad

La simulación es llevada a cabo detectando una "posible" relación de causalidad entre transiciones de la Red de Petri. Un mayor número de transiciones disparables en cada estado puede ser obtenido verificando la "posible" causalidad mediante la evaluación de los predicados asociados a las transiciones.

Los experimentos realizados para verificar la posible causalidad muestran que se penaliza enormemente el tiempo de ejecución de la simulación. Sin embargo, habría que contemplar algún tipo de análisis con poco coste computacional con el que se pueda descartar algunas de las posibles relaciones de causalidad.

2 - Algoritmos de distribución de tareas

Se debe abordar la obtención de un algoritmo de distribución de tareas en las que su tiempo de ejecución viene determinado mediante una variable aleatoria.

3 - Entorno de simulación paralela

Un campo de aplicación es el desarrollo de un simulador paralelo de Redes de Petri interactivo con un entorno user-friendly, donde se pueda manejar la escala de tiempo simulado a voluntad del usuario. Exigirá una metodología para el modelado de sistemas y otras especificaciones. Trabajos en este campo pueden encontrarse en [BIL 88] [LAK 92a] [LAK 92b] y [LAK 94].

4- Simulación sobre procesadores débilmente acoplados

Las simulaciones paralelas han sido realizadas sobre procesadores que comparten memoria. Otro tipo de arquitectura son las consistentes en procesadores interconectados mediante una red de comunicaciones.

Las simulaciones sobre estas arquitecturas están fuertemente penalizadas por el inevitable tiempo de propagación de los mensajes a través de la red de comunicaciones, aunque los avances tecnológicos han resultado en la convergencia de arquitecturas con memoria compartida y arquitecturas con paso de mensajes. En [KUS] se describe un sistema multiprocesador que soporta ambos tipos de arquitectura.

Con la intención de evitar lo más posible la comunicación entre procesadores, la Red de Petri deberá ser particionada de forma que la ejecución de los subconjuntos obtenidos sea lo más independiente posible del resto.

Para aumentar la eficacia de la simulación, puede ser implementada una simulación no conservativa mediante el método Time Warp, donde la ejecución de los disparos es realizada sin garantizar el cumplimiento de las reglas de espera de Chandy-Misra.

Se obtiene más rapidez a costa de manejar grandes volúmenes de memoria que almacenen los diversos estados por los que pase la ejecución de la Red, con el objeto de retroceder a estados anteriores si la ejecución de algún evento ha sido incorrecta [RIG 89][BIL 85][JEF 83][CLE 88][FUJ 90a][FUJ 90b][JEF 85].

Referencias

Referencias

[ALV 93] G.A. Alverson, David Notkin. Program Structuring for Effective Parallel Portability. *IEEE Trans on Parallel and Distributed Systems*, vol. 4, nº 9, Sept. 1993, pp. 1041-1059.

[BAN 96] Jerry Banks, John S. Carsons, Barry L. Nelson. *Discrete-Event System Simulation*. Ed. Prentice-Hall, 1996.

[BIL 85] W. E. Biles, D. M. Daniels, T. J. O'Donell. Statistical considerations in simulation on a network of microcomputers. *Proceeding 1985 Winter Simulation Conference*, pp. 388-392.

[BIL 88] Jonathan Billington, Geoffrey R. Wheeler, Michael C. Wilbur-Ham. Protean: A High-level Petri Net Tool for the Specification and Verification of Communication Protocols. *IEEE Transactions on Software Engineering*, vol. 14, nº 3, Mar. 1988, pp. 301-316.

[CAS 88] Thomas L. Casavant, Jon G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, vol. 14, nº 2, Feb. 1988, pp. 141-154.

[CLE 88] J. Cleary, B. Unger, X. Li. A distributed and parallel back-tracking algorithm using virtual time. *Proceedings SCS Multiconference on distributed Simulation*, 1988, pp. 177-182.

[CHA 79] K. M. Chandy, J. M. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, vol. SE-5, Sept. 1979, pp. 440-452.

[CHA 81] K. M. Chandy, J. M. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communications of the ACM*, vol. 24, nº 11, Abril 1981, pp. 198-206.

[CHA 85] K. M. Chandy, Leslie Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, vol. 3, nº 1, Feb. 1985, pp. 63-75.

[CHA 88] M. Chandy. Distributed Simulation Tutorial. *SCS Multiconference on Distributed Simulation*, 1988.

[CHU 87] Wesley W. Chu, Lance M-T Lan. Task Allocation and Precedence relations for Distributed Real-Time Systems. *IEEE Transactions on Computer*, June 1987, pp. 667-679.

[DIJ 59] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, vol. 1, Oct. 1959, pp. 269-271.

[DUE 94] J. C. Dueñas. *Propotipado Concurrente de Sistemas de Tiempo Real.* Tesis Doctoral. Universidad Politécnica de Madrid. Junio 1994.

