"UN MÉTODO PARA LA ESPECIFICACIÓN Y VALIDACIÓN DE LA ESTRUCTURA LÓGICA DE LOS PROTOCOLOS DE COMUNICACIÓN"

GARCÍA-HOFFMANN, M.

En este trabajo se presentan dos aspectos de una metodología nueva de diseño de protocolos, concretamente, los referentes a la especificación y validación de la estructura lógica de protocolos de comunicación.

Para especificar el funcionamiento de los elementos integrantes de los protocolos se crea un lenguaje descriptivo de alto nivel, el LDP. Las descripciones de protocolos en LDP son el punto de partida de los algoritmos de validación que se proponen.

La validación se basa en la construcción de un árbol que expresa todos los posibles caminos de ejecución del protocolo y permite detectar errores en su estructura lógica tales como bloqueos, bloqueos temporales, comportamiento cíclico, etc.

1. INTRODUCCION

El diseño correcto de protocolos de comunicación es de gran importancia para el desarrollo de la informática distribuida y, particularmente, de las redes de computadores /24/. Por protocolo de comunicación se entiende el conjunto de reglas que rigen el intercambio de información entre procesos que cooperan en una actividad común /10/. El desarrollo de técnicas formales para la especificación, validación y verificación de protocolos de comunicación ha acaparado la atención de diversos grupos de investigadores en los últimos diez años. La forma de modelar y describir los protocolos de los diversos métodos propuestos está condicionada por los objetivos perseguidos con la modelación. A su vez, la forma que adopta la descripción condiciona el estudio de las características de los protocolos de comunicación. Así, cabe plantear dos cuestiones: ¿Cuáles son los objetivos que debe cumplir un modelo descriptivo de protocolos?, ¿Cuáles son las características de los protocolos que han de modelarse?.

El diseño correcto de protocolos de comunicación incluye diversos tópicos como son la especificación, validación de la estructura lógica, verificación, análisis cuantitativo del protocolo, simulación, realización práctica y documentación. El método descriptivo ha de permitir la especificación del funcionamiento del protocolo de forma clara y concisa. La descripción formal obtenida debe servir de base para el estudio de las propiedades de la especificación.

El modelo ha de poder representar los procesos integrantes del protocolo así como el entorno en el que está inmerso. Entre los aspectos que con más frecuencia han de modelarse se cuentan las estructuras de datos y de control de los procesos, las decisiones internas que estos toman en función del valor de ciertas variables, la forma de comunicación y sincronización entre los procesos, los medios de comunicación, los sucesos externos, las interfases entre los diversos niveles de las estructuras jerárquicas de protocolos, las temporizaciones, ...

En la validación se estudian las características estructurales del protocolo como son los bloqueos, bucles, comportamiento cíclico, etc., mientras que la verificación considera las propiedades que durante el diseño intentó imponer el diseñador al comportamiento del protocolo. Aun cuando la mayor parte de los métodos de verificación existentes incluyen la validación del protocolo es interesante considerar ambos aspectos por separado. Por una parte, la aplicación de los métodos de verificación a protocolos de cierta com-

- M. García Hoffmann, de la Càtedra de Mètodes Informàtics de la ETSEIB. Av. Diagonal, 647. Barcelona - 28.
- Article rebut el Febrer de 1980.

plejidad resulta en ocasiones muy costosa, - cuando no imposible, por lo que habremos de conformarnos con validar la estructura lógica del protocolo. Por otra, las técnicas de verificación existentes pueden contemplarse como extensiones de las de validación de la estructura lógica.

En este artículo se describe una metodología para modelar y especificar el funcionamiento de protocolos así como la forma de validar - las descripciones obtenidas. En /13/ se expone una técnica de verificación de las descripciones de protocolos y en /12/ se considera la realización práctica de las descripciones verificadas a que conduce la metodología propuesta. En /10/ y /11/ puede encontrarse un estudio crítico de las técnicas de descripción, validación y verificación de protocolos más relevantes.

2. EL MODELO

Un protocolo se considera formado por un conjunto de procesos secuenciales, distribuidos geográficamente o no, que interaccionan entre sí. Los procesos son disjuntos, es decir, no tienen acceso a variables comunes. Los distintos procesos que integran un protocolo se ejecutan en procesadores independientes. Dado que los procesadores carecen de memoria común, los procesos se comunican a través de mensajes. La comunicación se realiza síncronamente; cuando un proceso está en condiciones de enviar un mensaje a otro y éste puede recibirlo se realiza la transferencia. Si uno de los procesos no está dispuesto para - realizar la transferencia la ejecución del - otro se detiene. El hacer esperar a un proceso que desea transmitir un mensaje hasta que el proceso destino está dispuesto a aceptarlo refleja correctamente la forma de comunicación real en la mayor parte de los casos. De todas formas, siempre que sea necesario - pueden definirse específicamente procesos que modelen otra forma de comunicación tales como buffers, monitores, colas, etc.

El medio que utilizan los procesos para comunicarse se modela como un proceso más del protocolo. El lenguaje empleado en la descripción de los procesos permite modelar fácilmente las características del medio de comunicación, entre las que destacan la alteración de la información, la pérdida, duplicación y reordenación de los mensajes.

Las temporizaciones se modelan también como procesos. La especificación de las interfases entre los procesos que forman un protocolo y los procesos de otros niveles de la jerarquía se realiza describiendo el comportamiento de estos sin considerar los detalles de su funcionamiento que sean ajenos a la interconexión.

3. EL LENGUAJE DE DESCRIPCION DE PROTOCOLOS

Existen un cierto número de métodos formales para la descripción de procesos paralelos. - Algunos de ellos son adaptaciones o extensiones de lenguajes de alto nivel ya existentes /5/, /17/, /18/ mientras que otros han sido especificamente diseñados para describir el paralelismo o estudiar las características - de la comunicación /3/, /14/, /15/, /16/, /19/, /20/. Los protocolos de comunicación - son un caso particular de procesos paralelos por lo que los métodos propuestos en las referencias indicadas pueden utilizarse para - los protocolos. Sin embargo es conveniente - disponer de un método orientado a la descripción de protocolos que sirva de base al estudio de su funcionamiento.

A continuación se exponen las características del lenguaje para la descripción de protocolos, el LDP, inspirado en los lenguajes propuestos por Hoare /20/ y Hansen /19/ para la descripción de procesos paralelos. - El lenguaje es una adaptación de las ideas - de Hoare a las necesidades pecualiares de los protocolos de comunicación. Debe entenderse que el LDP no es un lenguaje ejecutable sino tan solo una herramienta para la especificación formal de protocolos.

La figura 1 recoge parcialmente la descripción BNF de las sentencias del LDP. Una sentencia define la acción a realizar por el procesador que la ejecuta. La composición secuencial de sentencias se indica mediante el signo ; y su ejecución se realiza secuencialmente y en orden. La sentencia nula no tiene efecto alguno, su ejecución es siempre correcta y no modifica el estado de ningún proceso. La ejecución de la sentencia de asignación, si no falla, puede modificar el estado del -

SENTENCIAS

<sentencia>::=<sentencia sin etiqueta>|<etiqueta de sentencia>:
<sentencia sin etiqueta>
<etiqueta de sentencia>::=<entero sin signo>
<sentencia sin etiqueta>::=<sentencia simple>|<sentencia estructurada>
<sentencia simple>::=<sentencia vacia>|<sentencia de asignacion>
<sentencia vacia>::=skip
<sentencia estructurada>::=<sentencia alternativa>|<sentencia repetitiva>|
<sentencia paralela>
<lista de sentencias>::={<declaracion>;|<sentencia>;}<sentencia>

SENTENCIA PARALELA

<sentencia paralela>::=[<proceso>{||<proceso>}]
<proceso>::=<etiqueta de proceso>::<lista de sentencias>
<etiqueta de proceso>::=<identificador con mayusculas>

SENTENCIA DE ASIGNACION

<sentencia de asignacion>::=<sentencia de asignacion interna>|
<sentencia de asignacion externa>
<sentencia de asignacion interna>::=<variable>:=<expresion>|<variable
simple booleana>:=#
<variable>::=<variable simple>|<variable estructurada>
<variable estructurada>::=<constructor>(<lista de variables>)
<costructor>::=<vacio>|<identificador>
<lista de variables>::=<vacio>|<variable>{,<variable>}
<expresion>::=<expresion simple>|<expresion estructurada>
<expresion estructurada>::=<constructor>(<lista de expresiones>)
<lista de expresiones>::=<vacio>|<expresion>{,<expresion>}
<sentencia de asignacion externa>::=<sentencia de entrada>|<sentencia
de salida>
<sentencia de entrada>::=<variable>:=<etiqueta de proceso origen>
<sentencia de salida>::=<etiqueta de proceso destino>:=<expresion>
<etiqueta de proceso origen>::=<etiqueta de proceso>
<etiqueta de proceso destino>::=<etiqueta de proceso>

SENTENCIA ALTERNATIVA

<sentencia alternativa>::=[<comando con guardia>{[]<comando con guardia>}]
<comando con guardia>::=<guardia>→<lista de sentencias>
<guardia>::=<lista de guardias>|<lista de guardias>;<sentencia de
asignacion externa>|<sentencia de asignacion externa>
<lista de guardias>::=<elemento de guardia>{;<elemento de guardia>}
<elemento de guardia>::=<elemento de guardia incondicional>|<elemento
de guardia condicional>
<elemento de guardia condicional>::=$<elemento de guardia incondicional>
<elemento de guardia incondicional>::=<expresion booleana>|<declaracion>

SENTENCIA REPETITIVA

<sentencia repetitiva>::=*<sentencia alternativa>

Figura 1: Descripción BNF parcial del LDP

proceso en que se encuentra, el estado de -- otro proceso, o ambos.

La sentencia paralela es similar al grupo -- parbegin-parend propuesto por Dijkstra y especifica la ejecución concurrente de todos - los procesos que la forman. La ejecución de los procesos se inicia simultáneamente y cada uno de ellos continúa ejecutando su lista de sentencias hasta que ésta se acaba. La -- ejecución de una sentencia paralela termina cuando terminan todos sus procesos constituyentes. La lista de sentencias que forman un proceso va siempre precedida de un identificador que da nombre al proceso. A fin de distinguir el nombre de un proceso de una variable en las sentencias de asignación externa, los identificadores de proceso se escribirán con mayúsculas. El nombre de una lista de -- sentencias no debe confundirse con el nombre de un proceso.

