UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONAL Y DE POSGRADO DEL:

COLEGIO DE CIENCIAS Y HUMANIDADES

INSTITUTO DE INVESTIGACIONES EN MATEMATICAS APLICADAS

Y EN SISTEMAS

ESPECIFICACION FORMAL E IMPLANTACION DE UN PROTOCOLO POR DIFUSION TOLERANTE A FALLAS

T E S I S

QUE PARA OBTENER EL GRADO DE

M A E S T R O E N C I E N C I A S

D E L A C O M P U T A C I O N

P R E S E N T A :

J. REFUGIO VALLEJO GUTIERREZ

MEXICO, D. F. MAYO 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Prefacio

Con la aparición de las redes de computadoras, el área de computación que más importancia ha adquirido son los Sistemas Distribuídos. Lo que caracteriza a un sistema distribuído no es sólo la distribución de sus partes, sino también que sus partes se puedan comunicar. El soporte de comunicación en un Sistema Distribuído permite la emisión de mensajes entre máquinas y el sistema operativo extiende esta facilidad permitiendo la comunicación entre procesos de diferentes nodos. Para la comunicación entre procesos, estos sistemas ofrecen facilidades para crear circuitos virtuales, los cuales tienen asociados una serie de protocolos que ofrecen cierto grado de confiabilidad en la comunicación. Sin embargo, desde el punto de vista de la programación distribuída, estas facilidades no dejan de ser de bajo nivel; esto ha llevado a la búsqueda de abstracciones más apropiadas y de más alto nivel que permitan la comunicación entre procesos. Algunos investigadores sugieren una comunicación a través de un tipo de memoria global compartida ocultándole al programador la parte de la distribución. Esta abstracción tiene la ventaja de que cualquier programa escrito en un sistema no distribuído se puede trasladar a uno distribuído de manera transparente. Una abstracción de alto nivel muy usada para comunicación entre procesos son los llamados "Remote Procedure Call(RPC)". Un proceso se comunica con otro utilizando una interfase que es similar a una llamada a un procedimiento. La ventaja de esta abstracción es que esto simplifica la programación distribuída porque permite la llamada a un proceso remoto como si se efectuara entre procesos locales. Una de las desventajas es que está pensada para modelos tipo "cliente-servidor", por lo tanto los "RPC" no son una abstracción conveniente cuando un programa distribuído se compone de un cierto número de procesos que tienen un alto grado de interdependencia entre sí y la comunicación entre ellos refleja su interdependencia. En este tipo de programas la comunicación que se requiere es una comunicación de un proceso a un conjunto de procesos, este servicio lo logramos con los protocolos **protocolos por difusión.**

La difusión en su expresión más simple hace una copia de un mensaje en cada proceso destino. Lo que hace interesante la difusión es que ésta pueda manejar la posibilidad de que algún proceso que forme parte de la difusión falle o que simplemente haya un error en la mitad de la difusión, por ejemplo, una falla de un emisor puede causar que un mensaje no llegue a todos sus destinos. Lo que es realmente útil a un programador es que la difusión tenga un comportamiento bien definido aún cuando existan fallas. La difusión que ofrece esta garantía es lo que llamamos **protocolos por difusión confiables o tolerantes a fallas.**

En los Sistemas Distribuídos frecuentemente se necesita resolver problemas complejos, por ejemplo, consistencia en bases de datos distribuídas, solución de problemas paralelos, entre otros. Para

lograr esta consistencia necesitamos que todos los mensajes enviados por el conjunto de máquinas que participan en la difusión tengan un orden total. Para lograr este orden, se han considerado dos enfoques: un enfoque centralizado en donde todos los mensajes pasan a través de un nodo, el cual asigna un orden a los mensajes y este orden es adoptado por los demás nodos, el otro enfoque es un enfoque distribuído en donde cada nodo localmente decide el orden correcto de acuerdo a la información que ha recibido; en este caso difícilmente se logra un orden total. Dependiendo del enfoque que se utilice tendremos ya sea un **protocolo por difusión de control centralizado tolerante a fallas** o un **protocolo por difusión de control distribuído tolerante a fallas** respectivamente.

En esta tesis lo que vamos a especificar e implantar es un **protocolo por difusión de control centralizado tolerante a fallas.**

Para ubicarnos dentro del contexto, en el capítulo I vamos a hablar en general de las redes de computadoras que va a ser la arquitectura en la cual se implementa el protocolo, haciendo incapié en las diferentes topologías en donde la difusión puede o no ser confiable; hablaremos también de los Sistemas Distribuídos (en los cuales son de gran importancia los protocolos por difusión) y de los protocolos de comunicación en donde describiremos claramente lo que se debe tomar en cuenta al especificar un protocolo. Mencionaremos las fallas a las cuales nos podemos enfrentar en el ambiente de las comunicaciones mencionando la influencia que pueden tener en los protocolos por difusión.

Como ya vimos, uno de los objetivos en los protocolos por difusión es el envío de una serie de mensajes a un conjunto de máquinas y garantizar el mismo orden de los mensajes en todas estas. Para garantizar este orden, se utiliza un control centralizado o un control distribuído lo cual da origen a los protocolos por difusión de control centralizado y protocolos por difusión de control distribuído que describiremos ampliamente capítulo 2.

Sabemos que para la especificación de un sistema o modelo en general podemos utilizar herramientas formales o informales. Entre las herramientas formales encontramos a los diagramas de estados [PER90], redes de Petri [Pet77], CCS [Mil80] y otros. Entre las herramientas informales encontramos al pseudocódigo, diagramas de flujo, etc. Cada técnica tiene sus ventajas y desventajas pero para poder asegurar que el modelo especificado no tiene inconsistencias es necesario utilizar técnicas de especificación formal; en particular las redes de Petri son a la vez una herramienta gráfica y una herramienta formal que permite especificar de manera muy natural sistemas que poseen concurrencia, no determinismo y sincronización. En este caso no se utilizan las máquinas de estados por su limitación en cuanto a poder de modelado sobre todo para sistemas concurrentes; en particular veremos que éstas son un caso particular de las redes de Petri. Hay otras técnicas de más alto nivel basadas en herramientas formales entre éstas encontramos a SDL [BOC90] basada en máquinas de estados finitos extendidas, a LOTOS [TB88] que esta basado en máquinas de estados finitos extendidas y el Cálculo de Sistemas Comunicantes de Milner [Mil80] y ESTELLE [BD87] que también se basa en máquinas de estados finitos extendidas.

Para especificar un sistema se van a utilizar las redes de Petri por su formalidad y potencial gráfico aunque hay ciertas restricciones en la validación de las especificaciones. En el capítulo 3 describimos esta herramienta de modelado, haciendo notar que existen diferentes extensiones que permiten especificar con más libertad y claridad; entre las cuales encontramos a las redes de

Petri con prioridades [BK92] y las redes de Petri generalizadas [AMC91] que al mismo tiempo nos permiten encontrar algunas características del modelo que no es posible encontrar utilizando el modelo original de las redes de Petri. En este capítulo también haremos incapié en los modelos reducidos en los cuales es posible efectuar cualquier validación de una especificación.

En el capítulo 4 describimos algunas herramientas para validar modelos en general, pero solamente vamos a profundizar en las técnicas para validar modelos especificados con redes de Petri.

En el capítulo 5 describiremos las herramientas de programación que se usaran para la implementación del protocolo; estas herramientas son sockets, pipes y creación de procesos.

Hasta el capítulo 5 hemos descrito la arquitectura(redes) sobre la cual se va a implantar el protocolo, las reglas que debemos seguir para poder implantar este protocolo(protocolos) y hemos visto la importancia que pueden tener los protocolos por difusión en el ámbito de los Sistemas Distribuídos, además se han descrito los diferentes enfoques de implantación (protocolos de control centralizado y control distribuído) y también se ha descrito una herramienta formal para especificar sistemas concurrentes en general(Redes de Petri) y al mismo tiempo hemos visto la forma de validar los sistemas especificados con esta herramienta descrita(Herramientas de validación) y por último también describimos las herramientas usadas para la implantación del protocolo. Con todo esto tenemos elementos suficientes para **especificar, validar e implantar un protocolo por difusión de control centralizado tolerante a fallas**, el cual va a ser descrito en el capítulo 6.

Agradecimientos:

- Agradezco a mis papás y a mis hermanos por haber depositado su confianza en mí y apoyado incondicionalmente.
- A la familia Lozano por su hospitalidad y apoyo que siempre me ha brindado.
- A la familia Delgado en la que siempre he confiado.
- Al Dr. Víctor Germán Sánchez por haberme dirigido esta tesis, al Dr. Sergio Rajsbaum por sus valiosas sugerencias sobre el tema desarrollado y a la Dra. Hanna Oktaba por su paciencia e invaluable apoyo al programa de Maestría en Ciencias de la Computación.
- Agradezco también a la M. en C. Gloria Quintanilla y al Dr. Felipe Bracho por haber aceptado ser mis sinodales y por revisar este trabajo.
- A todos mis amigos y en particular a los del grupo de Sistemas Distribuídos quienes han contribuído a enriquecer mí conocimiento en el área.

José Refugio Vallejo Gutiérrez.

México D.F., Mayo 1993.

Contenido

Lista de Figuras

Capítulo 1

Redes de computadoras, protocolos y sistemas distribuídos

1.1 Redes de computadoras

En el área de computación y comunicaciones actualmente existe una serie de términos que no siempre son claros para la gente especialista en estas áreas, mucho menos para el público en general. En este capítulo se describirán una serie de conceptos básicos que nos permitirá ubicarnos dentro del contexto de las redes de computadoras, protocolos de comunicación y sistemas distribuídos.

1.1.1 Introducción.

Vamos a partir del hecho de que existe un conjunto de elementos individuales y cada elemento tiene ciertos atributos que le permiten autocontrolarse y autocomunicarse entre cada parte que lo conforma. Si los elementos de los que hablamos son personas, estas tienen un procesador central(cerebro) del cual se controla el resto de las partes del cuerpo, a cada elemento le surge la necesidad de comunicación ya sea con algún elemento del mismo grupo o con un elemento de otro grupo, de esta necesidad nacen los medios de comunicación.

Esto se observa en la sociedad en donde los individuos inician el habla con un vocabulario muy reducido y evolucionan hasta enriqueserlo. Por razones sociales o políticas se forman grupos(ciudades) ajenos, y la comunicación solo se da entre grupos; a partir de esto surge la necesidad de comunicación entre ellos lo cual da origen al surgimiento de las vías y medios de comunicación. La comunicación surge del hecho de que ambas partes tienen necesidad de intercambio ya sea de productos comerciales, de información o en general de recursos. De la misma forma que evolucionan tanto la lengua y los medios de comunicación evoluciona la tecnología para el desarrollo de nuevos medios de comunicación que trae como consecuencia la aparición del teléfono generando un medio de comunicación eficiente entre individuos y al mismo tiempo dá origen a la creación de redes telefónicas, resolviendo el problema de transferencia de información entre individuos.

Hasta aquí lo que podemos concluir es que a partir de elementos individuales con necesidades de comunicación y la evolución de la tecnología, nos lleva a establecer redes telefónicas para la comunicación entre individuos.

Por otro lado tenemos la aparición de elementos capaces de procesar grandes cantidades de información a altas velocidades, estos elementos a los que nos referimos son las computadoras que también han evolucionado desde un ábaco hasta lo que son ahora las supercomputadoras. Aquí también ha evolucionado el proceso de comunicación entre la computadora y el humano; inicialmente la comunicación entre estos era una tarea engorrosa en donde la comunicación se llevaba a cabo mediante el uso de tarjetas pero ahora a evolucionado a tal grado que la computadora es capaz de entender lo que habla el individuo sin necesidad de escribir.

Una necesidad natural es la comunicación entre computadoras, lo cual da lugar al nacimiento de los medios de comunicación tales como: cable coaxial, par trenzado, fibra óptica, microondas y otros. Todo esto da lugar a las redes de computadoras de lo cual hablaremos en este capítulo.

Una vez que hemos logrado establecer la comunicación entre computadoras, las exigencias naturales es lograr la comunicación de manera eficiente. Recordemos también que hasta ahora, a lo que la mayoría del público está acostumbrado es a una computadora en la cual se almacena toda la información y se tienen conectados una serie de dispositivos de entrada y salida, tales como terminales, impresoras, ploters y otros; todos estos obtienen la información del procesador central al cual están conectados.

Sabemos que individualmente se tiene cierta capacidad pero en general si unimos esfuerzos, se logra tener mucha mayor capacidad que como elemento individual, veamos entonces lo que podemos lograr si unimos computadoras; es decir, si formamos redes de computadoras.

1.1.2 Objetivos de las redes de computadoras.

Actualmente existen organizaciones que tienen computadoras en lugares diferentes y distantes que tienen una serie de información que cuando es necesario accesarla en un lugar en donde no está presente se necesita trasladarla físicamente. Con las redes de computadoras lo que se pretende es poder compartir recursos e información entre las computadoras conectadas en red. Otro objetivo importante es la confiabilidad de la información, esto se logra efectuando un duplicado de información de tal forma que si alguna computadora falla, se puede extraer la información de la otra en la cual se encontraba duplicada; este servicio es típico en las bases de datos porque en general es información muy valiosa. Con la conexión de la red otro de los objetivos que se logran es el ahorro de dinero, las computadoras pequeñas son mucho mas baratas que un "Mainframe". Uno de los puntos importantes es que se deja abierta la posibilidad de incrementar la eficiencia conforme se agregan mas procesadores, al contrario de los "mainframes" que cuando se satura el sistema disminuye la eficiencia y en tal caso es necesario reemplazarlo por uno mas eficiente en cuyo caso es demasiado caro. Otra de las ventajas que se logran al establecer la red es que nos prové de un medio de cooperación entre personas o grupos de personas separadas, por ejemplo, cuando algún autor efectúa un cambio a cierta información que se encuentra en línea, otros miembros del grupo que cooperan se enteran inmediatamente del cambio.