[EAG 86] Derek L. Eager, Edward D. Lazowska, John Zahorjan. Adaptative Load Sharing in Homogeneus Distributed Systems. *IEEE Transactions on Software Engineering*, vol. SE-12, nº 5, Mayo 1986, pp. 662-675.

[EFE 82] Kemal Efe. Heuristic Models of Task Assignment Scheduling in Distributed Systems. *IEEE Transactions on Computer*, June 1982, pp. 50-56.

[FER 87] G. Fernández, F. Saez Vacas. *Fundamentos de Informática.* Ed. Alianza-Informática, 1987.

[FID 91] Colin Fidge. Logical Time in Distributed Systems. *IEEE Computer*, Ag. 1991, pp. 28-33.

[FUJ 90a] R. M. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, vol. 33, nº 10, Oct. 1990, pp. 31-53.

[FUJ 90b] R. M. Fujimoto. Optimistic Approaches to Parallel Discrete Event Simulation. *Transactions of the Society for Computer Simulation*, vol. 7, nº 2, 1990, pp. 153-191.

[GHE 89] Carlo Ghezzi, Dino Mandroli, Sandro Morasca, Mauro Pezzè. A General Way to Put Time in Petri Net. *Proceedings of the Fifth Intl. Workshop on Software Specifications and Design*, 1989, pp. 60-67.

[GHE 91] Carlo Ghezzi, Dino Mandroli, Sandro Morasca, Mauro Pezzè. A Unified High-Level Petri Net Formalism for Time-Critical Systems. *IEEE Transactions on Software Engineering*, vol. 17, nº 2, Feb. 1991, pp. 160-172.

[JEF 83] D. Jefferson, H. Sowizral. *Fast concurrent simulation using the time warp mechanism. Part I: Local control.* The Rand Corp, 1983.

[JEF 85] D. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, vol. 7, nº 3, Jul. 1985, pp. 404-425.

[JEN 87] K. Jensen. Colured Petri Nets. *Proceedings of Advances in Petri nets* 1986. Ed. New York: Springler-Verlag, 1987.

[KIR 83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by Simulated Annealing. *Science*, vol. 220, nº 4598, Mayo 1983, pp. 671-680.

[KNU 81] D. E. Knuth. *Seminumerical Algorithms*, Addison-Wesley, vol. 2, 1981.

[KUS] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Somoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, J. Hennesy. *The Stanford FLASH Multiprocessor.* Computer Systems Laboratory, Stanford University.

[LAM 78] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed Systems. *Communications of the ACM*, vol. 21, nº 7, Jul. 1978, pp. 558-565.

[LAK 92a] C.A. Lakos. *LOOPN User Manual.* Department of Computer Science of University of Tasmania, Mar. 1992.

[LAK 92b] C.A. Lakos. *LOOPN System Manual.* Department of Computer Science of University of Tasmania, Abril 1992.

[LAK 94] C.A. Lakos, C. D. Keen. *LOOPN++: A New Language for Object-Oriented Petri Nets*". Department of Computer Science of University of Tasmania, Abril 1994.

[LIN 93] Yi-Bing Lin. Parallelism Analyzers for Parallel Discrete Event Simulation. *ACM Trans. Model. Comput. Simulation*, vol. 2, nº 3, 1993, pp. 239-264.

[LO 88] Virginia Mary Lo. Heuristic Algorithms for Task Assignment in Distributed Systems. *IEEE Transactions on Computer*, vol. 37, nº 11, Nov. 1988, pp. 1384-1397.

[MA 82] Perng-Yi Richard Ma, Edward Y. S. Lee, Masahiro Tsuchiya. A Task Allocation Model for Distributed Computing Systems. *IEEE Transactions on Computer*, 1982.

[MAT 89] Friedemann Mattern. Virtual Time and Global States of Distributed Systems. *Parallel and Distributed Algorithms.* Elsevier Science Publisher B.V. (North-Holland), 1989, pp. 215-226.

[MIS 86] J. Misra. Distributed-Discrete Event Simulation. *ACM Computer Surveys*, vol. 18, nº 1, Mar. 1986, pp. 39-65.

[MOR 96] Donald J. Morton, J. M. Tyler. Minimizing Development Overhead with Partial Parallelization. *IEEE Parallel and Distributed Technology*, 1996, pp. 15-24.

[MOR 85] Carroll Morgan. Global and Logical Time in Distributed Systems. *Information Processing Letters*, nº 20, Mayo 1985, pp. 189-194. Elsevier Science Publisher B.V.

[MUR 89] T. Murata .Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, vol. 77, nº 4, Abril 1989, pp. 541-580.

[PAP 90] C. Papadimitriou, M. Yannakakis. Towards an Architecture-Independent Analysis of Parallel Algorithms. *SIAM J. Computer*, vol. 19, 1990, pp. 322-328.