La sentencia de asignación interna, excepto en el caso de la función #, tiene el sentido usual de la asignación. El valor calculado - por la expresión se asigna a la variable que aparece a la izquierda del signo :=. Este valor, así como la variable a que está destinado, puede ser simple o estructurado. El resultado de una expresión simple puede ser -- simple o estructurado, según sea el carácter de los operandos. Por el contrario, una expresión estructurada evalúa un valor estructurado cuyos elementos son los valores calculados por cada expresión de la lista de expresiones. La asignación no puede llevarse a cabo si alguno de los valores queda indefinida o si éstos no pueden pasar a ocupar la variable que indica la sentencia de asignación.

Una variable simple puede recibir un valor - simple o estructurado, siempre que sean del mismo tipo. En cuanto a las variables estructuradas, pueden recibir tan sólo un valor estructurado con el mismo constructor y tal -- que el número y tipo de sus elementos constituyentes coincida con el número y tipo de -- los elementos que forman la variable. El valor estructurado que no tiene componentes -- sirve para modelar señales, cuyo nombre es - el de su constructor.

La "función incorporada" denotada por el signo # produce de forma aleatoria un valor -- cierto o falso que se asigna a la variable - booleana de la izquierda del signo :=. Esta función sirve para modelar sucesos de los -- que se desconoce la función de distribución de su aparición.

Ejemplos de sentencias de asignación interna:

(a) último no aceptado:=número de aceptación+1

El valor de la variable último no aceptado tras la asignación es el mismo que el de la variable número de aceptación más una unidad, antes de la asignación.

(b) bits:=bloque(datos,número de secuencia)

Con el constructor bloque forma un valor estructurado y lo asigna a la variable - bits.

(c) bloque(a,b):=bits recibidos

Si bits recibidos tiene la forma bloque (c,d) y los tipos de a,c y b,d coinciden, se asigna c a a y d a b; en caso contrario la asignación no puede realizarse.

(d) mensaje(a):=paquete(a)

La asignación no puede realizarse debido a la diferencia de constructores.

(e) acción:=marcha()

Asigna a la variable acción la señal de constructor marcha.

(f) pérdida de mensaje:=#

Asigna a la variable pérdida de mensaje un valor cierto o falso.

En el modelo empleado los procesos se ejecutan en procesadores independientes que carecen de memoria común. La transferencia de información de un proceso a otro puede contemplarse como la asignación de un determinado valor calculado en el proceso origen a una - variable situada en el proceso de destino. - La asignación sólo podrá llevarse a cabo cuando los procesos origen y destino estén dispuestos a realizarla y en el supuesto de que la variable y la expresión sean compatibles. Esto quiere decir que la ejecución de una -- sentencia de asignación externa queda retra-

sada hasta que el proceso correspondiente esté en condiciones de ejecutar una sentencia de asignación externa compatible. Cuando ambos están dispuestos la asignación se realiza. Concretamente, la ejecución de una sentencia de salida (entrada) de un proceso A hacia (desde) un proceso B se ejecuta simultáneamente con una sentencia de entrada (salida) del proceso B desde (hacia) el proceso A.

Ejemplo de sentencias de asignación externa:

(a) [A::EMISOR || B::RECEPTOR]

Si la lista de sentencias denominada EMISOR intenta ejecutar una sentencia:

```
B:=mensaje para enviar
```

podrá hacerlo cuando en la lista RECEPTOR pueda ejecutarse una sentencia:

```
mensaje recibido:=A
```

Ambas sentencias se ejecutarán simultáneamente y el resultado será la asignación del valor de la variable mensaje para enviar del proceso A a la variable mensaje recibido del B. La asignación fallará si el valor y la variable no son compatibles. Cualquiera de los dos procesos que esté en condiciones de ejecutar la sentencia de asignación externa deberá detener su ejecución hasta que el otro esté dispuesto a realizarla.

(b) [A::PROCESO || B::RELOJ]

Si A desea ejecutar B:=marcha() podrá hacerlo simultáneamente con la ejecución de orden :=A en B. El resultado es la asignación de la señal de constructor marcha a la variable orden.

La sentencia alternativa es la forma empleada para introducir el no determinismo en el LDP. Su funcionamiento se basa en los comandos con guardia de Dijkstra /7/. La sentencia alternativa especifica la ejecución de una única lista de sentencias cuya guardia evalue a cierto. Su ejecución comienza evaluando las guardias. La evaluación de una guardia se realiza elemento por elemento y de izquierda a derecha. En el caso de que todas las guardias evaluen un valor falso la sentencia alternativa produce un error. Si más de una guardia es cierta se escoge arbitrariamente para su ejecución una de las listas de sentencias de guardia cierta.

La evaluación de los elementos de la guardia no tiene un efecto de asignación sobre sus elementos constituyentes, excepto si se trata de una sentencia de asignación externa. El símbolo $ precediendo un elemento de guardia pospone su evaluación hasta que el valor sea cierto, posponiendo por tanto la evaluación de la guardia. Obsérvese que esta estructura de control recoge características de la estructura if y de la when, sugeridas por Hansen /19/ y otros.

Como último (o único) elemento de una guardia puede figurar una sentencia de asignación externa. Esta pospone la evaluación de la guardia hasta que pueda ejecutarse conjuntamente con una sentencia de asignación externa del proceso correspondiente. Cuando todos los elementos de la guardia evaluan un valor cierto y la sentencia de entrada/salida puede ejecutarse, la guardia evalua un valor cierto. Si la sentencia alternativa selecciona este comando para ser ejecutado, la entrada/salida se realiza pasando el control a la lista de sentencias del comando. El permitir la presencia de tan solo una sentencia de asignación externa en la guardia se debe a su carácter ejecutable; los procesos individuales son secuenciales y, por tanto, sólo pueden ejecutar una sentencia de entrada o salida en un instante dado.

Ejemplos de sentencias alternativas:

(a) Un proceso contiene las sentencias

```
error:=#;[error→LISTA1 ▯ error→LISTA2]
```

Según sea el valor asignado por la función # a la variable error se ejecutará la lista de sentencias LISTA1 o la LISTA2.

(b) Un proceso A que ha enviado un mensaje a un proceso B está esperando la aceptación. Si ésta no llega en un tiempo determinado el proceso C produce una señal de temporización. El proceso A contiene la sentencia

```
[aceptacion:=B →LISTA1 [] temporizacion:=C →LISTA2]
```

La sentencia repetitiva ejecuta, mientras -- sea posible, los comandos con guardia que -- forman la sentencia alternativa y tan sólo - termina, sin efecto alguno, cuando todas las guardias evaluan un valor falso.

Ejemplos de sentencias repetitivas:

(a) Un programa A solicità datos de un fichero en disco a través de un proceso intermedio que se modela

```
*[leer:=A→BUSQUEDA;A:=dato]
```

(b) Un proceso B recibe mensajes numerados - pero desordenados de A. Al recibir un -- mensaje lo almacena temporalmente en temporal (número) y deja constancia de la - recepción haciendo cierta la variable -- recibido (número). Se le exige que suministre los mensajes a C con la numeración correlativa. El proceso B podrá expresarse

```
...B::*[mensaje(dato,numero):=A →
        temporal(numero):=dato;
        recibido(numero):=cierto;
        *[recibido(siguiente) →
        C:=temporal(siguiente);
        siguiente:=siguiente+1]]
```

La variable siguiente indica el número - del mensaje que requiere C (número del - último mensaje pasado a C más uno).

4. UTILIZACION DEL LDP

En este apartado se consideran algunos ejemplos de modelación y descripción de protocolos mediante el LDP con el fin de familiarizar al lector con este lenguaje. Los protocolos escogidos como ejemplo aparecen descritos en la literatura anglosajona especializada por lo que he preferido conservar los anglicismos en las descripciones para facilitar el estudio comparativo.

4.1 DESCRIPCIÓN DE PROTOCOLOS

Considérese el protocolo HDLC (High Level Data Link Control) de la ISO (International -- Standards Organization) /21/, en su forma -- NRM (Normal Response Mode), ampliamente utilizado en redes existentes e inspirador de - numerosos protocolos de control de línea como son el SDLC (SYnchronous Data Link Control) de IBM /9/ y el LAP (Link Access Procedure) del nivel 2 de la recomendación X.25 de la - CCITT (International Telegraph and Telephone Consultative Commitee) /4/. Para simplificar se consideran tan solo tramas de información y se organiza el buffer de mensajes para enviar como un buffer infinito.

El protocolo está orientado a la transmisión simultánea de información en los dos sentidos sobre un enlace punto a punto. En cada - estación coexisten cinco procesos:

(a) Proceso S: genera los mensajes que han - de enviarse.

(b) Proceso D: consume los mensajes enviados por la otra estación.

(c) Proceso T: controla la transmisión de -- los bloques sobre la línea.

(d) Proceso R: controla la recepción y detecta los errores introducidos - por la línea.

(e) Proceso P: responsable del protocolo.

La figura 2 muestra esquemáticamente la relación entre los procesos.

Cada bloque enviado consta de un campo con - el número de secuencia del mensaje, un campo con el mayor número de secuencia de los mensajes correctamente recibidos y pasados al - proceso consumidor, un campo con el mensaje y un campo con información que permite detectar errores de transmisión. El proceso P dispone de un buffer infinito donde almacena -- los mensajes entregados por S para su transmisión. Los parámetros que indican el estado de la transmisión son:

(a) s: número de secuencia del mensaje enviado en el último bloque.

(b) r: número de secuencia del último mensaje recibido correctamente y pasado al proceso consumidor.

(c) m: constante que indica la cantidad de mensajes con numeración correlativa que pueden enviarse sin recibir la aceptación.

(d) a: número mayor de secuencia de los mensajes recibidos correctamente por la estación opuesta de que se tiene constancia (aceptación recibida).

El funcionamiento del protocolo se resume en los puntos siguientes:

(a) Se verifica la corrección de los bloques que llegan.

(b) Si el bloque es correcto se compara el número de secuencia (sr) con el número del mensaje esperado (r+1). Si el mensaje es el esperado y el proceso consumidor está dispuesto a aceptarlo se entrega; en caso contrario el mensaje se ignora.

(c) El campo rr del bloque indica la aceptación de la estación opuesta de todos los mensajes enviados con número de secuencia menor o igual a rr. Por tanto a se actualiza tomando el valor máx(a,rr).

(d) Cuando el número de mensajes enviados sin aceptación se hace igual al valor de una constante m se retransmiten todos los mensajes con número de secuencia superior a a.

(e) Si una estación no dispone de mensajes nuevos para transmitir, retransmite el último bloque enviado.

La figura 3 constituye la descripción LDP de una estación.