Dentro de las facilidades o aplicaciones que se logran al tener una red, podemos mencionar las siguientes:

- Acceso a información remota como programas, bases de datos, etc.
- Las empresas que desarrollan productos pueden permitir a sus clientes la ejecución remota del producto mediante una renta, es decir, alquilar el uso de algún software sin que se tenga que vender.
- También podemos accesar bibliotecas científicas distribuídas en las universidades conectadas en red.
- Actualmente entre las computadoras conectadas en la red ya se está logrando la trasmisión de voz, imágenes y videos, próximamente se estará logrando trasmitir programas de televisión.

La entrada de las redes tendrán un impacto social del estilo de la revolución industrial.

1.1.3 Estructura de una red.

Como red vamos a entender a un conjunto de máquinas conectas entre sí mediante líneas de comunicación. Desde el punto de vista de funcionalidad existen 2 tipos de redes:

1. Redes de área local que en general abarca un edificio.
2. Redes de área extensa que constan de 2 elementos:
   (a) Líneas de trasmisión (que incluye circuitos, canales, etc.).
   (b) Elementos de "switcheo" que son computadoras especializadas para conectar más de una línea de trasmisión, algunas veces son conocidos como interfase para el proceso de mensajes.

En las redes existen 2 tipos de comunicación:

1. Comunicación punto a punto (uno a uno).
2. Comunicación por difusión (uno a muchas).

En la comunicación punto a punto cuando se envía algún mensaje o paquete a una computadora pero pasa por una computadora intermedia, esta lo recibe y lo almacena hasta que la línea hacia la computadora destino se desocupa; en este caso la difusión no es natural, para lograrla necesitamos desarrollar protocolos que implementen este mecanismo. El algoritmo para estas topologías se vuelve más complejo que en el caso de topologías por difusión. En la comunicación por difusión un canal es compartido por todas las máquinas de la red y cuando un paquete es enviado al canal, es recibido por todas las máquinas conectadas a la red. En estos casos si se desea enviar un paquete a cierta máquina se especifica la dirección dentro del paquete y aunque todas lo ven solo lo recibe la máquina a la que fue enviada. Aquí la difusión es natural aunque en general las primitivas que

se ofrecen son para la comunicación punto a punto independientemente de la arquitectura. De esto observamos que la complejidad de la comunicación depende del tipo de topología que se tenga, en la figura 1.1 se muestran algunas topologías de comunicación punto a punto y en la figura 1.2 algunas de comunicación por difusión.

En cualquier tipo de comunicación, se deben definir mecanismos para el control de acceso al medio, estos mecanismos pueden ser centralizados o distribuídos; si la decisión se deja a un procesador particular, el control es centralizado y si cada procesador toma la decisión independientemente entonces decimos que el control es distribuído.

1.1.4 Arquitectura de redes.

Con el objeto de reducir la complejidad una de las tendencias que se ha adoptado en el diseño de las redes es la de organizarlas por niveles. En general, cada red tiene diferentes niveles y el objetivo de cada nivel es ofrecer servicios a los niveles posteriores ocultando los detalles de implementación del servicio que ofrece.

Cuando comunicamos redes del mismo tipo la comunicación entre niveles es transparente a cualquier otro nivel, es decir, parece efectuarse directamente aunque todos sabemos que realmente pasa a las capas inferiores, este proceso de comunicación se efectúa como se muestra en la figura 1.3 [Tan81].

Al proceso para lograr la comunicación entre niveles opuestos se le conoce como protocolo de comunicación en donde se definen las reglas y convenciones para que esta se lleve a cabo. Entre cada nivel adyacente se define una interfase en donde se especifican las operaciones y los servicios que se van a ofrecer al nivel superior. El conjunto de protocolos e interfases es lo que forma la arquitectura de la red. En el diseño de cada nivel se deben tomar en cuenta los siguientes puntos:

- Cada nivel debe tener un mecanismo para establecer la conexión.
- Mecanismo para terminar la comunicación.
- Un punto importante es establecer las reglas para trasmisión de mensajes.
- Control de errores.
- Políticas para ordenar mensajes.
- Políticas para el manejo de información de emisores rápidos.
- Debe tener la habilidad de procesar mensajes de diferentes tamaños.

1.1.5 Estándares internacionales.

Como veremos más adelante, existen diferentes tipos de redes y cada red tiene definidos sus propios niveles; esto trae como consecuencia que no haya una compatibilidad directa entre ellas. Para evitar este problema en cada país se forman organismos que definen las normas que deben seguir

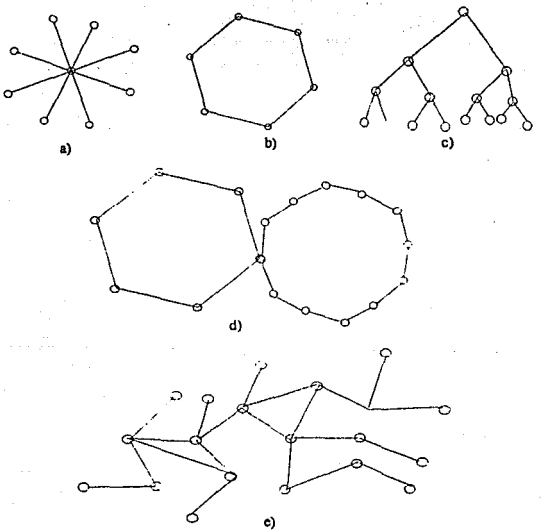

Figura 1.1: Topologías para comunicación punto a punto

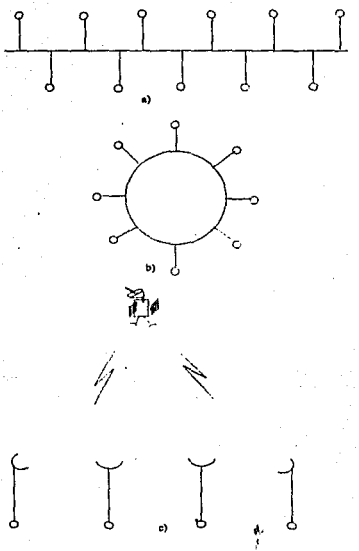

Figura 1.2: Topologías para comunicación por difusión

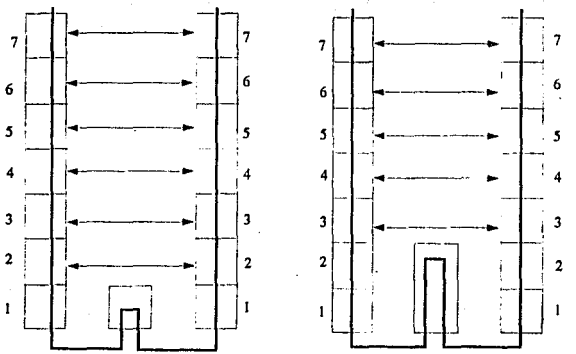

Figura 1.3: Proceso de comunicación entre arquitecturas

los fabricantes de ciertos productos para que haya compatibilidad entre estos. En este caso nos vamos a referir a los estándares de comunicación.

Inicialmente se definieron estándares para la operación interna de las computadoras y conexión de dispositivos locales para cada fabricante; el efecto de esto es que todas las redes de comunicaciones ofrecidas por los fabricantes en términos de software y hardware ha sido sólo entre sus propias computadoras, debido a esto a los sistemas de comunicación se les denomina sistemas cerrados [Tan81].

A partir de este fenómeno y de la necesidad de comunicación entre computadoras de diferentes fabricantes han surgido una serie de organismos internacionales con el propósito de definir estándares de comunicación. Todos estos organismos han definido estándares tanto para la comunicación como para el formato y control que permiten el intercambio de información entre los sistemas. Con estos estándares el equipo de un fabricante se puede interconectar con cualquier otro; el sistema resultante es lo que conocemos como ambiente de interconexión de sistemas abiertos [Tan88].

A mediados de los 70's las industrias de computación comenzaron a darse cuenta de las ventajas que tienen los sistemas abiertos y se comenzó a introducir una serie de estándares. El primer estándar abarcó la estructura completa en la comunicación y fue producido por la "International Standards Organization (ISO)" y se le denominó modelo de referencia OSI(Open Systems Interconection). Este modelo se compone de 7 niveles y cada nivel tiene una estructura bien definida, este modelo se muestra en la figura 1.1, vease [HAL91]. En particular los principios en los que se basa el modelo OSI son:

1. Cada nivel debe crearse de acuerdo al nivel de abstracción necesario.
2. Cada nivel debe ejecutar una función bien definida.
3. La función de cada nivel debe elegirse con miras hacia la estandarización internacional.
4. La función de los niveles se debe elegir para minimizar el flujo de información a través de la interfase.
5. La cantidad de niveles debe ser suficientemente grande como para que no se tengan que juntar funciones distintas y suficientemente pequeña de tal forma que no se vuelva ineficiente o inmanejable.

Como ya mencionamos cada nivel tiene una función bien definida pero no vamos a describir cada nivel, al lector interesado le recomendamos [Tan88, Tan81, HAL91].

En el modelo OSI básicamente tenemos 2 tipos de niveles:

- Los niveles dependientes del hardware que incluye al nivel físico, nivel de enlace y nivel de red.
- Los niveles independientes del hardware que incluye los niveles de transporte, de sesión , el de presentación y el de aplicación.

Para efectuar la transferencia de información está se envía entre los diferentes niveles y en cada nivel se le agrega un encabezado. Para llevar a cabo la comunicación entre niveles, cada nivel posee puntos de interfase por donde se hace el intercambio de información entre éstos.

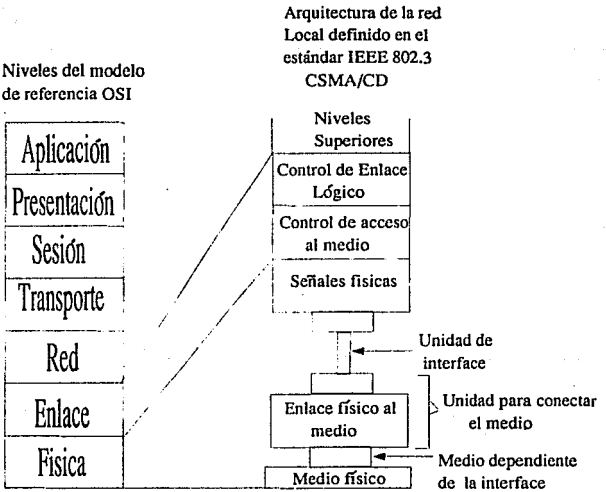

Figura 1.4: Arquitectura del modelo OSI y del estándar IEEE 802.3

1.1.6 Redes locales.

En general las redes se pueden dividir en dos categorías: aquellas que utilizan comunicación punto a punto y las que utilizan conexión por difusión. En las redes por difusión, uno de los problemas que hay que resolver es como controlar el uso del canal cuando hay competencia por su uso. Hay muchos protocolos que resuelven el problema y vamos a describir algunos de estos de manera general. Esta sección básicamente esta dedicada al subnivel (Medium Acces Control(MAC)) el cual es especialmente importante en redes locales por el hecho de que la comunicación se basa en este nivel.

A una red local generalmente la identificamos por las siguientes características:

- Interconexión entre máquinas separadas por pocos kilómetros.
- Comunicación a grandes velocidades (varios Mbytes).
- En general pertenecen a una sola organización.
- Son altamente confiables y la verificación de errores solo se hace en los niveles bajo y superior y esto hace que los protocolos sean más simples y más eficientes.

Entre los mecanismos que se han desarrollado para resolver el problema del uso del canal compartido, encontramos primeramente los trabajos de Norman Abramson y sus compañeros de la Universidad de Hawaii en 1970; a este trabajo le llamaron sistema ALOHA y es aplicable a muchos sistemas en donde los usuarios compiten por el uso de un canal compartido de manera desorganizada.

Básicamente el sistema ALOHA consiste en lo siguiente: los usuarios trasmiten si tienen datos que trasmitir. Si hay colisión los bloques que colisionan se destruyen; si el bloque fue destruído el emisor espera un tiempo aleatorio y envía nuevamente.

En 1972 fué publicado un método que duplica la capacidad del sistema ALOHA, para lograrlo, el tiempo fue dividido en intervalos discretos haciendo corresponder cada intervalo con un bloque. La forma de sincronizar a los usuarios es que cada estación emita una pequeña señal al inicio de cada intervalo. Este método fue denominado ALOHA particionado, mientras que el de Abramson fue llamado ALOHA puro.