[PET 77] James L. Peterson. Petri Nets. *Computing Surveys*, vol. 9, nº 3, Sept. 1977, pp. 223-252.

[PET 85] James L. Peterson, Abraham Silberschatz. *Operating System Concepts*, Ed. Addison-Wesley, 1985.

[PUE 93a] J. A. de la Puente, A. Alonso, G. León, J.C. Dueñas. Distributed Execution of Specifications. *Real Time Systems*, vol. 5, nº 2/3, Mayo 1993, pp. 213-234.

[PUE 93b] J. A. de la Puente, A. Alonso, G. León, J.C. Dueñas. Distributed Simulation of Discrete Event Systems Modelled by Time Petri Nets. *Proceedings of the Second European Control Conferende*, vol. 1, June 1993, pp. 139-143.

[PUE 97] F. de la Puente, J. D. Sandoval, P. Hernández, F. Herrera. Parallel Simulation for Queueing Networks on Multiprocessor Systems. *Proceedings of the 5th Euromicro Workshop on Parallel and Distributed Processing*, Enero 1997.

[RAM 80] C. V. Ramamoorthy, Gary S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE Transactions on Software Engineering*, vol. SE-6, nº 5, Sept. 1980, pp. 440-449.

[RIG 89] R. Righter, J. C. Walrand. Distributed Simulation of Discrete Event Systems. *Proceedings of the IEEE*, vol. 77, nº 1, Ene. 1989, pp. 99-113.

[SAN 95] J. D. Sandoval, P. Hernández, F. Herrera. Algorithm for Parallel Simulation of High Level Petri Nets. *Proceedings of the Thirteenth IASTED International Conference on Applied Informatics*, Feb. 1995, pp. 346-349.

[SAN 96] J. D. Sandoval, P. Hernández, F. Herrera, F. Puente. Multithread Object Oriented Simulation. *Congreso Internacional de IASTED Modelling, Simulation and Optimization*, 1996.

[SAR 86] Vivek Sarkar, John Hennesy. Compile-time Partitioning and Scheduling of Parallel Programs. *Proceedings of the SIGPLAN 86 Symposium on Compiler Construction*, ACM Press, New York, 1986, pp. 17-26.

[SAR 89] Vivek Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*, MIT Press, Cambridge, Mass, 1989.

[SHI 90] Behrooz Shirazi, Mingfang Wang, Girish Pathak. Analysis and Evaluation of Heuristic Methods for Static Task Scheduling. *Journal of Parallel and Distributed Computing*, vol. 10, 1990, pp. 222-232.

[SHI 92] Niranjan G. Shivaratri, Phillips Krueger, Mukesh Singhal. Load Distributing for Locally Distributed Systems. *IEEE Transactions on Computer*, Dic. 1992, pp. 33-44.

[SIM 84] ----. The simulation of a master-slave event set processor. *Simulation*, vol. 42, nº 3, Marzo 1984, pp. 117-124.

[SIN 94] J. B. Sinclair, W. P. Dawkins. ES: A Tool for Predicting the Performance of Parallel Systems. *Proceedings of the Second International Workshop on Modelling, Analysis and Simulation of Computer and telecomunication Systems*, 1994, pp. 164-168.

[SOL 95] Viliam Solcany, Jiri Safarik. Parallel Analyzer for Conservative Parallel Discrete Event Simulation. *Proceedings of the Thirteenth IASTED International Conference on Applied Informatics*, pp. 350-353.

[SUN 95] Xian-He Sun, Jiamping Zhu. Performance Considerations of Shared Virtual Memory Machines. *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, nº 11, Nov. 1995, pp. 1185-1194.

[TPROF] *Turbo Profile version 2.0 User's Guide*. Borland.

[VAZ 90] E. Vázquez Gallo, A. Martínez Más . *Simulación de Sistemas*. Programa de Postgrado en Sistemas y Redes de Comunicaciones. Departamento de Ingeniería de Sistemas Telemáticos de la UPM, 1990.

[WAN 85] Yung-Terng Wang, Robert J. T. Morris. Load Sharing in Distributed System. *IEEE Transactions on Computer*, vol. c-34, nº 3, Marzo 1985, pp. 204-217.

[WIL 93] M. H. Willebeck-LeMair, A. P. Reeves. Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, nº 9, Sept. 1993, pp. 979-993.

[WON 95] Yung-Chang Wong, Shu-Yuen Hwang, Jason Yi-Bing Lin. A Parallelism Analyzer for Conservative Parallel Simulation. *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, nº 6, Jun. 1995, pp. 628-638.

[ZHU 95] Weiping Zhu, C. F. Steketee. An Experimental Study of Load Balancing on Amoeba. *IEEE 1995*, pp. 220-226.