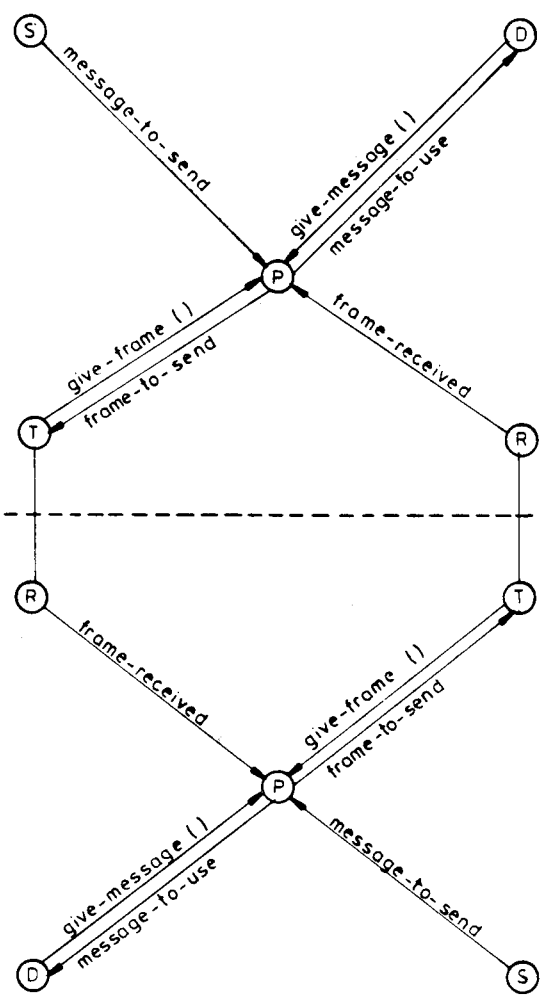


Figura 2: Relación entre los procesos del protocolo HDLC considerado.

4.2 MEDIOS DE COMUNICACIÓN

Los procesos geográficamente distribuidos utilizan un medio de comunicación para intercambiar información. El medio puede ser un simple canal para protocolos que enlazan procesos sobre un enlace punto a punto o multipunto, o puede tratarse de una red de conmutación de paquetes formada por decenas de computadores para protocolos que controlan el flujo de información entre dos procesos localizados en computadores diferentes.

La figura 4 muestra la modelación de tres enlaces simples. Obsérvese que los tres modelos suponen la posibilidad de que la línea puede almacenar la información list hasta que el proceso receptor esté dispuesto a tomarla. Por otra parte, el proceso productor ha de esperar a que la línea acepte la información. Esta situación no es realista si se considera exclusivamente la línea física. En la práctica el enlace de comunicaciones está formado por la línea mas dos procesos -hardware o software- destinados respectivamente a controlar la emisión y la recepción. El modelo que se adopte dependerá del nivel de detalle necesario para el estudio. Considérese, por ejemplo, un proceso S que genera y envía bytes asíncronamente a un proceso D emplean-

```
[S::SOURCE || P::PROTOCOL || T::TRANSMITTER || R::RECEIVER || D::DESTINATION]

PROTOCOL

l:=s:=a:=r:=0;

*[message to send:=S→l:=l+1;send buffer(l):=message to send

 []give message():=D→twaiting:=true

 []give frame():=T→[s=l∨s=a+m→[s=a→skip[] s≠a→s:=a+1]] [] s<l∧ s<a+m→s:=s+1];

                    T:=frame(s,r,send buffer(s))

 []frame(sr,rr,message rec):=R→[rr≤a→skip[] rr>a→a:=rr];

    [sr≠r+1∨¬twaiting→skip

    []sr=r+1∧twaiting→r:=r+1;D:=message rec;twaiting:=false]

 ]

Bucle principal de SOURCE

...*[true→GENERATE MESSAGE;P:=generated message...

TRANSMITTER

*[P:=give frame()→frame to send:=P;TRANSMIT FRAME]

RECEIVER

*[true→LISTEN FRAME;error:=#;[¬error→P:=received frame[] error→skip]

Bucle principal de DESTINATION

...*[P:=give message()→message to use:=P;USE MESSAGE...
```

Figura 3: Descripción LDP del protocolo HDLC considerado.

Linea simplex ideal

```
*[list:=A→C:=list]
```

Linea half-duplex ideal

```
*[list1:=A→B:=list1 []list2:=B→A:=list2]
```

Linea full-duplex ideal

```
  L1::LINK1   L2::LINK2
LINK1              LINK2
*[list1:=A→B:=list1] *[list2:=B→A:=list2]
```

Figura 4: Enlaces simples

do dos UART's simples como los de la figura 5a. S puede cargar el registro t1 cuando éste se encuentre vacío. El biestable t1 full indica el estado, lleno o vacío, del registro t1. El registro t2 transmite su contenido en serie a la línea y cuando se vacía toma el contenido de t1 para transmitirlo. Cuando r2 recibe un byte se pasa a r1; si éste estaba lleno se produce un overrun del que queda constancia en un biestable. El estado de r1, lleno o vacío, se conoce por el biestable r1 full. La figura 5b muestra la relación entre dos procesos, y la 6 recoge la descripción LDP de su funcionamiento. En ella se observa que las UART's se modelan mediante dos procesos paralelos. En la UART encargada de la transmisión estos procesos corresponden a la carga del registro t1 desde el proceso S y la transferencia de éste al registro t2, por un lado, y la transmisión del contenido de t2 sobre la línea, por otro. Efectivamente, estas dos acciones pueden realizarse al mismo tiempo. Obsérvese que la carga de t1 y la transferencia de ésta a t2 son acciones que se excluyen al encontrarse en distintos comandos con guardia de una misma sentencia alternativa. Análogamente, la UART que controla la recepción consta de dos procesos que se ejecutan en paralelo. Uno es la recepción en r2 del byte que viaja por la línea y su transferencia a r1. El otro se encarga de entregar el contenido de r1 a D. En este segundo proceso la acción que entrega el contenido de r1 a D y la que carga r1 desde r2 son excluyentes.

Las alteraciones de la información que con más frecuencia interesa modelar son los errores de transmisión y la pérdida de información. La figura 7 considera dos ejemplos de modelización de estas características que no necesitan más aclaración.

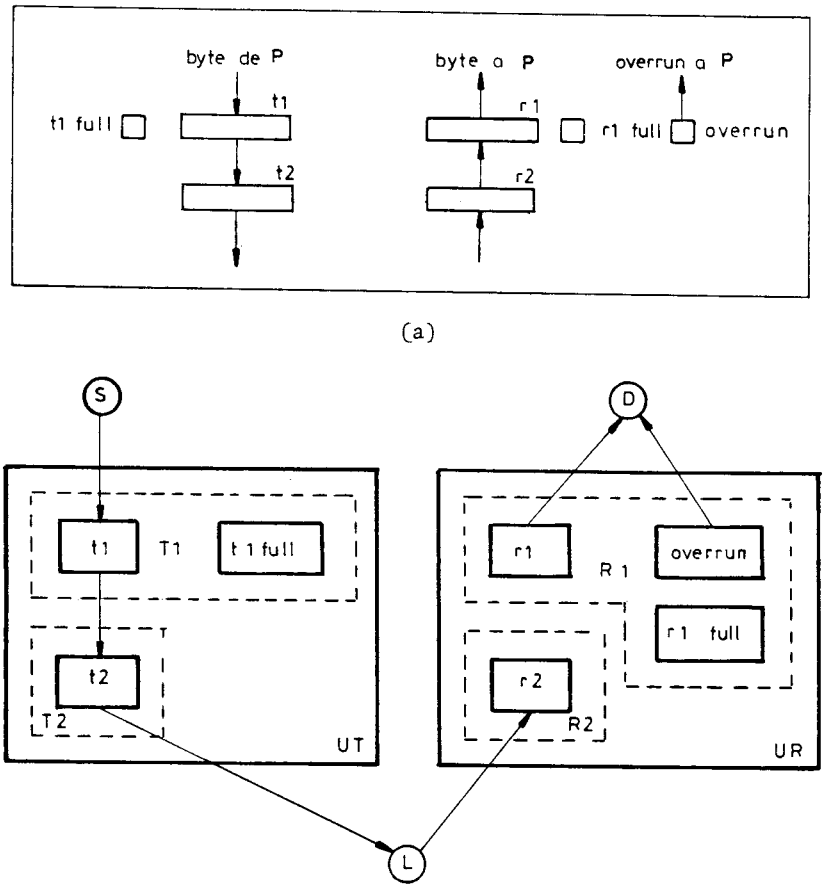


Figura 5: (a) UART's empleadas en el ejemplo
(b) Relación entre los procesos de este ejemplo.

El estudio de ciertos protocolos empleados - en las redes de computadores para controlar el flujo de información entre procesos requiere la modelación del medio virtual de comunicaciones que utilizan. El medio en estos casos es con frecuencia una red de conmutación de mensajes o de paquetes con capacidad de - almacenamiento y que puede alterar, perder, reordenar y duplicar los mensajes. El estu-- dio del comportamiento de estas redes cae -- fuera de la perspectiva de este trabajo, remitiéndose al lector interesado a /6/, /8/, /22/, /27/ y /28/. Sin embargo, el LDP permite describir el comportamiento del medio tal como lo ven los procesos que lo utilizan sin entrar en los detalles de su funcionamiento. Este aspecto de la modelación se considera - más adelante.

4.3 TEMPORIZACIONES

Los protocolos tipo PAR (Positive Acknowledgment Retransmission on time-out) tienen una gran difusión /29/. En ellos, cuando un proceso envía un mensaje espera recibir la confirmación del proceso destino indicando la - recepción correcta. Si la aceptación no llega en un tiempo prefijado el mensaje se retransmite. La definición LDP no ha sido restringida por ninguna noción particular de -- tiempo, por lo que la modelación de las temporizaciones resulta un tanto artificiosa. - La noción de tiempo en el LDP se introduce - en el apartado 5, sin embargo, resulta conveniente ver en este apartado la forma de modelar temporizaciones mediante un proceso que simula un reloj a través de un ejemplo que -

```
[S::SOURCE || UT::UART TRANS || L::LINE || UR::UART REC || D::DESTINATION]

SOURCE
...*[UT:=byte→GENERATE NEXT BYTE...

UART TRANS
...[T1::*[¬t1full∧t1:=S→t1full:=true
        [] T2:=t1→t1full:=false]
  ||T2::*[t2:=T1→L:=t2]
  ]

LINE
*[byte:=UT→UR:=byte]

UART REC
...[R1::*[r1full∧D:=(r1,overrun)→r1full:=overrun:=false
        [¬r1:=R2→[r1full→overrun:=true[]¬r1full→r1full:=true]
  ||R2::*[r2:=L→R1:=r2]
  ]

DESTINATION
...*[(byte,overrun):=UR→USE BYTE...
```

Figura 6: Descripción LDP de los procesos de la figura 5.

será utilizado en el resto de este artículo para la exposición de diversos tópicos.