Otros de los métodos conocidos para coordinar el uso de un canal compartido son los que identificamos como CSMA(Carrier Sense Multiple Acces) persistentes y no persistentes. En este caso lo que se hace es escuchar si hay trasmisión en el canal y actuar de acuerdo a las circunstancias. Dentro de estos algoritmos existen diferentes categorías, por ejemplo los CSMA 1-persistentes, los no persistentes y los p-persistentes. Pero básicamente lo único que varía es la insistencia en censar el medio y la probabilidad de envío después de que se haya detectado una falla. El algoritmo que siguen estos protocolos es el siguiente: cuando una estación tiene dato para enviar primero escucha en el canal para ver si no hay alguien trasmitiendo. Si el canal esta ocupado la estación espera hasta que se desocupa. Una ves que detecta que se ha desocupado esta envía su mensaje. Si hay colisión la estación espera un tiempo aleatorio e inicia nuevamente. En particular los 1-persistentes son aquellos que siempre van a insistir con probabilidad 1.

Una extensión a estos algoritmos es que cuando las estaciones detecten una colisión éstas dejen de trasmitir, esto evitará la repetición de más bloques de información. A esta extensión se le identifica como CSMA/CD(Carrier Sense Multiple Acces with Colition Detection).

Hay otros protocolos que tienen como objetivo optimizar el uso del canal de comunicación entre estos encontramos a los protocolos libres de colisión, el protocolo de reconocimiento de difusión con prioridades alternantes, entre otros; pero no los vamos a describir, aquel lector interesado en conocer ampliamente estos protocolos consulte [Tan81].

1.1.7 Estándares para redes locales.

La IEEE ha producido diferentes estándares para redes locales, estos estándares se conocen como IEEE 802 que incluye CSMA/CD, "Token bus", y la "Token ring". Estos estándares difieren en el nivel físico y en el subnivel (MAC) pero son compatibles en el nivel de enlace. En esta sección vamos a describir brevemente estos estándares.

El estándar IEEE 802.3 y Ethernet.
Este estándar es un protocolo 1-persistente de CSMA/CD. El estándar inicia con el sistema ALOHA y posteriormente se le agrega la detección de trasmisión en el medio; con esto Xerox construye un sistema CSMA/CD de 2.9 Mbps que permite conectar alrededor de 100 estaciones de trabajo en 1 km de cable, este sistema fue llamado **Ethernet**.

El sistema **Ethernet** de Xerox fué tan exitoso que Xerox, DEC e intel apoyaron el desarrollo de un sistema estándar **Ethernet** de 10 Mbps, este estándar formó la base para el 802.3. El estándar publicado 802.3 difiere de la especificación **Ethernet** en que este describe una familia de sistemas CSMA/CD 1-persistentes que corren desde 1 hasta 10 Mbps en diferentes medios de comunicación. Mucha gente (incorrectamente) usa el nombre de Ethernet refiriéndose a la familia de protocolos CSM/CD, pero en realidad es solo el producto que implementa el 802.3, es decir, **Ethernet** es el cable no los protocolos.

En las redes que utilizan estos protocolos generalmente se utiliza cable coaxial del cual se tienen dos tipos: Cable **Ethernet** delgado y grueso. En este caso, para conectar una computadora al cable por lo general se utiliza ya sea una "T" o un "transceiber". Si en la conexión se utiliza un "transceiber" este tiene la electrónica para enviar datos y detectar colisiones. Este estándar tiene demasiados puntos y recomendamos al lector interesado en ampliar su conocimiento sobre el tema consulte [Tan81].

Estándar 802.4 : Token Bus
Aunque el 802.3 es ampliamente usado en oficinas, durante el desarrollo de los estándares 802 la gente General Motors y otras compañías interesadas en la automatización de fábricas tuvieron ciertos recelos con este protocolo. Por un lado debido a su comportamiento probabilistico para accesar el medio puede suceder que haya estaciones que esperen una cantidad arbitraria de tiempo(es decir, el peor caso es no acotado). Por otro lado este protocolo no ofrece prioridades y cualquier dato por importante que sea es tratado igual que cualquier otro pero en sistemas reales estos datos merecen un trato especial.

Un sistema en donde el peor caso es conocido consiste en un anillo en donde las estaciones esperan turnos para enviar un dato. La gente de automatización de empresas en el comité 802 acepto la idea conceptual del anillo, pero no la implementación física.

Como resultado, se desarrolló un nuevo estándar en donde se ofrece un cable confiable para la difusión y en donde es bien conocido el peor caso; este estándar es el 802.4 y se le denomina "token bus".

Físicamente el "token bus" es un cable lineal al cual están conectadas las estaciones; lógicamente las estaciones están organizadas en forma de anillo en donde cada estación conoce la dirección del vecino izquierdo y derecho. Cuando un anillo lógico es inicializado, la estación con el identificador más grande es la que puede enviar el primer dato, después este pasa el permiso a su vecino inmediato enviándole un dato especial que se le llama ficha. La ficha circula a través del anillo y el nodo que la posee es el único que puede trasmitir.

El protocolo 802.4 es más complejo que el 802.3, ambos protocolos son diferentes en estilo, por ejemplo, el 802.3 se describe con procedimientos tipo Pascal, mientras que el 802.4 está descrito en máquinas de estados finitos con las acciones descritas en Ada.

El estándar 802.5: Token Ring

Las redes de tipo anillo se han venido usando desde hace algunos años y se han usado tanto para redes locales como en redes de área extendida; su principal característica es que posee comunicación punto a punto y su conección forma un anillo. En la comunicación punto a punto se tiene una tecnología clara y bien probada.

En este caso se conocen tanto cotas inferiores como superiores del tiempo de acceso al canal; por estas razones IBM ha escogido una arquitectura de tipo anillo para red local y IEEE ha incluido un anillo en el estándar 802.

Existen diferentes tipos de anillos, al estandarizado en 802.5 se le conoce como **token ring**. Como mencionamos, un anillo consta de una colección de interfases conectadas punto a punto y cada bit que llega a la interfase se copia en un "buffer" y posteriormente se envía nuevamente al anillo. Cuando la información esta en el "buffer" es inspeccionada y posiblemente modificada antes de enviarla nuevamente al anillo.

En la red **token ring** hay un patrón de bits denominado "token" que circula a lo largo del anillo cuando todas las estaciones están inactivas. Cuando una estación desea trasmitir esta se apodera del "token" quitándolo de circulación antes de trasmitir. Cada interfase en la red tiene dos modos de operación, escucha o trasmite, cuando esta escuchando los bits de entrada se copian directamente a la salida; el modo de trasmisión se alcanza después de que el "token" se ha capturado, en este momento se rompe la conexión entre la entrada y la salida y envía sus propios datos al anillo.

Como los bits que se han propagado a lo largo del anillo regresan en algún momento, estos son eliminados del anillo por el emisor, la estación emisora puede ya sea guardarlos para compararlos con el original para verificar la confiabilidad del anillo o para descartarlos. La arquitectura del anillo no pone límite al tamaño de los datos porque el dato completo nunca aparece en el anillo al mismo instante. Después de que una estación a terminado de trasmitir el último bit de su último dato éste debe regenerar la ficha; cuando el último bit le ha dado la vuelta al anillo este es eliminado y la

estación cambia inmediatamente a modo escuchar evitando eliminar la ficha que sería el siguiente patrón de bits si es que no hubo otra estación que la eliminó.

En esta arquitectura cuando hay poco trafico en la red el token va a gastar la mayor parte de su tiempo circulando en el anillo sin embargo cuando el trafico en la red es denso de tal forma que en cada nodo hay una cola, el permiso para enviar rota lentamente a lo largo del anillo.

En el nivel físico, 802.5 utiliza par trenzado y corre entre 1 o 4 Mbps. El lector interesado en profundizar sobre este protocolo consulte [Tan81].

1.2 Sistemas Distribuídos.

Las primeras cuatro décadas de la tecnología en computación se caracterizan por las diferentes formas de uso de la computadora; en los años 50's los programadores apartaban tiempo de cómputo y disponían de toda la computadora mientras la estaban utilizando, en los 60's se utiliza el procesamiento en lote. En esta epoca los usuarios ponían sus tareas en cola para ser procesadas y posteriormente regresaban por sus resultados. Los sistemas de tiempo compartido aparecen en los 70's en donde el usuario puede compartir una computadora pero haciéndole creer que está dedicada completamente a él. Los 80's es la época de las computadoras personales (los usuarios tienen su propia máquina).

La evolución que se ha dado en los sistemas operativos ha sido posible debido a razones económicas y a los desarrollos tecnológicos. Los sistemas en lotes se desarrollaron debido a que se construyeron "chips" de memoria que podían almacenar grandes cantidades de información, por ejemplo, el sistema operativo y una aplicación a la vez; esto trae como consecuencia un uso más eficiente de una computadora con alto costo. Los sistemas de tiempo compartido son deseables porque permiten mayor productividad en la programación y un uso más eficiente del sistema. Esto se logró debido a que las computadoras fueron más baratas y más poderosas. Con la integración a gran escala (VLSI) y la aparición de las redes locales, las estaciones de trabajo han sido una buena alternativa para sistemas de tiempo compartido con la misma capacidad de cómputo.

Actualmente un procesador con poder para satisfacer muchas necesidades de una sola persona cuesta menos que un décimo de lo que cuesta un procesador poderoso que puede servir a diez personas. Los sistemas de tiempo compartido no son del todo satisfactorios en la forma como se usa el sistema, por ejemplo, el desplegado de un mapa de bits en una interfase gráfica demanda una respuesta visual instantánea de parte del sistema gráfico; esto se lograba sólo con una computadora personal dedicada.

En los 90's las estaciones de trabajo son mucho más poderosas que los sistemas de tiempo compartido, estas tienen monitores a colores de alta resolución y dispositivos de entrada y salida de voz y video. Las interfases de red permiten comunicaciones a velocidades que satisfacen los requerimientos de los canales de trasmisión de video en tiempo real.

Un sistema de tiempo compartido prové al usuario de un ambiente que permite compartir recursos tales como impresoras, espacio de almacenamiento, software y datos. Para permitir a los usuarios el acceso a los servicios, las estaciones de trabajo en general están conectadas por una red y el software del sistema operativo de la estación de trabajo permite copiar archivos y establecer sesiones remotas

a través de la red de una estación de trabajo a otra. Aquí los usuarios deben conocer la diferencia entre objetos locales y remotos y deben conocer a cúal máquina remota están conectados. En un sistema de cómputo con muchas estaciones de trabajo esto se convierte un serio problema.

En los sistemas de tiempo compartido los operadores pueden efectuar un respaldo del sistema de archivos todas las noches, el administrador del sistema puede asociar tiempo de procesador donde mas se necesite y los programadores de sistemas pueden ya sea instalar nuevo software o mejorar el que ya existe; sin embargo, en el ambiente de las estaciones de trabajo cada usuario puede ser operador, administrador y programador del sistema. En un edificio con cientos de estaciones de trabajo autónomas ningún operador puede hacer respaldo del sistema, ni los programadores de sistemas pueden instalar software simplemente cargándolo al sistema de archivos.

Se han propuesto algunas soluciones a este problema pero ninguna ha sido satisfactoria para el ambiente distribuído como el caso de los sistemas compartidos, por ejemplo, una aproximación muy conocida consiste en agregar comandos de copiado en red que permiten transferir archivos de una estación de trabajo a otra, u otra alternativa un poco mejor consiste en sistemas de archivos que permiten compartir archivos reales. Con estas soluciones, el usuario debe saber distinguir entre operaciones locales y remotas. El problema es que los sistemas operativos reales que forman la base del software de las estaciones de trabajo actuales nunca fueron diseñados para un ambiente con muchos procesadores y muchos sistemas de archivos. Para este tipo de ambiente se requiere de un sistema operativo distribuído.

Los 90's es la época de los sistemas distribuídos y dar una definición es un poco aventurado pero quiza hasta ahora una de las definiciones más razonables es la dada por Tanenbaum y van Renesse [Mul89] y es la siguiente:" **Un sistema operativo distribuído es aquel ve a los usuarios como los ve un sistema operativo centralizado, pero corre en procesadores múltiples e independientes.** Aquí, el concepto clave es la transparencia, es decir, el uso de múltiples procesadores y la presencia de fallas debe ser transparente para el usuario; el usuario ve al sistema como un solo procesador.

A cambio de dar una definición precisa de lo que es un sistema distribuído, vamos a dar algunas de las características que debe poseer un sistema distribuído:

- Un sistema distribuído debe ser capaz de continuar si existe una sola falla y además continuar con la ejecución paralela, por lo tanto debe tener múltiples elementos de procesamiento que se ejecutan independientemente; esto implica que cada elemento de procesamiento o nodo debe contener al menos un CPU y una memoria.

- Debe haber comunicación entre los elementos de procesamiento, por lo tanto un sistema distribuído debe contener un hardware de interconexión que permita correr procesos en paralelo que se comuniquen y sincronicen.

- Un sistema distribuído no puede tolerar fallas si todos los nodos fallan simultáneamente, por lo tanto el sistema debe estar estructurado de tal forma que los elementos de procesamiento fallen independientemente.

- Para que haya recuperación de fallas es necesario que los nodos guarden un estado compartido del sistema distribuído, si no se tiene este estado, la falla de un nodo puede provocar la pérdida de una parte del estado del sistema.