El ejemplo escogido es una simplificación del protocolo de bit alternante del National Physical Laboratory /1/, /2/. El protocolo es responsable de controlar la transferencia alternada de mensajes entre una estación emisora S y una receptora R sobre un enlace half-duplex M que puede alterar o perder la información. El emisor aguarda la recepción de un mensaje de aceptación notificando la correcta recepción en R del último mensaje transmitido para proceder al envío del mensaje siguiente. Los mensajes transmitidos por cada estación contienen información para detectar errores producidos en la comunicación y un bit de control denominado el bit alternante. Cada estación conserva el valor del bit alternante del último mensaje enviado, alternando su valor en cada nuevo mensaje transmitido; el cambio de valor del bit alternante indica la aceptación del mensaje. El protocolo detecta los errores de transmisión y, mediante un temporizador, la pérdida

```
[S::SOURCE || M::MEDIUM || D::DESTINATION]

SOURCE
...*[M:=data→GENERATE NEW DATA...
MEDIUM
*[data:=S→error:=#;D:=(data,error)]
DESTINATION
...*[(data,error):=M→USE DATA...
```

(a)

```
[S::SOURCE || M::MEDIUM || D::DESTINATION]

SOURCE
...*[M:=data→GENERATE NEW DATA...
MEDIUM
*[data:=S→loss:=#;[loss→skip [] ¬loss→D:=data]
DESTINATION
...*[data:=M→USE DATA...
```

(b)

Figura 7: (a) Medio con posibilidad de alterar la información.
(b) Medio con posibilidad de perder la información.

de los mensajes. En ambos casos el emisor retransmite el último mensaje. La figura 8 recoge la descripción LDP del protocolo y el significado de las variables de los procesos S y R. Como puede verse el medio se modela como un proceso que puede alterar o perder la información. Para simplificar se ha considerado que la guardia $time toma un valor cierto cuando todos los otros procesos se encuentran bloqueados por la pérdida de un mensaje. Tampoco he considerado la forma de comunicación entre S y R y sus respectivos usuarios, de modo que el empleo de los vectores indata y outdata pueden parecer artificiales.

Las sentencias de S

```
...datas:=indata(pt);pt:=pt+1
```

puede sustituirse por

```
datas:=US
```

donde US sería el proceso usuario de S, generador de los datos que han de transmitirse. Pueden hacerse consideraciones análogas sobre R y su vector outdata.

4.4 DESCRIPCION JERARQUICA DE PROTOCOLOS

Los trabajos actuales sobre estandarización de protocolos coinciden en concebirlos como una estructura multinivel jerarquizada. La figura 9 muestra la división en niveles más extendida. Cada nivel crea un canal virtual de comunicaciones de características determinadas que puede ser utilizado por el nivel superior sin que éste tenga que considerar los detalles de realización del canal. Para estudiar las propiedades de un protocolo situado a un nivel determinado se define, además de su funcionamiento, el entorno en el que está inmerso. Para ello basta definir las interfases con los procesos con lo que coopera sin tener que considerar los detalles de la realización de estos. Hansen ha empleado una técnica similar en la descripción del sistema operativo SOLO /18/, utilizando el lenguaje Concurrent Pascal.

5. ALGUNAS FORMALIZACIONES

Antes de pasar a considerar la validación de protocolos descritos en LDP conviene introducir algunos conceptos y formalizaciones. Este es el objetivo del presente apartado.

5.1 ESTADO DE UN PROTOCOLO

En el método de estudio y diseño de protocolos que se está exponiendo las descripciones son algorítmicas, pero es necesario además - caracterizar la noción de estado del protoco

```
[S::SENDER || M::MEDIUM || R::RECEIVER || C::CLOCK]

SENDER
...*[true →
      [¬ea ∧ ack=seq → datas:=indata(pt);pt:=pt+1;seq=¬seq
      [] ea ∨ ack≠seq → skip];M:=(datas,seq);C:=start();
      [ack:=M → C:=stop();ea:=# [] timeout():=C → skip]
    ]

MEDIUM
*[(datam,seqm):=S → lossm:=#;[¬lossm → R:=(datam,seqm) [] lossm → skip]
 [] expm:=R → lossa:=#;[¬lossa → S:=expm [] lossa → skip]
 ]

RECEIVER
...*[(datar,seqr):=M → em:=#;
      [¬em ∧ seqr≠exp → outdata(ps):=datar;ps:=ps+1;exp:=¬exp
      [] em ∨ seqr=exp → skip];M:=exp
    ]

CLOCK
*[start():=S → [stop():=S → skip [] $time → S:=timeout()]]
```

Figura 8: Descripción LDP del protocolo de bit alternante..

SENDER

seq	bit alternante de S
ack	valor del bit alternante recibido por S
datas	datos enviados por S
indata	vector conteniendo los datos que han de transmitirse
pt	puntero de indata
ea	error detectado por S en la recepción

RECEIVER

exp	bit alternante dr R
seqr	valor del bit alternante recibido por R
datar	datos recibidos por S
outdata	vector conteniendo los datos correctamente recibidos
ps	puntero de outdata
em	error detectado por R en la recepción

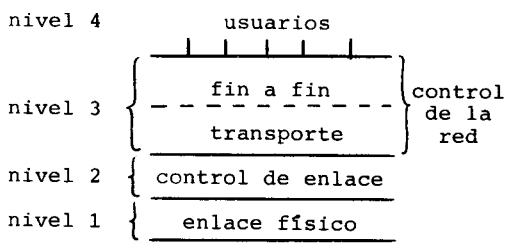
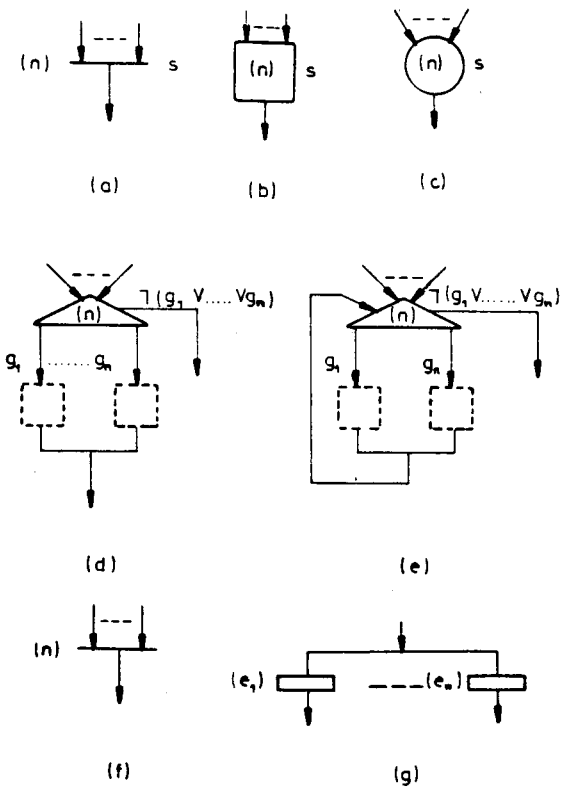


Figura 9: Jerarqía de protocolos

lo. Se define el estado de un proceso integrante del protocolo como la etiqueta de la sentencia en que se encuentra dicho proceso. En la descripción LDP se colocan etiquetas en los puntos que hayan de considerarse estados. La etiqueta corresponde a un estado cuyas acciones asociadas son las indicadas por las sentencias comprendidas entre la etiqueta considerada y la siguiente etiqueta. El estado de un protocolo queda definido por el estado de sus procesos constituyentes.



n: etiqueta de sentencia
s: sentencia
g: guardia
e: etiqueta

Figura 10
Representación gráfica de las sentencias
(a) Sentencia de asignación interna
(b) Sentencia de salida
(c) Sentencia de entrada
(d) Sentencia alternativa
(e) Sentencia repetitiva
(f) Sentencia nula
(g) Sentencia paralela

5.2 EL LDP GRÁFICO

Con el único fin de facilitar la comprensión del funcionamiento de un protocolo y del método de validación que se desarrolla más adelante resulta conveniente la representación gráfica de las descripciones LDP. La figura 10 muestra la representación gráfica de las sentencias del LDP y la figura 11 es la descripción gráfica del protocolo de bit alternante de la figura 8.

5.3 ESTRUCTURA DE UN PROTOCOLO

Para el estudio del funcionamiento lógico de protocolos descritos en LDP conviene prescindir del detalle de la estructura de datos para centrar la atención en los aspectos de control. Se define la estructura gráfica de un protocolo como el grafo que se obtiene de la descripción LDP prescindiendo de todas las variables y rotulando cada arco con la condición que permita la transición entre los estados que une. Existen tres tipos de transiciones; libres (p.e. asignación interna), simultáneas con una transición compatible de otro proceso (p.e., asignación externa) y condicionadas (p.e. sentencia alternativa). Junto a los arcos de las transiciones simultáneas se escribe el nombre del proceso con el que se realiza la transición y la lista de las transiciones entre estados de aquel proceso que pueden realizarse simultáneamente con la transición que se considera. Cuando la ejecución de una transición depende del estado en que se encuentra otro proceso se rotula con los estados del otro proceso en los cuales puede realizarse la transición. La figura 12 es la estructura gráfica del protocolo de bit alternante considerado.

La estructura gráfica de un protocolo conserva las restricciones en la comunicación de la descripción original, siendo mas general que esta al suponerse que cualquier elemento de las guardias puede evaluar a cierto. Si se demuestra alguna propiedad de la estructura, tal como la ausencia de bloqueos, puede afirmarse que el protocolo posee esa propiedad.

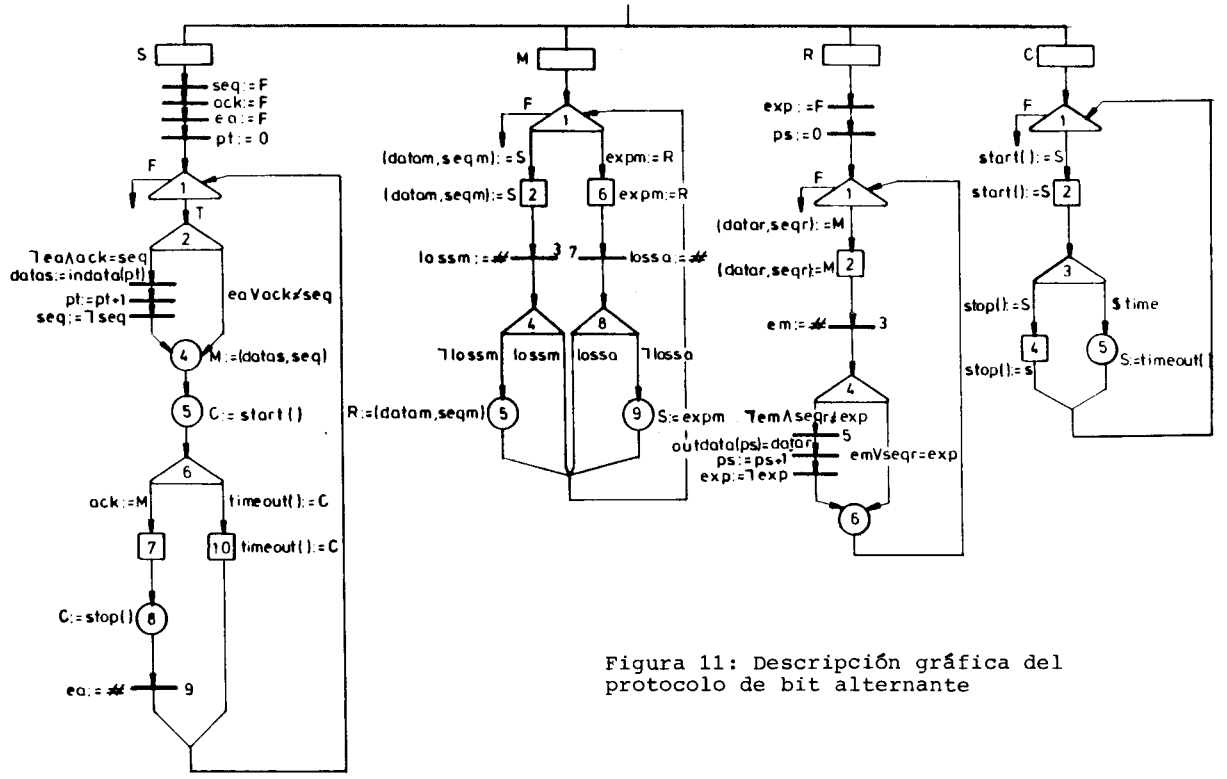


Figura 11: Descripción gráfica del protocolo de bit alternante

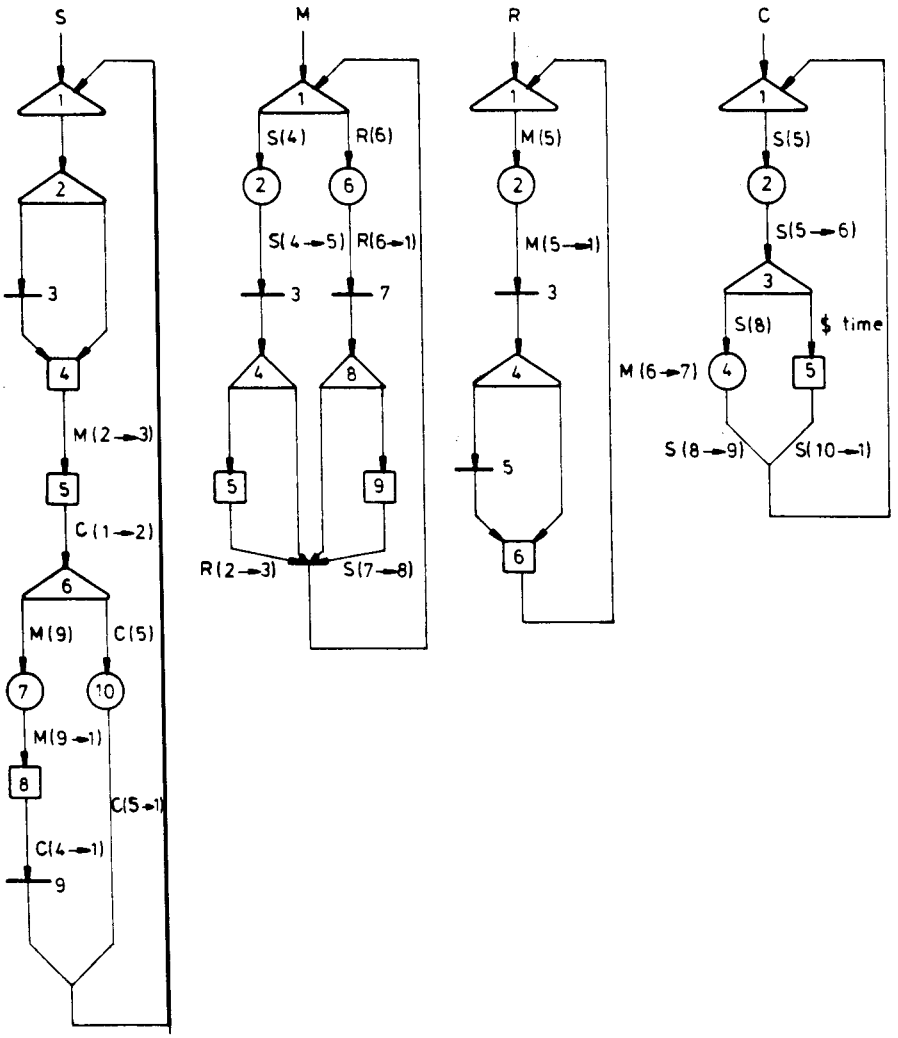


Figura 12: Representación gráfica de la estructura del protocolo de bit alternante.

5.4 EL TIEMPO DE EJECUCIÓN

El estudio de la sincronización de procesos distribuidos involucra ineludiblemente el -- concepto tiempo. Este problema viene siendo abordado en los últimos años en un intento - de formalizar conceptos tales como sistema - distribuido, tiempo y espacio en sistemas -- distribuidos, sucesión de sucesos, etc... -- /23/, /25/, /26/. En las descripciones LDP - se supone una única referencia absoluta de - tiempos lógicos; en la terminología de /26/, un sistema multireferencial perfecto en que la evolución de cada proceso se considera a intervalos de tiempo constantes e iguales para todos. Las transiciones posibles entre estados del sistema quedan perfectamente determinadas conociendo el tiempo lógico de ejecución de las transiciones individuales de cada proceso.

La selección de la unidad de tiempo lógico y la asignación de éste a las distintas transiciones debe tomar en cuenta las características del sistema real y el grado de precisión que se desea obtener en el estudio. La figura 13 muestra la descripción gráfica temporal del protocolo de bit alternante despreciando los tiempos de ejecución de los procesadores y considerando tan solo los tiempos de transmisión, la temporización y los tiempos de comunicación del proceso fuente y consumidor de mensajes. Un valor siempre adecuado para la unidad de tiempo lógico es el máximo común divisor de los tiempos reales de ejecución.

6. VALIDACION DE DESCRIPCIONES LDP

Se exponen en este apartado los algoritmos -

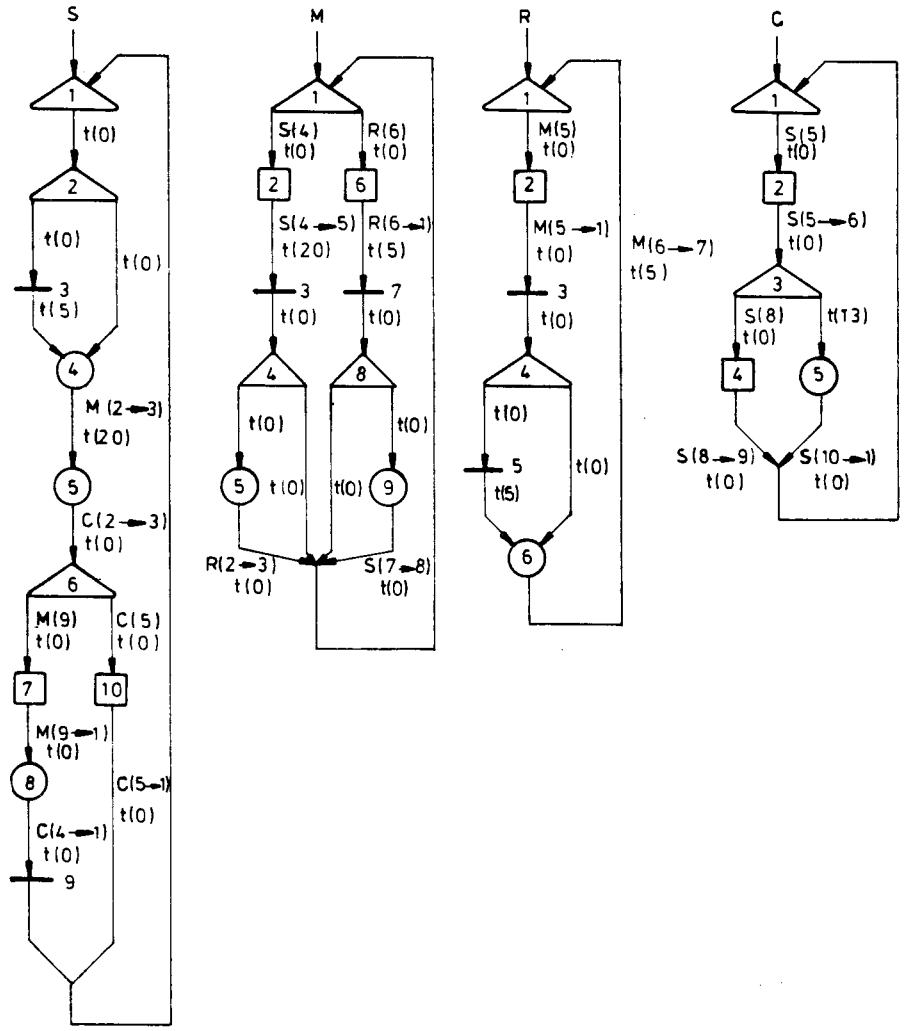


Figura 13: Representación gráfica temporal de la estructura del protocolo de bit alternante.

de validación y de validación temporal con - tiempo de ejecución fijo. Por falta de espacio y por tratarse de una generalización de la validación con tiempo de ejecución fijo - no se considera aquí la validación con tiempo de ejecución variable /10/.

6.1 VALIDACIÓN NO TEMPORAL DE DESCRIPCIONES LDP

La validación se basa en el análisis de la - alcanzabilidad de la estructura del protocolo. Para buscar el conjunto de estados siguientes de uno dado se introduce formalmente el concepto de perturbación como sigue.

Los n procesos integrantes de un protocolo - se numeran en orden creciente. Llamando S_i - al conjunto de estados s_i^j en que puede encontrarse el proceso i, el conjunto de estados $\bar{e}_j$ alcanzables por el protocolo es, en general, $E \subseteq S = S_1 x \ldots x S_n$.