- Costo y desempeño: el costo de una computadora depende de su desempeño en términos de velocidad de procesamiento y la cantidad de memoria que posee, los costos tanto del procesador como de la memoria están disminuyendo y cada año con el mismo dinero se compra una estación de trabajo mucho más poderosa. El costo de la comunicación depende del ancho de banda y la longitud del canal de comunicacióny actualmente los costos de la comunicación están disminuyendo mucho más rápido que el costo de las computadoras. La velocidad de comunicación entre computadoras y pantalla ha aumentado enormemente. esto es especialmente ilustrativo en aplicaciones gráficas: también la eficiencia en los canales de comunicación ha aumentado tanto que actualmente en las estaciones de trabajo se puede emitir voz y gráficas para animación.

- Modularidad: ahora en los sistemas distribuídos las partes de un sistema tienen que ser diseñadas con mucho más cuidado que en los sistemas centralizados, como consecuencia los sistemas distribuídos deben ser construidos de una forma más modular que los sistemas centralizados. Una de las tareas típicas en los sistemas distribuídos es correr servicios en la máquina local pero físicamente están en otra máquina, esto se hace con llamadas a procedimientos remotos en los cuales se impone un cierto estándar para la interfase entre módulos.

- Expandible: los sistemas distribuídos son capaces de crecer, para aumentar la capacidad de almacenamiento o procesamiento de un sistema distribuído se deben agregar un servidor de archivos o procesador a la vez.

- Disponible: como los sistemas distribuídos replican datos y tienen información redundante de todos los recursos que pueden fallar, estos deben estar potencialmente disponibles aún cuando haya ocurrido una falla arbitraria en un solo punto.

- Escalabilidad: La capacidad de cualquier componente centralizado de un sistema impone un límite en el tamaño máximo del sistema, idealmente los sistemas distribuídos no tienen componentes centralizados así que esta restricción de crecimiento no existe en este caso. Claramente pueden existir otros factores que restringen la escalabilidad del sistema pero los diseñadores de sistemas distribuídos tratan de encontrar algoritmos que incluyan gran cantidad de componentes.

- Confiabilidad: La disponibilidad es un aspecto de confiabilidad, un sistema confiable no sólo debe estar disponible, sino que debe hacer lo que es correcto aun cuando existan fallas. Los algoritmos usados en sistemas distribuídos no sólo deben comportarse correctamente cuando todo funcione bien sino también deben ser capaces de recuperarse de fallas y continuar con su ejecución de manera normal.

1.2.1 Problemas complejos en los sistemas distribuídos.

Una de las principales razones por la cual el diseño de los sistemas distribuídos es difícil es que la complejidad de estos sistemas aún está fuera de nuestro alcance. Se conocen ciertas razones de la complejidad pero aún se está lejos de poder diseñar y construir sistemas distribuídos confiables.

La mayoría de las veces para poder encontrar fallas en un sistema es necesario que ocurra dicha falla pero una falla puede traer consecuencias graves, algunas veces el descubrimiento de una falla implica la perdida de vidas; en general este proceso es inevitable y se tienen que producir los errores para aprender de estos.

Una de las principales fuentes de complejidad en los sistemas distribuídos es que la interconexión de componentes bien conocidos genera nuevos problemas en estas componentes; mucha de esta complejidad se debe al comportamiento inesperado del sistema que aparentemente creemos entender.

Actualmente se han estado desarrollando algunas herramientas formales que nos ayudan a predecir lo que a va a suceder cuando dos sistemas se interconectan, sin embargo estos métodos son de ayuda limitada principalmente cuando no se entiende completamente el sistema o simplemente aún no existen herramientas formales para describirlos completamente.

Los problemas más comunes se presentan por:

- Interconexión: se debe principalmente a fallas en las conexiones.
- Interferencia: se debe principalmente a la presencia de ruido en las líneas de comunicación.
- Propagación de efecto: es decir, la falla de un componente puede provocar la caída de toda la red si el diseño de un sistema no se hace con suficiente cuidado.
- Efecto de escala: este efecto se refiere a que en el momento de diseño no se piensa por ejemplo en el crecimiento de la red sino que se diseña sólo para la red que se tiene en ese momento(10 nodos por ejemplo) y si la red crece pueden aparecer cuellos de botella o simplemente no funciona el sistema.
- Fallas parciales; este problema se refiere a que el sistema debe soportar fallas parciales y esto es una fuente de complejidad adicional en el diseño de aplicaciones tolerantes a fallas.

Algunos de los puntos que podemos recomendar para el diseño de sistemas distribuídos son los siguientes:

- Replicar información para incrementar la disponibilidad.
- Considerar qué es más importante: disponibilidad o consistencia y dependiendo del resultado de la consideración se sacrifica una o la otra o se busca balancear ambos puntos.
- Almacenar copias de información clave cuando la información remota no esté disponible.

- Usar "TIMEOUT" para la liberación de recursos; hay muchos casos en sistemas distribuídos donde un proceso bloquea algunos recursos mientras los esta utilizando, cuando el proceso o la máquina se caen, el recurso puede quedar inaccesible a otros.

  Como principio general, es bueno utilizar "TIMEOUT's" en el bloqueo de recursos, los clientes deben estar actualizando sus candados o información bloqueada periódicamente, de lo contrario pueden perderlos.

- Utilizar un mecanismo estándar para la invocación de procesos remotos.

- Correr protocolos o programas en máquinas que tengan sistemas operativos confiables y que físicamente estén seguras.

- Tratar de usar encriptamiento para mantener autenticidad y seguridad de los datos.

- Tratar de probar los algoritmos distribuídos: las pruebas formales han ganado popularidad y actualmente existen diferentes técnicas. Razonar acerca de la correctez de un algoritmo secuencial es difícil y razonar sobre la correctez de un algoritmo paralelo es prácticamente imposible. Las pruebas de correctez por si solas no son suficientes para garantizar que el programa distribuído sea correcto; después de todo los algoritmos están casados con el mundo ideal y el programa esta casado con el mundo real. Las condiciones de frontera usadas en la prueba pueden no corresponder a las condiciones reales; pueden ocurrir fallas inesperadas que el modelo no toma en cuenta, de esto a lo que llegamos es que la construcción de un sistema distribuído confiable requiere de la combinación de técnicas formales y técnicas para prueba.

1.2.2 Objetivos de las aplicaciones distribuídas, principales técnicas de implementación y características de las arquitecturas de los sistemas distribuídos que más influyen en la implementación de programas distribuídos.

En realidad para desarrollar aplicaciones distribuídas es necesario categorizarlas dentro de un conjunto de características y después seleccionar del conjunto de paradigmas de programación los apropiados para dicha aplicación. En esta sección vamos a anotar los paradigmas más importantes de la programación distribuída y además se describirán las características más importantes de las arquitecturas distribuídas.

Describir el desempeño de las aplicaciones distribuídas depende mucho de la aplicación especifica y la arquitectura bajo la cual se este ejecutando; para la descripción de esta sección nos vamos a basar en el modelo de la figura 1.5.

El nivel de la arquitectura base que se descompone en el nivel de hardware que comprende los nodos y la comunicación entre nodos y el núcleo del sistema operativo que soporta llamadas de control, espacios de direcciones, multiplexado de procesadores, comunicación entre procesos, etc. El nivel de programa distribuído que implementa algoritmos distribuídos y no distribuídos que son necesarios para responder a las aplicaciones.

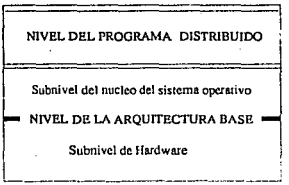

Figura 1.5: Arquitectura base de un sistema distribuído

El nivel de hardware de la arquitectura base comprende nodos de procesamiento y la comunicación de la red. Los nodos de procesamiento pueden o no tener fallas drásticas; si fallan se supone que se detienen y no ejecutan cálculos erróneos. Además estos nodos pueden tener uno o muchos procesadores. Los multiprocesadores pueden usar ya sea memoria compartida o paso de mensajes.

En general, puede haber muchos tipos de nodos de procesamiento en el mismo sistema distribuído. Los nodos de procesamiento pueden o no fallar independientemente. La base de hardware puede estar en un solo gabinete o puede estar geográficamente disperso. Todos los nodos de procesamiento pueden tener memoria volátil en donde residen los objetos a ser procesados; o también pueden tener memoria no volátil en donde residen los objetos que han sido accesados mas recientemente; otro tipo de memoria es la memoria estable en donde a pesar de la presencia de fallas no pierde la información.

La base de hardware también incluye la comunicación de la red que prové de comunicación entre los nodos; esto comprende comunicación local y comunicación a distancia. En este caso se supone que la red nos ofrece conectividad completa.

En la capa del núcleo del sistema operativo se implementan espacios de direcciones protegidas, llamadas de control, sincronización local y comunicación entre llamadas remotas y locales.

La arquitectura base se puede caracterizar de acuerdo a una serie de puntos que van a ayudar a describir desempeño, confiabilidad y funcionalidad.

Entre los puntos que caracterizan la arquitectura tenemos:

1. Los relacionados con la comunicación que tiene como elementos a:

   (a) Estructura principal que permite llamar a otras estructuras de control. Aquí se describe la estructura principal para la comunicación entre procesos ya sea local o remota.

   (b) Tiempo transcurrido para llamadas asíncronas a otras estructuras que incluye tiempos de transferencia asíncrona de entrada/salida en la red o en los sistemas de memoria.

   (c) Ancho de banda de la comunicación que caracteriza la disponibilidad de datos entre dos procesos que se comunican de manera local o remota.

(d) Distancia de la red que describe la distancia geográfica en la cual están distribuídos los nodos.

2. Los relacionados con fallas, que incluye:

   (a) La velocidad de falla y de recuperación.

   (b) Integridad del procesador y de la memoria en donde se indica si los nodos de procesamiento o la memoria fallan de manera detectable.

   (c) Independencia de fallas, en donde se especifica si las fallas afectan la operación del sistema en general.

   (d) Disponibilidad de almacenamiento estable que especifica si los nodos tienen memoria disponible que sobreviva a fallas anticipadas.

3. Los relacionados con paralelismo, que incluye:

   (a) Número de procesadores en un nodo de procesamiento lo cual me da una medida del paralelismo en el nodo.

   (b) Número de nodos en la red que me da una medida del paralelismo disponible entre los nodos de la red.

4. Los relacionados con el desempeño, en donde se incluyen:

   - Desempeño de los nodos en donde se describe su desempeño y su capacidad de memoria.
   - Frecuencia de acceso y ancho de banda de la memoria estable en donde se especifica la velocidad de lectura y escritura a la memoria estable.
   - Frecuencia de acceso y ancho de banda de la memoria secundaria en donde se especifica la velocidad de lectura y escritura a la memoria secundaria.

5. Entre otros puntos importantes tenemos:

   (a) Homogeneidad de los nodos de procesamiento que describe el grado de substitución de un procesador por otro.

   (b) Autonomía de los nodos de procesamiento en donde se especifica el grado de dependencia de un nodo con respecto a periféricos o a otros nodos.

   (c) Seguridad física en donde se especifica que tan seguro esta cada nodo de la red.

Para mayores referencias sobre este tema consultese [Mul89, Cri88, And91]

1.3 Protocolos de comunicación.

Los protocolos son conjuntos de reglas que rigen la interacción de procesos concurrentes en sistemas distribuídos; su diseño es una área que engloba diferentes campos de computación tales como sistemas operativos, redes de computadoras y comunicación de datos. A esta área no se le ha

dedicado mucho tiempo de estudio por si sola, en general se estudia junto con los otros campos con los cuales tiene relación, sin embargo el diseño de protocolos lógicamente consistentes cada vez toma mayor importancia debido principalmente a la aparición de los sistemas distribuídos. En esta sección nos vamos a inclinar mas hacia el diseño y análisis de protocolos que a la descripción de protocolos actualmente aceptados como estándares internacionales. Se van a describir las reglas de diseño de protocolos y la manera de usar las herramientas formales de las que hablaremos en el capítulo 3 para la prueba de estos.

Los problemas de diseño de protocolos de comunicación eficientes y no ambiguos han existido desde hace mucho tiempo, ha habido muchos intentos para construir sistemas para transferir información a grandes distancias y uno de los puntos clave en el diseño de protocolos es que no se esta seguro de la existencia de una sucesión de mensajes que hagan fallar el protocolo, es por eso que uno de los objetivos en el diseño de protocolos es tratar de esperar lo inesperado.

La necesidad del diseño de protocolos incrementó dramáticamente después de los años 50 cuando la ejecución de estos se lleva acabo en computadoras; en esta época surge la necesidad de comunicar la computadora con algunos periféricos como impresoras, terminales y otros; posteriormente se piensa en la posibilidad de comunicar diferentes computadoras a grandes distancias. El primer protocolo de comunicación que se ejecuta en computadoras fué para enviar código a través de operaciones manuales y básicamente el problema es la coordinación entre maestro y esclavo; en este caso en todo momento alguna de las dos partes participantes en la comunicación tiene el control y es responsable de toda la transferencia, recuperación, sincronización y manejo de la conexión.

Conforme ha aumentado la velocidad de trasmisión la carga de trafico ha aumentado y se ha dejado de lado la comunicación tipo **maestro-esclavo** porque esto genera cuellos de botella.

A partir de la creación de la primer red de computadoras de propósito general para el intercambio de mensajes desarrollada por el departamento de defensa de los Estados Unidos y de la evolución de los medios de comunicación surgen otros problemas en el diseño de protocolos. En este caso el problema es poder establecer acuerdos para el uso de recursos compartidos en una red de procesadores. En principio no es claro qué proceso es responsable por cual tarea, estas responsabilidades pueden ser negociadas porque si mas de un proceso de manera errónea asume responsabilidad por una tarea puede causar problemas.