A partir de la estructura gráfica de un protocolo se define para cada proceso i una relación

$R_i \subset SxS$,

denominada relación de perturbación de la -- forma siguiente:

(a) Para todo par de estados s_i^j, $s_i^k \in S_i$ unidos por una transición sin rotular o rotulada con \$

$$[(x \ldots s_i^j \ldots x), (x \ldots s_i^k \ldots x)] \in R_i$$

en donde una x en la posición p, indica cualquier estado $s_p^q \in S_p$

(b) Para todo par de estados s_i^j, $s_i^k \in S_i$ unidos por una transición simultánea con - el proceso 1, y para cada $s_1^m \rightarrow s_1^n$ que rotula el arco

$$[(x \ldots s_i^j \ldots s_1^m \ldots x), (x \ldots s_i^k \ldots s_1^n \ldots x)] \in R_i$$

(c) Para todo par de estados, s_i^j, $s_i^k \in S_i$ unidos por una transición condicionada al - estado de un proceso 1, y para cada (s_1^m) que rotula el arco

$$[(x \ldots x_i^j \ldots s_1^m \ldots x), (x \ldots s_i^k \ldots s_1^m \ldots x)] \in R_i$$

La relación R_i permite conocer los estados a que puede llegar un protocolo desde un estado dado por una transición del proceso i. -- Formalmente se define la perturbación de un estado $\bar{e}_j \in E$ por el proceso i como el conjunto

$$P_i(\bar{e}_j) = \{\bar{e}_k \in E \mid (\bar{e}_j, \bar{e}_k) \in R_i\},$$

conjunto que puede ser vacío.

De un modo análogo, se define el conjunto de estados siguientes de un estado $\bar{e}_j \in E$ como el conjunto

$$P(\bar{e}_j) = \{\bar{e}_k \in E \mid \bar{e}_k \in P_1(\bar{e}_j) \cup \ldots \cup P_n(\bar{e}_j)\}$$

El conjunto de estados siguientes contiene - todos los estados a los que puede llegar el protocolo por la ejecución de una única transición de cualquiera de los procesos que lo forman.

La definición de perturbación y de estado siguiente del apartado anterior permite realizar de un modo simple el análisis de la alcanzabilidad, mediante el cual pueden encontrarse diversos errores de diseño. El análisis se basa en la construcción de un árbol - de ejecución. El nudo inicial o raíz contiene el estado inicial; a partir de él se generan todas las posibles secuencias de estados.

Para la construcción del árbol de ejecución se propone el siguiente algoritmo:

(a) Definir una sucesión de conjuntos A(i) - de estados del sistema. El conjunto A(0) contiene el estado inicial del protocolo. Hacer i=0.

(b) Tomar un elemento $\bar{e} \in A(i)$ cuyo conjunto - de estados siguientes no haya sido aún - determinado e ir a (c). De no existir -- tal elemento mirar si $A(i+1)=\{\emptyset\}$. Si -- A(i+1) es vacío la validación ha concluído, en caso contrario hacer i=i+1 y volver al principio de (b).

(c) Determinar el conjunto de estados siguientes $P(\bar{e})$ y añadir a A(i+1) los elementos de $P(\bar{e})$ que no se encuentren ya en algún A(j) para j=1...i+1. Ir a (b).

Durante la construcción del árbol pueden de-

terminarse ciertos errores de diseño, principalmente:

(a) Bloqueos

Si el conjunto de estados siguientes de uno dado, $P(\bar{e})$, es vacío (apartado (c) del algoritmo), puede afirmarse que $\bar{e}$ es un estado terminal o un bloqueo potencial. Del árbol de ejecución se determina el camino seguido por el protocolo para llegar al estado $\bar{e}$. Del examen de la descripción LDP del protocolo se desprende si el protocolo puede efectivamente recorrer ese camino o no (recuérdese que la estructura del protocolo pierde el sentido de la lógica interna de la descripción original).

(b) Bloqueos temporales

En el apartado (c) del algoritmo se determina si los estados sucesores de uno dado, $P(\bar{e})$, han sido previamente generados. Los estados sucesores que ya existan en el árbol no se consideran más. Si un estado que ya existe es antecesor de un estado generado, se ha detectado un bucle. De la descripción LDP se determina fácilmente si esta situación es o no correcta.

(c) Comportamiento cíclico

Si el protocolo debe retornar al estado inicial tras un número finito de transiciones, en el análisis de la alcanzabilidad debe aparecer el estado inicial en algún $P(\bar{e})$. Si la validación acaba y no se ha dado esta circunstancia el comportamiento del protocolo es erróneo.

La figura 14 ilustra la validación del protocolo de bit alternante cuya estructura representa la figura 12. Para simplificar se ha supuesto que la guardia $time no se hace cierta mientras sea posible la ejecución de cualquier otra transición. Los asteriscos indican que el estado generado ya existe en el árbol. Del análisis de la alcanzabilidad del protocolo de bit alternante se desprende que su comportamiento es cíclico y carece de bloqueos.

6.2 VALIDACIÓN CON TIEMPO DE EJECUCIÓN FIJO

Considerando el tiempo de ejecución de las transiciones la determinación de los sucesores de un estado requiere, además del conocimiento de las características estáticas del protocolo -grafo temporal-, la consideración del cambio de ejecución seguido para llegar al estado.

En la validación temporal puede producirse mas de una transición en un salto discreto de tiempo lógico. Por ello se relajan las definiciones de perturbación y relación de perturbación como se indica en el párrafo siguiente, donde el símbolo - denota la indeterminación del estado de un proceso. El conjunto de estados del sistema en el que el estado de algún proceso esté indeterminado se representa por $\underline{E}$.

A partir de la descripción gráfica temporal de un protocolo se define para cada proceso una relación T_i denominada de perturbación parcial, de la forma siguiente:

(a) Para cada par de estados s_i^j, $s_i^k \in S_i$ unidos por una transición libre o rotulada con \$

$$[(x_1 \ldots x_i^j \ldots x_n), (- \ldots - s_i^k - \ldots -)] \in T_i$$

donde $x_p = s_p^1 \in S_p$ para algún 1.

(b) Para cada par de estados s_i^j, $s_i^k \in S_i$ unidos por una transición simultánea, y por cada $s_1^m \rightarrow s_1^n \in S_1$ que rotula la transición

$$(x_1 \ldots s_i^j \ldots s_1^m \ldots x_n), (- \ldots - s_i^k - \ldots - s_1^n - \ldots -) \in T_i$$

(c) Para cada par de estados s_i^j, $s_i^k \in S_i$ unidos por una transición condicional, y por cada $s_1^m \in S_1$ que la rotula

$$(x_1 \ldots s_i^j \ldots s_1^m \ldots x_n), (- \ldots - s_i^k - \ldots - s_1^m - \ldots -) \in T_i$$

Dado un estado $\bar{e} \in \underline{E}$ del sistema se define para cada proceso la perturbación parcial del estado $\bar{e}$, como el conjunto

$$U_i(\bar{e}) = \{(\bar{f}, t_i^{jk}) \in \underline{E} \times N \mid (\bar{e}, \bar{f}) \in T_i\}$$

en donde N es el conjunto de números naturales, t_i^{jk} es el tiempo que rotula la transición de s_i^j a s_i^k, elementos i-ésimos de $\bar{e}$ y $\bar{f}$

respectivamente. La perturbación parcial de un estado $\bar{e}$ por un proceso i no es más que - la relación de los estados que pueden alcanzar el proceso i y los que interaccionan con él y los tiempos de ejecución de las transiciones.

La validación requiere la construcción de un árbol de ejecución temporal. Los nudos de árbol constan de dos elementos:

(a) El estado $\bar{e}$ del sistema.

(b) Los conjuntos de transiciones parciales posibles de cada proceso y el tiempo que necesitan para realizarse. Estos conjuntos se denominarán perturbaciones parciales dinámicas, y se denotan por V_i.

Obsérvese que las V_i son parte del nudo y no dependen exclusivamente del estado del sistema, $\bar{e}$. Como se verá más adelante su contenido depende del camino seguido para llegar al nudo.

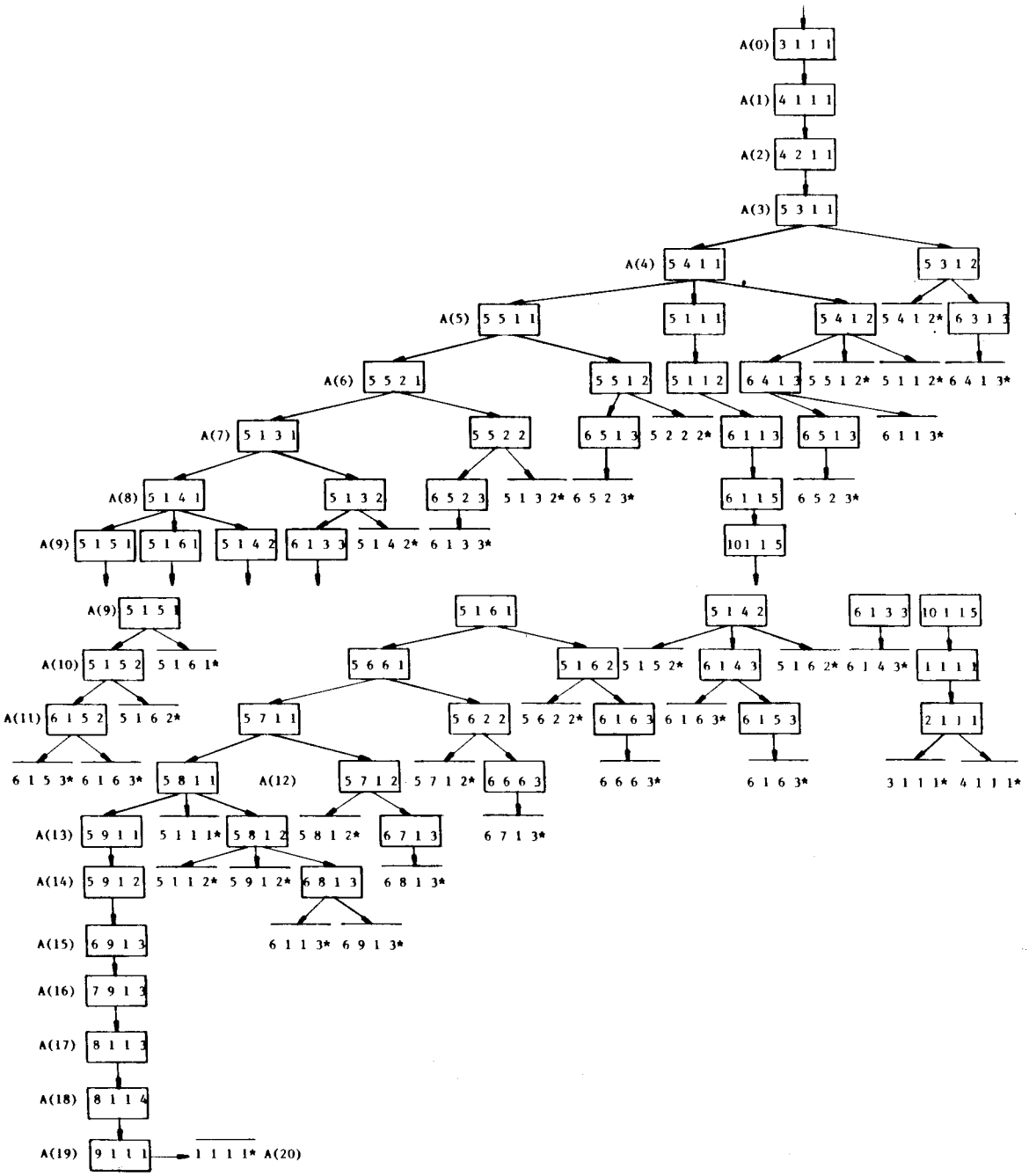


Figura 14: Arbol de ejecución del protocolo de la figura 13.