Los diseñadores iniciales de las redes se dieron cuenta de que es muy probable la aparición de sucesiones de eventos que pueden arruinar el mejor diseño y se puede inhibir la red completa si se diseña un protocolo defectuoso o incompleto y una colisión de paquetes en una red puede ser tan grabe como la colisión de dos trenes. Pero veamos una definición tentativa sobre protocolos de comunicación.

Definición 1 *Un protocolo es un tipo de acuerdo para el intercambio de información en un sistema distribuído.*

En realidad la definición completa de un protocolo se parece mas a la definición de un lenguaje, por ejemplo, se define un formato preciso para mensajes validos (sintaxis), un conjunto de reglas para el intercambio de mensajes(gramática), un vocabulario de mensajes validos que pueden ser intercambiados con un significado(semántica).

Algo muy importante es que la gramática de los protocolos debe ser lógicamente consistente y completa y bajo estas circunstancias las reglas deben prescribir sin ambigüedad lo permitido y lo prohibido, pero prácticamente este es un requerimiento difícil de lograr.

En resumen, los protocolos que se han desarrollado actualmente son muy grandes y sofisticados y el objetivo es ofrecer funcionalidad y confiabilidad pero el problema al que se enfrenta el diseñador es cómo diseñar un conjunto de reglas para el intercambio de información que sean mínimas, lógicamente consistentes, completas e implementables eficientemente. Este problema puede ser abordado desde dos puntos de vista.

1. Dado el problema, cómo puede el diseñador resolverlo sistemáticamente de tal forma que se logren los requerimientos de diseño.

2. Dado el protocolo, cómo puede un analizador demostrar convincentemente que éste cumple con los requerimientos de correctitud.

Aunque desde hace tiempo los protocolos de una forma u otra se han usado para comunicar sistemas a grandes distancias hasta muy recientemente estos habían sido manejados por el humano y mediante el sentido común se pueden resolver los problemas inesperados.

Con la sustitución del humano por las computadoras claramente siguen existiendo los mismos problemas de comunicación y coordinación pero en este caso los errores se presentan más frecuentemente y no hay posibilidad de que intervenga el humano para atender los casos inesperados.

Un requerimiento importante en el diseño de protocolos es que no sólo debe haber reglas para el intercambio de información si no que también debe haber un acuerdo entre el emisor y el receptor sobre el uso de estas reglas.

Hay diferentes organismos internacionales dedicados a definir los requerimientos que deben cumplir ciertos protocolos. los organismos más importantes son la "International Standards Organization (ISO)" que incluye diferentes organismos nacionales tal como la "American National Standards Institute (ANSI)". Otro de los organismos importantes son "The Comité Consultatit International Télégraphique et Téléfonique (CCITT) que se formo por la unión de "CCIT (Telegraph Systems)" y el "CCIF (Telephone Systems". Para mayor referencia véase [Tan88, Tan81, BOC90, PEH90, Hol91]

En la definición de protocolos debe haber acuerdos para:

- La inicialización y terminación del intercambio de mensajes.
- La sincronización de emisores y receptores.
- La detección y corrección de errores de trasmisión.
- Formatear y codificar los datos.

Los elementos que debemos tomar en cuenta en la especificación son:

- Qué servicio debe proveer el protocolo.

- Cúales son los supuestos acerca del medio ambiente en el cual el protocolo se va a ejecutar.
- Cúal es el vocabulario de los mensajes que van a enviarse en el protocolo.
- Cúal es el formato de cada mensaje en el vocabulario.
- Definir las reglas para el intercambio de información que garanticen la consistencia en los mensajes intercambiados.

Por otro lado, cuando diseñamos protocolos debemos tratar de que se cumplan una serie de puntos como los siguientes:

1. Simplicidad: un protocolo bien estructurado se puede construir a partir de piezas pequeñas bien estructuradas y bien diseñadas y cada pieza ejecuta una función. Para entender el funcionamiento de un protocolo debe ser suficiente entender el funcionamiento de las piezas de las cuales esta construido y la forma de interactuar; los protocolos que son diseñados de esta forma son fáciles de entender y fácil de implemetar, además son más fáciles de verificar y darles mantenimiento.

2. Modularidad; un protocolo que ejecuta una tarea compleja se puede construir a partir de piezas mas pequeñas que interactúan de una forma simple y bien definida y cada pieza se puede desarrollar, verificar, implementar y mantener por separado; con esto lo que obtenemos son protocolos con estructura abierta, entendibles y modificables sin afectar la función de los demás componentes individuales.

3. Un punto importante es que los protocolos deben estar bien formados, es decir un protocolo que no contiene código muerto ni tampoco le debe faltar código porque en este caso pueden haber valores no especificados durante su ejecución. Una de las características que debe tener un protocolo bien formado es que sea acotado, lo cual quiere decir que este no debe sobrepasar los límites del sistema, como por ejemplo la capacidad de la cola de mensajes.

4. Autoestable, que un protocolo sea autoestable quiere decir que si hay un error que manda al protocolo a un cierto estado, este debe regresar siempre a un estado deseable en un número finito de transiciones y continuar operando normalmente; o equivalentemente si un protocolo inicia en un estado, este siempre debe llegar a un estado válido en un tiempo finito.

5. Autoadaptable, un protocolo es autoadaptable si es capaz de adaptar la velocidad a la cual se reciben los datos con la velocidad a la que se pueden transferir y además a velocidad a la cual el receptor puede consumirlos.

6. Robustez, que un protocolo sea robusto quiere decir que este debe ser capaz de reaccionar a los eventos inesperados y en particular requieren de ayuda y consideración instantánea.

7. Consistencia; algunas de las puntos importantes que debemos considerar para decir que un protocolo es consistente son:

(a) Abrazo mortal se presenta cuando se detienen dos o más procesos porque están esperando que se cumplan condiciones que jamás se van a presentar.

(b) Ciclos infinitos se refiere a la ejecución de una sucesión de instrucciones sin presentar algún avance.

(c) Terminaciones impropias se refiere a la terminación de la ejecución de un protocolo sin satisfacer las condiciones de terminación.

En general la verificación de estas propiedades no se puede efectuar por inspección manual y se requiere del uso de herramientas que permitan prevenir o detectar tales propiedades. Una de las herramientas que vamos a usar en este caso son las redes de Petri, las cuales se describen en un capítulo posterior.

1.4 Fallas.

Las fallas en general es un tema demasiado amplio y existen muchos reportes de investigación en donde se desarrollan algoritmos en redes tolerantes a fallas pero no existen algoritmos que toleren cualquier tipo de falla. En esta sección vamos a describir de manera general las diferentes fallas que se han considerado en el desarrollo de algoritmos tolerantes a fallas.

Las fallas pueden presentarse tanto en las líneas de comunicación como en los nodos y en cada caso se pueden presentar diferentes señales de falla.

Dado que en las redes no es posible tener un ambiente de comunicación perfecto (sin fallas en los nodos o de las líneas) entonces la única salida es diseñar algoritmos o protocolos que sean capaces de recuperar información a pesar de la presencia de fallas.

Para que un algoritmo se recupere a pesar de la presencia de fallas, es necesario tener en cuenta algunos puntos; por ejemplo.

- Replicar la información; de esta forma si algún nodo falla, es posible recuperar la información de los otros en los cuales se tenía replicada, para lograr esto se necesita un cierto número de mensajes extra y también cierta cantidad de memoria adicional, pero al mismo tiempo esto alenta el sistema y por lo tanto disminuye la eficiencia en cierto porcentaje aunque no necesariamente haya fallas porque todo esto implica trabajo extra para estar preparado para el momento que se presenten.

- Tener procesos para detección de fallas; estos procesos pueden tener deferente grado de complejidad dependiendo del tipo de falla que se desee detectar. En este caso la disminución de la eficiencia o alentamiento del sistema se presenta en el momento de presentarse la falla.

Entre los tipos de fallas que se han considerado en la literatura encontramos a los siguientes:

- Caída de nodos: en este caso se considera que el nodo deja de operar inadvertidamente y una ves que se detiene no envía ningún mensaje, ni genera ninguna perturbación extra.

- Falla en las líneas, en cuyo caso puede suceder que la línea de comunicación se parta y entonces la red se divide en dos partes ajenas sin posibilidades de comunicación entre estas. Para la recuperación de estas fallas por lo general es necesario detener las partes de la red.
- Falla por omisión de envíos; lo cual quiere decir que un proceso posiblemente no envié algunos mensajes que estaban prescritos en el algoritmo.
- Falla por omitir recepciones; en cuyo caso se considera que puede haber nodos que no reciben algunos mensajes que le fueron enviados.
- Fallas arbitrarias de procesos sin restricciones en su comportamiento.

En general casi todos los algoritmos tolerantes a fallas han considerado algunas de las descritas arriba, pero en realidad las fallas no son tan simples, por ejemplo un nodo sin detenerse puede estar enviando mensajes que generan ruido en las líneas. Otro caso común es cuando en las líneas de comunicación se esta generando ruido aleatorio lo cual es difícil predecir si es o no una falla. Otra posibilidad de error es cuando se caen las líneas de comunicación en cuyo caso ni el hardware funcion; es por eso que diseñar algoritmos que sean capaces de recuperarse de cualquier falla es imposible.

En el diseño de los algoritmos tolerantes a fallas entre más fallas se toleran más ineficiente se vuelve el sistema. Dado que actualmente las redes de computadoras son cada vez más confiables y tomando en cuenta que una serie de medidas probabilisticas acerca de la frecuencia de fallas, se están diseñado algoritmos que toleren ciertas fallas más frecuentes pero que al mismo tiempo sean eficientes. Para el lector interesado en temas de tolerancia fallas consulte [Mul89, Lam78a, HH91, JJS84]

Capítulo 2

Estudio general de los protocolos por difusión.

En el diseño de sistemas distribuídos, una herramienta básica es la difusión confiable y eficiente de mensajes en la red. Una vez que se tenga una herramienta de este estilo, podemos construir herramientas de un nivel más alto en donde la utilicemos como una primitiva. La necesidad de una primitiva con estas características se hace plausible en el momento del diseño de sistemas distribuídos tales como despachadores distribuídos, balanceadores de cargas, manejo de bases de datos distribuídas, etc.

Actualmente la comunicación entre las redes nos ofrecen protocolos confiables entre diferentes entidades de la red, sin embargo en los sistemas distribuídos se requiere de la cooperación entre una colección de entidades. En particular las entidades del sistema distribuído necesitan enviar mensajes al resto de las entidades que están cooperando en la ejecución de un proceso. Si se puede ofrecer comunicación por difusión entre todas las entidades de manera confiable entonces la creación de sistemas distribuídos se puede efectuar de manera más fácil y más eficiente.

Uno de los problemas a los que nos enfrentamos en la implantación de protocolos por difusión es que la red no es del todo confiable, algo que debemos garantizar en los protocolos por difusión es que a cada nodo le deben llegar todos los mensajes y además el orden en que los reciba un nodo debe ser el mismo para todos, es decir, debe haber un orden total de los mensajes.

Un problema a resolver en el diseño de protocolos es, decidir quien va a coordinar la cooperación entre las diferentes entidades. Se han diseñado algunos protocolos [MFK89, CM84] en donde se centraliza el control a un líder del grupo y éste decide el orden como deben recibirse los datos en los demás nodos; en este caso los procesos deben esperar la decisión del líder.

En los protocolos de control distribuído todas las entidades deciden localmente el orden correcto de los datos que reciben en base a los datos que han recibido [NT91].

En este capítulo vamos a analizar las características de las dos tendencias de desarrollo de protocolos por difusión con control centralizado y control distribuído.

2.1 Protocolos por difusión de control centralizado.

2.1.1 Selección de líder.

En los protocolos de control centralizado uno de los principales problemas es la selección de un líder o nodo coordinador, el cual se va a encargar de asignar el orden a los datos que se van a difundir y posteriormente difundirlos. Pero la selección de líder no es un problema sencillo, se debe asegurar la unicidad del líder, es decir, debemos asegurar que en cualquier momento todos los nodos reconozcan al mismo líder.

Para poder asegurarlo necesitamos implementar algoritmos para la selección de líder y además el líder debe ser un nodo con un poder de procesamiento que garantice la atención eficiente de todos los nodos que cooperan en la difusión de mensajes. Para mayores referencias sobre algoritmos para elección de líder véase [EK89].

2.1.2 Ventajas y desventajas de usar protocolos por difusión de control centralizado.

El usar protocolos de control centralizado tiene ciertas ventajas pero al mismo tiempo sus desventajas.

Ventajas

1. En los protocolos centralizados basta hacer un control cuidadoso en el nodo que se encarga de difundir los mensajes y no hacen falta cambios en ningún otro nodo.
2. Son fáciles de diseñar, implementar y fácil de soportar cambios para nuevas extensiones.
3. Es más fácil asegurar el orden de los mensajes.
4. Se pueden soportar fallas muy fácilmente.
5. Si hay falla en algunos de los nodos, es fácil de recuperar su información.