A continuación se presenta el algoritmo utilizado para la generación del árbol de ejecución temporal. El nudo inicial, o raíz, está formado por el estado inicial $\bar{e}_0 \in E$ y las V_i son $V_i = U_i(\bar{e}_0)$ (perturbaciones de $\bar{e}_0$). Para obtener los nudos sucesores de uno dado n, se siguen los pasos indicados a continuación.

(a) Buscar en los V_i^n la transición que requiere un tiempo menor para ejecutarse. Se denota t_{min} al tiempo de ejecución de esta transición.

(b) Buscar los estados sucesores de $\bar{e}$ al cabo del tiempo t_{min}. Para ello

(1) Formar los conjuntos

$$W_i = \{\bar{f} \mid (\bar{f}, t_{min}) \in V_i^n\}.$$

Cada W_i contiene las transiciones parciales del estado $\bar{e}$ que puede ejecutar el proceso i en un tiempo t_{min}.

(2) Tomando un único elemento de cada W_i formar las transiciones compatibles de los procesos constituyentes que conducen a los nuevos estados $\bar{g}$, sucesores del $\bar{e}$.

(c) Cada estado sucesor $\bar{g}$ constituye un nuevo nudo, cuyas perturbaciones parciales dinámicas V_i^m se forman a partir de las V_i^n del nudo origen en tres pasos

(1) Para todo i tal que $e_i \neq g_i$ hacer $V_i^m = \{\emptyset\}$.

(2) Para todo i tal que $e_i = g_i$ hacer $V_i^m = V_i^n$ y decrementar todos los tiempos de V_i^m en t_{min}.

(3) Para todo i, añadir a V_i^m los elementos de $U_i(\bar{g})$ que no correspondan a estados siguientes del sistema ya reflejados en algún elemento de V_i^m.

A pesar de la aparente complejidad del algoritmo, su aplicación manual es sencilla. La idea es que de las transiciones que pueden realizarse en un momento dado se seleccionan para su ejecución las que requieren un menor tiempo de ejecución. Los procesos que no han cambiado de estado y que podían ejecutar una transición continúan en esta situación, pero el tiempo que debe transcurrir para que pueda ejecutarse la transición ha disminuido en t_{min}. Por otra parte, en el estado alcanzado pueden admitirse nuevas transiciones, que deben incluirse en la caracterización del nudo.

La figura 15 muestra una simplificación del

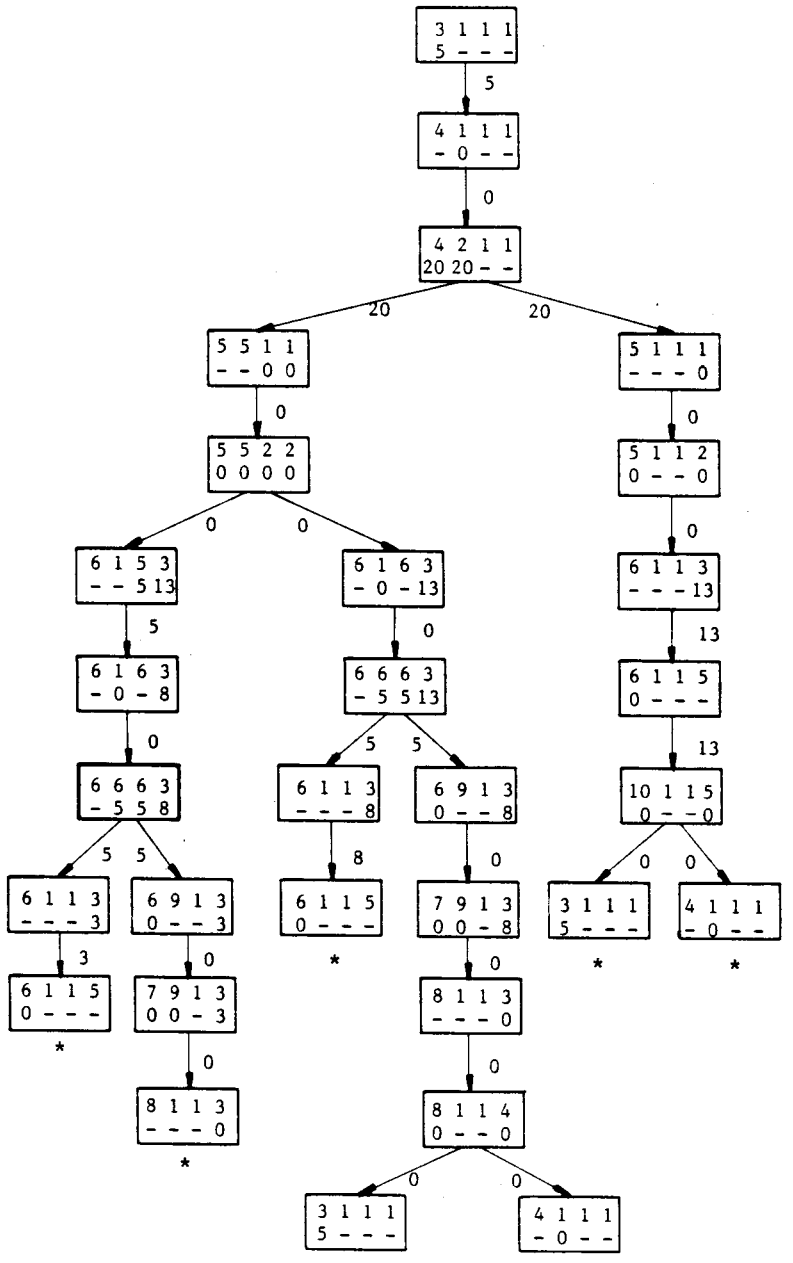


Fig. 15. Arbol de ejecución temporal del protocolo de la figura 13.

árbol de ejecución temporal del protocolo de la figura 13. La parte superior de los rectángulos indica el estado del sistema correspondiente al nudo; en la parte inferior se representan los tiempos más pequeños de los V_i. Los V_i vacíos se representan por -, indicando que el proceso correspondiente no puede modificar su estado (puede pensarse que - significa tiempo de ejecución infinito).

El algoritmo de validación adquiere un aspecto similar al de la validación no temporal. Se desglosa en los siguientes pasos:

(a) Definir una sucesión de conjuntos A(i) - de nudos del árbol. El conjunto A(0) contiene el nudo inicial, $\bar{e}_0$ y $V_i=U_i(\bar{e}_0)$. - Hacer i=0.

(b) Tomar un nudo de A(i) cuyo conjunto de - nudos siguientes no haya sido aún determinado e ir a (c). De no existir tal nudo mirar si $A(i+1)=\{\emptyset\}$. En este caso la validación ha concluído, en caso contrario hacer i:=i+1 y volver al principio - de (b).

(c) Determinar los nudos sucesores del nudo considerado y añadir a A(i+1) los que no se encuentran ya en algún A(j) para -- j=1...i+1. Ir a (b). Dos nudos n y m son iguales cuando los estados son iguales y $V_i^n=V_i^m$ para todo i.

La figura 15 corresponde a la validación temporal del protocolo de bit alternante que se viene considerando. Es interesante hacer notar la notable reducción en el número de estados (nudos) respecto a la figura 14 que representa la validación no temporal del mismo protocolo. Asimismo, es importante remarcar la simplicidad del proceso de validación a - partir de la descripción gráfica temporal -- LDP de la figura 13. Si en ésta se modifica la temporización, dándole un valor de 9 unidades de tiempo, t(9), se obtiene la (no) validación de la figura 16. En ella se observa claramente que el sistema puede quedar bloqueado en el estado (4911). Este error proviene de la selección inadecuada de la temporización.

7. OBSERVACIONES Y CONCLUSIONES

Con el LDP puede especificarse el funcionamiento de los procesos que integran un protocolo y su entorno. La ejecución concurrente de los procesos y el no determinismo se expresan mediante la sentencia paralela y la - alternativa. El lenguaje permite describir - la estructura de datos de los procesos así - como su lógica interna, medios de comunicación, estructuras jerárquicas, temporizaciones, etc. La comunicación entre procesos se realiza a través de mensajes mediante la ejecución simultánea de una sentencia de salida en un proceso y una de entrada en otro. El - LDP permite especificar sencillamente otras formas de comunicación como son la comunicación a través de memoria común, monitores, - etc. El aspecto del LDP es el de un lenguaje de alto nivel por lo que las conocidas técnicas de programación estructurada y modular - pueden utilizarse en la descripción de protocolos. En particular, el LDP permite realizar corutinas, subrutinas, procedimientos, - etc.

En la validación no temporal se generan todas las posibles secuencias de transiciones sin tener en cuenta la velocidad de ejecución de los procesos por los que en el árbol de - ejecuciones aparecen más caminos de los que el protocolo real puede seguir, al considerarse todas las formas en que los procesos - pueden entrelazar su ejecución tomando como cero el tiempo de ejecución de las transiciones.

La validación temporal resulta mas simple -- que la no temporal. El árbol de ejecución -- contiene todos los caminos de ejecución que el protocolo puede seguir en su funcionamiento, sin considerar las limitaciones impuestas por la estructura de datos de los procesos, pero tomando en cuenta su tiempo de ejecución.