Desventajas

1. Con el aumento de nodos, fácilmente se pueden presentar cuellos de botella en el nodo central porque conforme aumentan los nodos aumenta su carga.
2. La eficiencia del protocolo disminuye conforme aumenta la cantidad de nodos.
3. Una gran desventaja es que si falla el nodo central, falla el protocolo completo.
4. Aumentar la eficiencia de un protocolo centralizado puede ser demasiado costoso.

2.1.3 Restricciones de los protocolos de control centralizado en presencia de fallas.

En general soportar fallas es un problema difícil. En los protocolos centralizados no se puede soportar una falla en el nodo central, esto se debe a que el control lo tiene este nodo y todos los demás nodos que participan en el protocolo extraen información de éste. En este caso la falla de algún otro nodo que no sea el líder no afecta tanto la evolución del protocolo.

Una posible solución a este problema es que la información no solamente se guarde en el nodo central sino que también se quede en algunos otros nodos que participen en el protocolo.

Otra solución en la que no aumenta el número de mensajes en la red es que cada mensaje enviado por alguno de los nodos sea mantenido por éste hasta que el líder confirme la llegada del mensaje al resto de los nodos. Con este mecanismo, el mensaje va a estar en el nodo líder y en el nodo que lo envió y en caso de que falle el líder, el mensaje se puede recuperar del nodo emisor. Una restricción en este caso es que si falla el nodo centralizado y algún otro nodo, no se podrán recuperar los mensajes del nodo que haya fallado.

2.2 Protocolos por difusión de control distribuído.

Este tipo de protocolos se vuelve cada ves más interesante porque una implementación de un protocolo de control distribuído que sea confiable, eficiente y tolerante a fallas puede revolucionar los sistemas distribuídos, porque después de todo los sistemas distribuídos estarán montados sobre este tipo de servicios que al mismo tiempo tienen un control distribuído. Entre las **ventajas** que podemos encontrar en estos protocolos tenemos las siguientes:

1. Cada nodo decide el orden correcto de los mensajes de acuerdo a la información que le ha llegado y esto trae como consecuencia la disminución del número de mensajes.
2. Estos protocolos no disminuyen su eficiencia por el aumento de nuevos nodos.
3. El aumentar nodos no requiere ningún cambio.
4. Estos protocolos aumentan su eficiencia al agregar nodos mas eficientes.

Como desventajas podemos mencionar:

1. En estos protocolos, no es fácil lograr un orden total de los mensajes, por lo general se logra un orden parcial.
2. Estos protocolos son difíciles de implantar.
3. No es fácil darse cuenta de la presencia de fallas.

4. Difícil de recuperarse de fallas.

5. Difícilmente se pueden soportar fallas tales como caída de nodos.

2.2.1 Posibilidades de implantación de un protocolo con control distribuído.

Implantar un protocolo distribuído confiable y eficiente no es un problema sencillo. Uno de los principales problemas es poder garantizar un orden total, sin embargo, Lamport [Lam78b] demuestra que con el uso de los relojes lógicos se puede lograr este orden. Usando la idea que presenta Lamport podemos implementar el protocolo en el cual se garantiza el orden total en los mensajes, sin embargo en el artículo no se prevé la posibilidad de fallas. Para poder soportar fallas en el protocolo de control distribuído se hace necesario considerar bastantes modificaciones a la idea original de Lamport, de hecho Lamport no considera ni siquiera fallas por perdida de mensajes mucho menos fallas totales de los procesadores. Por el lado de diseño e implementación de protocolos de control distribuído hay bastantes alternativas y hay mucho que investigar y explotar en esta dirección.

2.2.2 Criterios que se pueden considerar para poder utilizar un mensaje cuando se tiene un protocolo de control distribuído.

En los protocolos de control centralizado cada nodo que recibe un mensaje compara el número de secuencia del mensaje recibido con el numero de secuencia del mensaje que espera, si estos son iguales entonces se dice que el mensaje recibido es correcto, en caso contrario el receptor pide al nodo coordinador la repetición del dato que esperaba. En este tipo de protocolos el criterio para decidir si el dato es o no correcto es muy sencillo porque cada dato esta etiquetado, sin embargo en protocolos con control distribuído no podemos utilizar este criterio. Veamos algunos criterios que podemos utilizar para poder decidir que un dato tiene el orden correcto para este caso.

Consideremos que tenemos un grupo (G) compuesto por ($n \geq 2$) entidades $\{E_1, E_2, \cdots, E_n\}$ y cada entidad es un autómata finito que tiene 2 tipos de eventos { envió y recepción }.

Vamos a denotar $S_k[m]$ y $R_k[m]$ a los eventos envió y recepción del mensaje (m) por la entidad k.

Vamos a decir que el evento e_1 esta antes que e_2, si e_1 sucedió antes que e_2 en la entidad E_k o si para dos entidades E_k, E_j existe un mensaje m tal que $e_1 = S_k[m]$ y $e_2 = R_j[m]$.

Denotemos a m.DST como un conjunto de entidades destino del mensaje m. Supongamos también que para cada entidad E_k un mensaje m transporta el conocimiento de lo que ha recibido E_k.

A partir de los supuestos y notación anteriores vamos a definir lo que significa que un mensaje pueda ser utilizado por la aplicación.

Definición 2 *Un mensaje m de E_j se dice que es aceptado en E_k si y solamente si todos los mensajes de E_j se reciben antes de recibir el mensaje m.*

Definición 3 *Un mensaje m de E_j se dice que es reconocido en E_k si y solamente si E_k sabe que todas las entidades de m.DST ya han reconocido a m.*

Definición 4 *Cuando el mensaje m es reconocido en E_k, E_k considera que m se ha recibido correctamente por todas las entidades destino de m.*

Que el mensaje sea aceptado quiere decir que el mensaje ha sido recibido pero no necesariamente es válido y que sea reconocido quiere decir que el mensaje es válido y puede ser utilizado por la aplicación.

Este criterio que hemos descrito es uno de tantos que se pueden tomar para poder decidir si el mensaje se puede o no enviar a la aplicación. Independientemente del criterio que se use lo que debemos garantizar es que el orden de los mensajes recibidos en un nodo debe ser el mismo para todos; esto es realmente lo que garantiza que todos los nodos puedan llegar a la misma conclusión. Para mayor referencia sobre este enfoque véase [NT91]

2.3 Observaciones

En conclusión a este capítulo, podemos decir que entre los enfoques que se utilizan para el diseño de protocolos por difusión existe una gran diferencia pero el objetivo es lograr protocolos que sean confiables y eficientes. Por un lado los protocolos centralizados presentan gran posibilidad de implantación y el control es bastante sencillo porque simplemente debemos fijarnos en el procesador que está controlando la difusión. Sin embargo una de las grandes desventajas que presenta es que están destinados a saturarse con el aumento en número de procesadores o con el aumento en la cantidad de mensajes en la red, es decir, es muy probable que se presenten cuellos de botella.

Por otro lado tenemos los protocolos de control distribuído para los cuales no ha habido mucho desarrollo y aún no se han implementado protocolos completamente distribuídos, aunque ha habido grandes avances teóricos y posiblemente pronto aparecerán desarrollos de este tipo de protocolos. Por la misma razón de que no ha habido mucho desarrollo sobre estos protocolos, tampoco ha habido protocolos de control distribuído tolerantes a fallas.

Capítulo 3

Una técnica para la especificación de protocolos.

3.1 Introducción.

En un principio el desarrollo de un sistema se hacia sin planificar o sin tener una visión general de todo el sistema y esto daba como resultado que nunca se desarrollaran sistemas bien estructurados y además poco seguros; al mismo tiempo, en el momento que requería de un cambio era necesario rehacer la mayor parte del sistema.

A partir de estos problemas nace la ingeniería de software y se plantean ciertas técnicas que ayudan al desarrollo de sistemas modulares y confiables. Las diferentes técnicas que se plantean se clasifican como: Técnicas formales y técnicas informales. Las técnicas formales utilizan un lenguaje mas natural y las formales utilizan un lenguaje mas técnico y especializado.

Un aspecto importante relacionado con la especificación es la decisión del lenguaje que se utiliza; muchas especificaciones se escriben en lenguaje natural, sin embargo, en muchos casos es de gran ayuda definir ciertos aspectos del comportamiento de una manera mas formal.

Aunque las especificaciones escritas en lenguaje natural son realmente accesibles a casi cualquier lector, estas presentan ciertas desventajas, por ejemplo, una especificación en lenguaje natural por lo general contiene ambiguedades y es difícil probar completez y consistencia. Otro aspecto que es importante, es que a las especificaciones en lenguaje natural no es posible automatizar su implementación o prueba; porque no siguen un cierto patrón.

Se han desarrollado varios lenguajes de especificación formal para diferentes propósitos y muchos de estos se han usado para la descripción de sistemas distribuídos y protocolos de comunicación, entre las técnicas y herramientas mas comunes tenemos:

1. Máquinas de estados finitos.
2. Gramáticas formales.
3. Redes de Petri.

4. Calculo Algebraico (CCS Milner).

5. Lenguajes de programación de alto nivel.

6. Tipos de datos abstractos.

7. Lógica temporal.

Todas estas herramientas se han combinado y extendido para especificar y probar aspectos que conciernen a un protocolo, como por ejemplo:

- Orden temporal de las interacciones (coordinación de actividades).
- Rango de los posibles valores que intervienen en el protocolo.
- Reglas para interpretar y seleccionar valores de los parámetros de interacción.
- Propiedades de vivacidad, justicia y completez.
- Propiedades de tiempo real.

A finales de los setentas, cuando OSI inicia los trabajos de estandarización algunas gentes reconocen que las especificaciones formales pueden ser útiles en el desarrollo de herramientas para la creación de los estándares de OSI, ya que, los protocolos estándar no deben tener ambiguedades. Se forman algunos grupos dentro de OSI y CCITT interesados en las técnicas de especificación formal y de ahí se desarrollan tres lenguajes para especificaciones formales: **Estelle** [BD87] que está basado en maquinas de estados finitos extendidas, **LOTOS** [TB88] que esta basado en máquinas de estados finitos extendidas y en el Calculo de Sistemas Comunicantes de Milner [Mil80] y SDL [BOC90] también basado en máquinas de estados extendidas.

Como técnica formal para especificar vamos a utilizar las redes de Petri por su expresividad gráfica y su poder de modelado de sistemas que poseen concurrencia, sincronización y no determinismo; es por eso que, en este capítulo sólo hablaremos de las redes de Petri y un poco sobre máquinas de estados finitos.

3.2 Redes de Petri.

Una red de Petri es un modelo formal que me permite describir el flujo de información en un sistema y analizar el control de los sistemas y más aun para sistemas que exhiben actividades asíncronas, concurrencia y paralelismo. Una red de Petri es una gráfica que me modela las propiedades estáticas de un sistema y contiene dos tipos de nodos: círculos que se llaman sitios y barras que se llaman transiciones y están conectados por aristas dirigidas de sitios a barras y de barras a sitios. Si un arco esta dirigido de un nodo i a un nodo j, decimos que i es una entrada a j y j es una salida de i. Observemos por ejemplo en la figura 3.1, el sitio p_1 es una entrada a la transición t_2 mientras que p_2 y p_3 son salidas de la transición t_2.

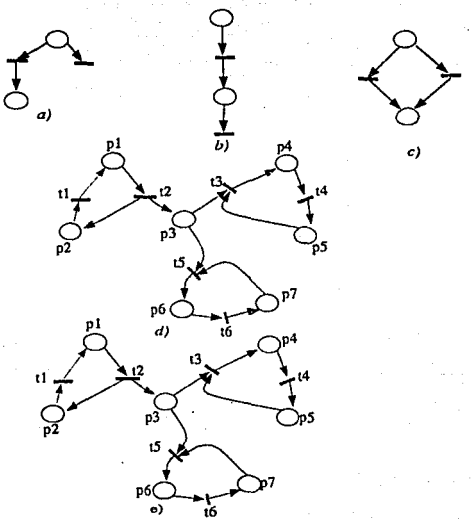

Figura 3.1: Redes de Petri

Aparte de las propiedades estáticas que representa la gráfica de la red de Petri, estas tiene propiedades dinámicas que resultan de la ejecución de la red de Petri. La ejecución de una red de Petri se controla por la posición y movimiento de marcas (denominadas fichas) en los lugares de la red de Petri. Las fichas se indican por puntos negros en los círculos que representan los sitios de la red de Petri. A una red de Petri con fichas se le denomina red de Petri marcada. El movimiento de las fichas se lleva a cabo mediante el disparo de una transición. Una transición se dispara o esta lista para disparar si todas las entradas de la transición tienen por lo menos tantas fichas como el peso de la arista de entrada. Cuando se dispara una transición se mueven las ficha del sitio de entrada a los sitios de salida de la transición. Todos los sitios de entrada de la transición quitan tantas fichas como peso de la arista de entrada y deposita tantas fichas como peso de la arista de salida en cada sitio de salida. La distribución de las fichas en una red de Petri marcada define el estado de la red. Si dos transiciones se encuentran de tal forma que al disparar una de ellas deshabilita la otra, se dice que las transiciones están en conflicto. En este caso las transiciones t_3 y t_5 de la figura 3.2 están en conflicto.

3.3 Modelación mediante el uso de redes de Petri.

En la mayoría de las ciencias experimentales aparecen fenómenos que son imposibles de efectuar ya sea por razones económicas o practicas y esto forza la creación de modelos que simulen el fenómeno. En general, un modelo es la representación (usualmente en términos matemáticos) de lo que es importante estudiar de un objeto. Mediante la manipulación de esta representación obtenemos nuevo conocimiento sin necesidad de costo y riesgos. Se han desarrollado herramientas de modelación y en general son herramientas matemáticas que permiten describir de manera muy natural ciertos fenómenos en algunas ciencias como Física y Química. En el área de Sistemas ha habido muchos modelos gráficos propuestos por autores como herramientas útiles para el análisis de ciertas características peculiares de los sistemas de cómputo, tales como, concurrencia, sincronización, comunicación y cooperación entre subsistemas. Mucho del trabajo en esta área esta relacionada a las ideas originales desarrolladas por C. A. Petri. Dichos modelos gráficos hoy se conocen como redes de Petri.

Las redes de Petri son una herramienta de modelación para una clase de problemas; estos problemas son sistemas de eventos discretos con concurrencia o paralelismo. Las redes de Petri modelan sistemas y en particular dos aspectos de los sistemas: Eventos, condiciones y la relación entre estos. Desde este punto de vista, en algún momento en un sistema se van a cumplir ciertas condiciones y esto va a provocar que ocurran ciertos eventos y esto a la vez provoca un cambio de estado en el sistema dando lugar a la aparición de nuevas condiciones y la desaparición de otras. Formalmente una red de Petri se define como sigue.

Definición 5 *Una red de Petri es un Cuádruplo*

$$N = (P, T, F, W)$$

Donde
P es un conjunto de lugares.

T es un conjunto de transiciones.
F ϵ (PxT) U (TxP) es una relación de flujo.
W : F: → N+ asigna un peso a cada arista.

Ejemplos de redes de Petri se muestran en la figura 3.2

Para poder hacer una descripción mas completa sobre redes de Petri vamos a presentar algunos conceptos básicos.

Definición 6 *Para xϵPUT, el pre-conjunto de x se define como*

$$^{\bullet}x = \{y \epsilon PUT | W(y,x) > 0\},$$

y

$$x^{\bullet} = \{y \epsilon PUT | W(y,x) > 0\}.$$

donde W es la función de peso de la arista que va de y a x.

Definición 7 *N es una red ordinaria si el codomínio de W ⊆ {0,1}*

Definición 8 *Una marcación de una red N = (P,T,F,W) es una función M : N+ → N, es decir, la asignación de un entero no negativo a cada lugar, indicando el número de fichas que contiene el lugar.*

Para $p \epsilon P$, $M(p)$ denota el número de fichas en p.

Definición 9 *Un sistema de red Σ = (P,T,F,W,M0) es una red N = (P,T,F,W) con una marcación inicial M0.*

Definición 10 *Una transición t es habilitada por la marcación M o t ϵ $\Sigma_{hab}(M)$ en el sistema Σ, si para toda p ϵP : M(p) ≥ W(p,t).*

Si t es una transición que es habilitada por la marcación M, entonces t puede disparar produciendo una nueva marcación K tal que $K(p) = M(p) - W(p,t) + W(t,p)$ para todo p ϵP. Esto se denota por $M[t\rangle K$.

Definición 11 *[M0⟩ denota al conjunto de marcaciones más pequeño de Σ y cumple las siguientes propiedades*

1. $M_0 \epsilon [M_0\rangle$ *y*
2. *si K ϵ[M0⟩ y K[t⟩L para alguna t ϵT, entonces L ϵ[M0⟩.*

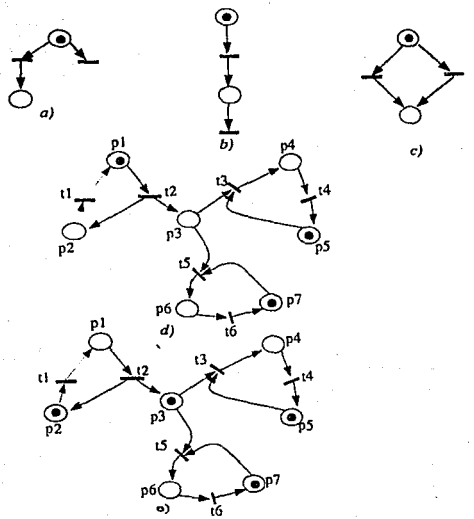

Figura 3.2: Modelos de redes de Petri marcadas

Definición 12 *Una sucesión de eventos de* Σ *es una sucesión de transiciones finita* $\sigma = t_1 t_2 \ldots t_n (n \geq 0)$ *tal que existen marcaciones* $M_1, M_2 \ldots, M_n$ *que satisface*

$$M_{i-1}[t_i\rangle M_i$$

para $i = 1, 2, \ldots, n$.

El conjunto de sucesiones de eventos en un sistema de red Σ se denota por $suc(\Sigma)$.

Definición 13 *Sea* $\Sigma = (P, T, F, W, M_0)$ *un sistema de red.*

1. *Un lugar* $p \epsilon P$ *es n-acotado para* $n \epsilon N aturales$, *si para todo* $M \epsilon [M_0\rangle$, $M(p) \leq n$.
2. Σ *es acotado si existe* n *tal todos los lugares* $p \epsilon P$ *son n-acotados.*
3. Σ *es justo si todos los lugares son 1-acotados.*
4. *Una función de capacidad de un sistema acotado es un mapeo* $c : P \rightarrow N$ *tal que para todo* $p \epsilon P$ *se tiene*

$$c(p) \geq Max(\{M(p) | M \epsilon [M_0\rangle\} U \{W(p,t) | t \epsilon p^*\} U \{W(t,p) | t \epsilon (^*p)\})$$

Notas:

- Si c(p)=0 entonces el lugar p es redundante porque es un lugar sin marca aislado que no contribuye al comportamiento de la red.
- Para un sistema justo, la función de capacidad satisface que cod(c)={ 1 }.

Para mayores referencias obre estos conceptos véase [Pet77, Reu91, BK92]

3.4 Propiedades de las redes de Petri útiles para la modelación.

Una de las propiedades inherente a las redes de Petri es su capacidad de modelar sistemas de control distribuído con múltiples procesos ejecutándose de manera concurrente. Otra propiedad importante es que son asíncronas por naturaleza; es decir, para controlar la secuencia de eventos no necesita de la noción del tiempo; la misma estructura contiene la información para definir la secuencia de eventos del sistema modelado. En las redes de Petri, el orden de los eventos está explícito, sin embargo los eventos que no necesitan estar restringidos a un orden de ocurrencia no se restringen. En realidad, un sistema modelado con una red de Petri se ve como un sucesión discreta de eventos cuyo orden de aparición es una de todas las posibles permitidas por la estructura básica y esto es lo que nos lleva al no determinismo en la ejecución de la red. Si en algún momento más de una transición esta habilitada, entonces algunas de estas pueden disparar y la elección de las que disparen se hace de forma aleatoria; esto es lo que refleja los fenómenos de la vida real, en donde diferentes sucesos se llevan a cabo concurrentemente y el orden de ocurrencia de los eventos no es único. El no determinismo es ventajoso desde el punto de vista de modelado pero introduce demasiada complejidad en el análisis de las redes de Petri.

3.5 Análisis de las redes de Petri.

Quizá una de las preguntas más inmediatas es el porqué modelar con redes de Petri. Originalmente el propósito de las redes de Petri fue meramente descriptivo, dado que las reglas de ejecución se usan para describir un sistema en términos de conceptos simples que nos brindan una forma natural para describir sistemas asincronos con procesos concurrentes. Rápidamente un uso posterior, fue la descripción de Sistemas mediante el uso de redes de Petri y en este caso lo que se hace es analizar el modelo para detectar la presencia de propiedades deseables o indeseables. Hay preguntas analíticas sobre redes de Petri difíciles de resolver; es por eso que se han estudiado subclases de redes de Petri para lograr un análisis más fácil en situaciones especificas. Las propiedades que se deben tratar de probar acerca de un sistema son:

1. Acotamiento: Si cada lugar esta acotado el sistema es acotado, lo cual implica que el espacio de estados es finito y se pueden probar otras propiedades del sistema.

2. Vivacidad: Esta propiedad se refiere al disparo potencial de las transiciones para todas las marcas alcanzables, con esto se garantiza que un sistema es libre de abrazos mortales.

3. Si busca conservación de fichas, ver si se cumple.

4. Reversibilidad si se cumple esta propiedad quiere decir que siempre puedo regresar a la marca inicial después de disparar todas las transiciones por lo menos una vez. Esta propiedad me garantiza que el sistema es vivo y libre de abrazos mortales [MS92].

3.6 Redes de Petri con prioridades

La especificación de prioridades a una red de Petri, nos ofrece una forma de resolver conflictos en el diseño de sistemas concurrentes. En el diseño de sistemas concurrentes puede ser deseable especificar prioridades (la preferencia de la ejecución de un proceso sobre otro). Las prioridades han sido usadas ampliamente por los Sistemas Operativos para forzar un orden preferido de la ejecución de tareas que esperan ser procesadas; también algunos lenguajes de programación como "occam" y "Ada" ofrecen constructores para especificar prioridades; estos constructores se ofrecen como un operador para elegir prioridades en donde se expresa la preferencia que se desea para la ejecución de una sección de programa sobre otra. En estos casos, las prioridades son usadas como una forma natural y conveniente para resolver conflictos entre dos o más actividades listas para ejecutarse al mismo tiempo, durante la evolución de un sistema concurrente.

Una de las razones por las cuales es difícil formalizar prioridades en presencia de concurrencia en redes de Petri se debe a un concepto de redes de Petri que se llama "confusión asimétrica". Básicamente la especificación de prioridades entre un par de acciones puede contradecir la especificación de concurrencia entre otro par de acciones.

Parte del objetivo de esta sección es mostrar que la inclusión de prioridades no cambia el comportamiento del sistema, pero sí nos permite mas libertades en el modelado. Aquí vamos a describir un poco informal el concepto de prioridades, pero trataremos de dejar clara la idea. También vamos

a mostrar que dada la red de Petri Σ y la relación de prioridades ρ definida en las transiciones de la red de Σ, es posible construir una red Σ_ρ que mantiene la concurrencia de Σ y al mismo tiempo no viola las restricciones impuestas por ρ. Se va a suponer que Σ es acotado, ya que las redes de Petri con prioridades tienen más poder computacional que las redes de Petri usuales y por lo tanto mayor complejidad.

Para describir con más precisión el concepto de redes de Petri con prioridades vamos a presentar los conceptos de manera mas formal.

Definición 14 *Un sistema de prioridades es un par* (Σ, ρ) *tal que* $\Sigma = (P, T, F, W, M_0)$ *es un sistema de red y* $\rho \epsilon T x T$ *es una relación denominada sistema de prioridades.*

Notas:

- $(t, u) \epsilon \rho$ se entiende que t tiene menor prioridad que u.
- Se consideran prioridades estáticas.
- No hay restricciones en la relación de prioridad ρ.

Una transición es habilitada en la relación de prioridades ρ por una marcación M si se cumplen los siguientes puntos:

1. La transición t esta habilitada por M.
2. No existen transiciones habilitadas por M que tengan mayor prioridad que t.

Esto expresado de manera formal se ve en la siguiente definición.

Definición 15 *Sea* (Σ, ρ) *un sistema de prioridades y sea* M *una marcación arbitraria de* $\Sigma = (P, T, F, W, M_0)$. *Una transición* $t \epsilon T$ *es* ρ *– habilitada en* M, *denotado por* $t\ chab_{\Sigma,\rho}(M)$, *si*

$$t chab_\Sigma(M)$$

y

$$\rho(t) \cap hab_\Sigma(M) = \emptyset.$$

Donde $hab_\Sigma(M)$ *indica que* t *el sistema* Σ *esta habilitada por* M.

Si t es ρ *– habilitado* y $M[t\rangle K$ entonces denotamos esto por $M[t\rangle_\rho K$.

Una sucesión de eventos de (Σ, ρ) es una sucesión de transiciones $\sigma = t_1 t_2 \ldots t_n (n \geq 0)$ tal que existen marcaciones $M_1, M_2, \ldots, M_n$ que satisfacen $M_{i-1}[t_i\rangle_\rho Mi$, para $i = 1, 2, \ldots, n$.

El conjunto de sucesiones de eventos de (Σ, ρ) se denota por $suc(\Sigma, \rho)$. Un ejemplo de un sistema de prioridades se muestra en la figura 3.3.

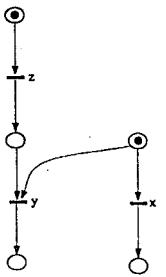

Figura 3.3: Gráfica de un sistema de prioridades

Teorema 1 *Dado el sistema de prioridades* (Σ, ρ) *se puede construir una red de Petri* Σ_ρ *que satisface lo siguiente:*
Σ_ρ *proporciona una semántica con un orden parcial del sistema de prioridades representado por* (Σ, ρ) *el cual mantiene tanta concurrencia presente en* Σ *como es posible.*

El que se pueda mantener tanta concurrencia como sea posible quiere decir que la sucesión de pasos que ejecuta el sistema pertenece al conjunto más grande en donde se mantiene la concurrencia del sistema original y no contradice las restricciones impuestas por el sistema de prioridades.

La construcción de esta red se lleva a cabo dividiendo las transiciones y agregando nuevos lugares y aristas. La forma como se agregan estos lugares y aristas es usando la noción de lugar complemento definido para redes de Petri. La demostración precisa de este teorema se encuentra en [BK92].