La aplicación manual de uno u otro algoritmo de validación es extraordinariamente sencilla. Sin embargo, para protocolos de cierta complejidad la construcción manual del árbol de ejecuciones se hace inabordable. Es interesante considerar la posibilidad de automatizar el proceso de validación mediante un programa que partiendo de la descripción LDP de un protocolo genere su árbol de ejecución. -

La limitación fundamental de este tipo de -- programas se encuentra en el número de estados diferentes que pueden aparecer en el árbol de ejecución. Por un lado, el almacena-- miento en memoria de los estados puede encon trar limitaciones de capacidad, por otro, al generar un nudo sucesor es necesario comprobar si se trata de un nuevo nudo o si es un nudo ya existente. Esta comprobación resulta tanto mas lenta cuanto mayor sea el número - de nudos del árbol. Este número depende fundamentalmente del grado de acoplamiento (interacciones) entre los distintos procesos -- que integran el protocolo.

De la validación de la estructura gráfica de un protocolo puede concluirse la validación de la descripción LDP. En efecto, si el sistema se comporta correctamente cuando en las sentencias alternativas se elige aleatoria-- mente la lista de sentencias que ha de ejecutarse, también lo hará si en la selección se

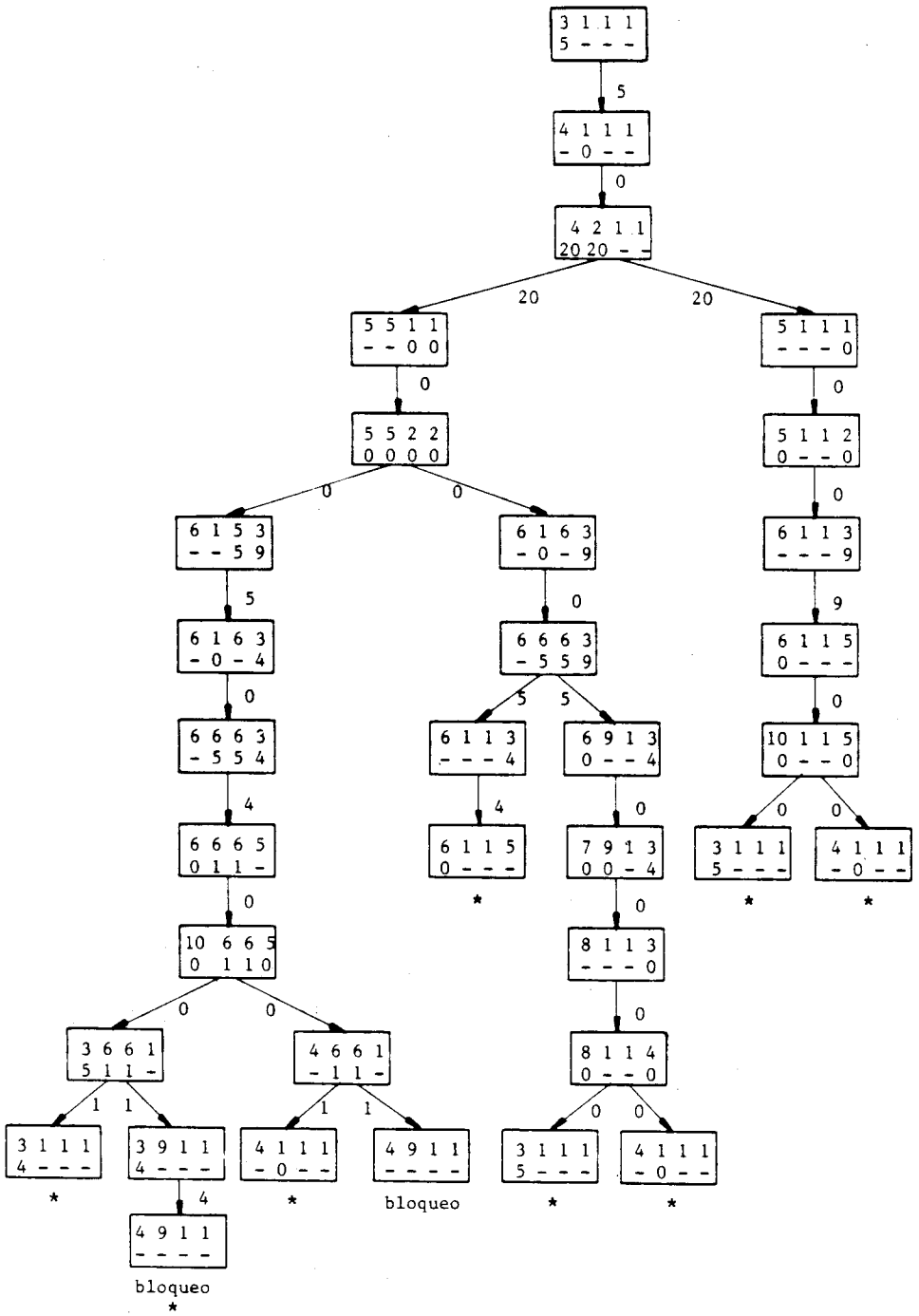


Figura 16 : Arbol de ejecución con temporización t(9).

considera el valor de las guardias. Contrariamente, los errores que se encuentran en la validación de la estructura no siempre serán errores de diseño del protocolo. De la descripción LDP y del camino de ejecución que conduce al error se determina de forma inmediata si el protocolo puede realmente llegar en su funcionamiento al estado erróneo.

En este artículo se han considerado tan solo dos aspectos de una metodología formal para el diseño correcto de protocolos, los referentes a la especificación y la validación. La verificación de protocolos descritos en LDP puede encontrarse en /10/, /13/, y la realización práctica de descripciones LDP verificadas en /12/.

8. AGRADECIMIENTOS

Gran parte del trabajo en que se basa este artículo fué realizado en el Instituto de Cibernética UPB/CSIC con el soporte económico de una Beca para Estudios Científicos y Técnicos de la Fundación Juan March. La Càtedra de Mètodes Informàtics de la E.T.S.E.I.B. me brindó el tiempo necesario para la confección de estas páginas.

9. REFERENCIAS

/1/ BARTLETT, K.A. et al.: "A Note on Reliable Full-Duplex Transmission over Half-Duplex Links". Comm. ACM, v.12, n.5, May 1969, pp. 260-261.

/2/ BOCHMANN, G.V., CHUNG, R.J.: "A Formalized Specification of HDLC Classes of Procedures". University of Montreal, Tr. n. 265, 1977.

/3/ CAMPBELL, R.H., HABERMANN, A.N.: "Specification of Process Synchronization by Path Expressions". Proc. of the International Symposium on Operating Systems. Rocquencourt, 1974.

/4/ CCITT Libro Naranja, vol. VII-2, Redes Públicas de Datos, Ginebra, 1977.

/5/ DAHL, O.J. et al.: "SIMULA67, Common Base Language". Norwegian Computing Center. Forksningveien, Oslo, 1967.

/6/ DAVIS, D.W., BARBER, D.L.A.: "Communication Networks for Computers". Wiley & Sons, Inc. London. 1973.

/7/ DIJKSTRA, E.W.: "Guarded Command, Nondeterminacy and Formal Derivation of Programs". Comm. ACM, v.18, n.8, Aug. 1975, pp. 453-457.

/8/ DOLL, D.R.: "Data Communications". John Wiley & Sons, 1978.

/9/ DONNAN, R.A., KERSEY, J.R.: "SDLC: A Perspective". IBM Systems Journal, v.15, n.1, 1976, pp. 140-162.

/10/ GARCIA HOFFMANN, M.: "Aportación al Estudio de la Descripción, Validación y Verificación de Protocolos de Comunicación". Tesis Doctoral. E.T.S.I.I. de Barcelona, 1979.

/11/ GARCIA HOFFMANN, M.: "Técnicas para la Descripción Formal y Verificación de Protocolos". Qüestiió, v.3, n.4, 1979, pp. 219-229.

/12/ GARCIA HOFFMANN, M.: "Hardware Implementation of Communication Protocols: A Formal Approach". Proc. of the Seventh International Symposium on Computer Architecture, La Baule, May 1980.

/13/ GARCIA HOFFMANN, M.: "Verificación de Programas Concurrentes. Un Caso de Aplicación". Pendiente de publicación.

/14/ GREIF, I.: "A Language for Formal Problem Specification". Comm. ACM, v.20, n.12, Dec. 1977, pp. 931-935.

/15/ HABERMANN, A.N.: "Synchronization of Communicating Processes". Comm. ACM, v.15, n.3, March 1972, pp. 171-176.

/16/ HABERMANN, A.N.: "Path Expressions". TR Computer Science Dep. Carnegie Mellon Univ., 1975.

/17/ HANSEN, P.B.: "The Programming Language Concurrent Pascal". IEEE Trans. on Soft. Eng., v.SE-1, n.2, June 1975, pp. 199-207.

/18/ HANSEN, P.B.: "The Architecture of Concurrent Programs". Prentice-Hall, 1977.

/19/ HANSEN, P.B.: "Distributed Processes: A Concurrent Programming Concept". Comm. ACM, v.21, n.11, Nov. 1978, pp. 934-941.

/20/ HOARE, C.A.R.: "Communicating Sequential Processes". Comm. ACM, v.21, n.8, Aug. 1978, pp. 666-677.

/21/ ISO: "High Level Data Link Control Procedures". ISO/TC/SGG, N-1005, 1975.

/22/ KLEINROCK, L.: "Queueing Systems: Compu ter Application". John Wiley & Sons, - 1976.

/23/ LAMPORT, L.: "Time, Clock and the Order ing of Events in a Distributed System". Comm. ACM, v.21, n.7, July 1978, pp. - 558-565.

/24/ LAYTON, C., "Computer Network Protocols. Their Place in European Strategy for -- Computer Communications". Proc. of the Computer Network Protocols Symposium. - Liege 1978.

/25/ LE LANN, G.: "Note sur la Synchronisa-- tion et l'Observation des Evenements -- dans les Systêmes de Traitament Repar-- tis". IRIA, Project Sirius, Bigre 9, -- 1978.

/26/ LE LANN, G.: "Distributed Systems. Towards a Formal Approach". Information - Processing 77, IFIP, North-Holland Pub. Co., 1977.

/27/ MARTIN, J.: "Systems Analysis for Data Transmission". Prentice Hall, 1972.

/28/ SCHWARZ, M.: "Computer Communication -- Network Design and Analysis". Prentice Hall, 1977.

/29/ SUNSHINE, C.: "Interprocess Communica-- tion Protocols for Computer Networks". Tech. Rep. #105 Digital Systems Lab., - Standford Univ., 1975.