3.7 Redes de Petri estocásticas generalizadas.

Las redes de Petri han sido objeto de investigación tratando de obtener un entendimiento completo de las propiedades básicas del modelo e investigando su aplicabilidad a la representación de sistemas reales. Pensando en aplicaciones prácticas, muchos autores han creído conveniente extender o modificar las definiciones del modelo básico para obtener una herramienta de modelado más conveniente. Como caso concreto ya vimos en la sección anterior las redes de Petri con prioridades.

Otra extensión que va a ser considerada en este capítulo es un modelo en donde se incluye el tiempo que va a representar el tiempo de ejecución de una operación.

Una primer representación consiste en la asignación de un tiempo fijo a cada transición para especificar el tiempo de retardo entre la habilitación y el disparo de la transición; los autores

han introducido el tiempo de disparo de una transición de diferentes formas; algunos le asignan a una transición un tiempo mínimo y un máximo de disparo. A estas redes se les denomina **redes de Petri generalizadas** y representan una herramienta de modelado más real; con estos modelos podemos representar el comportamiento del sistema de una forma mucho más cercana a la realidad.

Otro modelo mucho más general consiste en asignarle una función de probabilidad al tiempo de disparo de una transición, a este modelo se le conoce como **redes de Petri estocásticas**; en particular M.K. Molly[Mol79] asoció una función de distribución exponencial al tiempo de disparo de una transición y la definición del modelo formal es la siguiente:

Definición 16 *Una red de Petri estocástica es una quin-tupla*

$$RPS = (P, T, F, M_0, \Lambda)$$

donde
P *es un conjunto de lugares.*
T *es un conjunto de transiciones.*
$F \subset (PxT) \cup (TxP)$ *es una relación de flujo.*
$M_0 = (m_{01}, m_{02}, \cdots, m_{0m})$ *una marcación inicial y*
$\Lambda = (\lambda_1, \lambda_2, \cdots, \lambda_n)$ *es un arreglo de velocidades de disparo asociadas a las transiciones*

La velocidad de disparo asociada a cada transición especifica el tiempo promedio que debe transcurrir para que la transición sea disparada. Es tiempo de retardo es una variable aleatoria con función de densidad de probabilidad exponencial. El parámetro de la función de densidad asociado a la transición t_i es la velocidad de disparo de la transición. Esta velocidad puede ser dependiente de la marcación y se va a escribir como $\lambda_i(M_j)$. El tiempo promedio de disparo de la transición t_i en la marcación M_j es $[\lambda_i(M_j)]^{-1}$.

Como el mínimo de dos variables aleatorias con función de densidad de probabilidad la exponencial negativa, que tienen como parámetros μ_1 y μ_2 también se distribuye como una exponencial con parámetro $(\mu_1 + \mu_2)$, entonces el tiempo de permanencia en la marcación M_j es una variable con función de densidad exponencial negativa con media

$$[\sum_{i:t_i \in H(M_i)} \lambda_i(M_j)]^{-1}.$$

donde
$H(_i)$ es el conjunto de todas las transiciones habilitadas por la marcación M_j.

El hecho de que los tiempos de disparo de las transiciones tengan una función de densidad exponencial permite escribir expresiones simples para la probabilidad de que una transición t_k tenga un disparo mínimo con respecto al resto de las transiciones y determine el cambio de marcación mediante el disparo; está dada por

$$P(t_k \| M_j) = \frac{\lambda_k(M_j)}{\sum_{i:t_i \in H(M_j)} \lambda_i(M_j)}$$

donde

$$t_k \epsilon H(M_j).$$

La propiedad de perdida de memoria de las funciones de densidad de probabilidad exponencial negativas permiten hacer diferentes y útiles interpretaciones sobre los modelos de redes de Petri estocásticas. Por ejemplo cuando un cambio de marcación habilita una transición que no se habilitó en el instante anterior esta transición muestrea una instancia de tiempo de disparo de su función de densidad y comienza a contar tiempo regresivo desde el valor muestreado. Mientras la transición es habilitada el tiempo decrese a velocidad constante. Si la transición es deshabilitada por el disparo de alguna otra transición en conflicto, el contador de tiempo se detiene y continua en el momento que la transición vuelve a ser habilitada. Cuando el contador llega a cero la transición dispara.

Una utilidad importante de la interpretación anterior es que podemos asociar actividades a las transiciones, estas pueden iniciarse e interrumpirse durante la dinámica del modelo y cuando se termina la actividad. ésta induce un cambio de estado.

Una observación importante es que el conjunto de estados alcanzables de una red de Petri estocástica es idéntico al de la red de Petri pura. La identidad de los conjuntos de alcanzabilidad implica que las propiedades estructurales obtenidas para la red de Petri son también validas para las redes de Petri estocásticas.

Otra observación es que el diagrama de estados de transición de la cadena de Markov con tiempo continuo correspondiente a la red de Petri estocástica se obtiene de la construcción de la gráfica de estados alcanzables etiquetando las aristas con la velocidad de disparo de la transición que produce el cambio de marcación.

Esto quiere decir que obtener el generador infinitesimal de la cadena de Markov con tiempo continuo tiene la misma complejidad que generar el conjunto de estados alcanzables de la red de Petri. La solución del modelo en el estado estable se obtiene resolviendo los sistemas de ecuaciones lineales

$$\pi Q = 0$$

y

$$\sum_i \pi_i = 1.$$

donde π_i denota la probabilidad de que la marcación M_i este en estado estable, y π es la distribución de probabilidad en equilibrio sobre el conjunto de las marcaciones alcanzables.

Una vez que tenemos la distribución de probabilidades en el estado estable sobre todas las marcaciones alcanzables, algunos de los parámetros importantes que podemos calcular son:

- *La probabilidad de un evento* acerca de marcaciones en ciertos lugares, por ejemplo, la probabilidad de que no haya fichas en un conjunto de lugares, la probabilidad de al menos una ficha en un lugar mientras otros estén vacíos, etc.

  Esta probabilidad se puede calcular asignando probabilidades a todas las marcaciones en la cual la condición del evento es verdadero. Por ejemplo, la probabilidad de un evento A

definido bajo una condición que es verdadera en un conjunto de marcaciones $M_i \epsilon M$ se obtiene como

$$P\{A\} = \sum_{i:M_i \epsilon M} \pi_i$$

- La probabilidad de que haya un número de fichas en cierto lugar, digamos en p_i se obtiene calculando las probabilidades individuales del evento **" el lugar p_i contenga k fichas".**
- El número promedio de fichas en un lugar se puede calcular a partir de la probabilidad que haya fichas en este lugar.
- La frecuencia de disparo de una transición se calcula como la suma ponderada de las velocidades de transición

$$f_i = \sum_{i:t_i \epsilon H(M_i)} \lambda_i(M_i)\pi_i$$

donde f_i es la frecuencia de disparo de la transición t_i, $H(M_i)$ es el conjunto de transiciones habilitadas en M_i y $\lambda_i(M_i)$ es la velocidad de disparo de t_i en la marcación M_i.

- El tiempo promedio que transcurre para que una ficha recorra una subred en condiciones estables se puede calcular usando la formula de Little [AMC91]

$$E[T] = \frac{E[N]}{E[\gamma]}.$$

Donde E[T] es el tiempo promedio transcurrido, E[N] es el promedio de fichas que atraviesan la subred y $E[\gamma]$ es la cantidad promedio de fichas que entran en la subred. Así como los puntos anteriores podemos encontrar algunos otros parámetros a evaluar; pero vamos a ver otro modelo que es una extensión mas de las redes de Petri y que nos permite modelar con mas facilidad y con más claridad; aunque en realidad no ganamos mas propiedades estructurales del sistema. Este modelo es la unión de las redes de Petri estocásticas y las redes de Petri generalizadas y se definen como sigue.

Definición 17 *Una red de Petri estocástica generalizada es*

$$RPEG = (P, T, \wp, F, H, M_0, W)$$

Donde
$(P, T, F, M_0, W$ son los elementos básicos de las redes de Petri.
$\wp$ es una función que asigna prioridades a transiciones. Esta función asocia prioridad 0 a las transiciones con tiempo y prioridad(≥ 1) a las transiciones inmediatas.
y $W = (w_1, w_2, \cdots, w_n)$ es una arreglo cuyas entradas w_i son:

1. *El parámetro de la función de probabilidad exponencial negativa asociada a la transición t_i.*
2. *Es un peso que se utiliza para calcular la probabilidad de disparo de las transiciones inmediatas si t_i es una transición inmediata.*

La interpretación del modelo de las redes de Petri generalizadas es muy similar al del modelo de las redes de Petri estocásticas con los cambios necesarios para incluir las transiciones inmediatas.

Cuando se obtiene una marcación, primero es necesario averiguar si habilita sólo transiciones con tiempo o al menos una transición inmediata. Las marcaciones del primer tipo se llaman marcaciones tangibles y las otras transiciones se llaman intangibles.

En el caso de transiciones tangibles, los "timers" de las transiciones con tiempo habilitadas inician su decremento o hacen un "reset" y luego inician el decremento hasta que alguna transición con tiempo dispara exactamente como las redes de Petri estocásticas.

En el caso de las transiciones no tangibles, la selección de la transición a disparar no se puede basar en la descripción temporal, ya que todas las transiciones inmediatas disparan en tiempo cero. La elección se basa en los pesos y las prioridades. Primero se encuentra el conjunto de transiciones con más alto nivel de prioridad y si tiene mas de una transición, entonces la elección se efectúa acorde a la siguiente expresión

$$P\{t_k\} = \frac{W_k}{\sum_{t_i \in H(M)} W_i}$$

Donde H(M) es el conjunto de transiciones inmediatas habilitadas por la marcación M; es decir, las transiciones con el más alto nivel de prioridades.

Notemos que la semántica de las redes de Petri estocásticas generalizadas también supone que las transiciones se disparan una a la vez, aun en las marcaciones no tangibles que comprenden transiciones habilitadas inmediatas que no están en conflicto.

La equivalencia de este comportamiento con el que resulta del disparo simultaneo de algunas transiciones inmediatas en el modelo, pueden ser aprovechadas para reducir la complejidad de los algoritmos [MB84, AMC91].

La presencia de arcos inhibidores y prioridades reduce el número de estados alcanzables con respecto al modelo básico de las redes de Petri, este conjunto en los RPEG es igual al conjunto que se alcanza incluyendo prioridades y arcos inhibidores.

El análisis del modelo de las RPEG requiere de la solución de un sistema de ecuaciones lineales que tiene tantas ecuaciones como el número de marcaciones tangibles alcanzables.

El generador infinitesimal de la cadena de Markov de tiempo continuo asociado al modelo de la RPEG se deriva de la contracción de la gráfica de alcance etiquetada con las velocidades o pesos de las transiciones que generan el cambio de marcación.

3.8 Subclases de redes de Petri.

El éxito de un modelo se debe a dos factores: su poder de modelado y su poder de decisión. El poder de modelado se refiere a la habilidad para representar correctamente el sistema que se va a modelar o qué tanto se apega la representación al modelo. El poder de decisión se refiere a la habilidad para analizar el modelo y determinar propiedades del sistema modelado.

Generalmente estos dos factores no se pueden tener al mismo tiempo, por ejemplo, las máquinas de estados finitos tienen un conjunto de estados alcanzable finito y es posible contestar cualquier

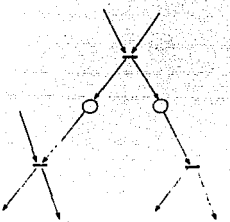

Figura 3.4: Redes de Petri que son gráficas marcadas

pregunta acerca de un modelo con estas, sin embargo, la clase de sistemas que pueden ser modelados es bastante limitado. esto implica que tiene bajo poder de modelado. Por otro lado, las máquinas de Turing tienen un buen poder de modelado, pero muchas preguntas generales son problemas indecidibles y por ende tienen un bajo poder de decisión. Por lo general, cuando se aumenta el poder de modelado, la capacidad de determinar propiedades del modelo, generalmente decrece.

Como sabemos, los problemas de decisión de las redes de Petri son equivalentes al problema de alcanzabilidad que se sabe que es un problema decidible, pero a pesar de eso es un problema muy difícil de resolver. Desde el punto de vista práctico, las redes de Petri pueden ser difícil de analizar, por eso se han definido diferentes subclases de redes de Petri en las cuales se espera tener poder de decisión y poder de modelado para ciertos propósitos prácticos. Las subclases más usualmente consideradas son:

- gráficas marcadas y
- máquinas de estados.

3.8.1 Gráficas marcadas o gráficas de sincronización.

Las gráficas marcadas son una subclase de redes de Petri en la que el número de transiciones de entrada y salida a cada lugar es uno. Para este tipo de modelo, las redes de Petri permitidas se muestran en la figura 3.4.

Con esta clase de gráficas podemos modelar sistemas para ordenar actividades como gráficas de PERT(Program Evaluation and Review Technique), aunque es más general porque permite el reciclamiento y puede contener muchas fichas.

Con las gráficas marcadas podemos modelar diferentes clases de problemas y lo importante del modelo es que ya existen algoritmos para mostrar si la gráfica marcada es viva, acotada y también para resolver el problema de alcanzabilidad; esto quiere decir que las gráficas marcadas tienen un gran poder de decisión. Estas tienen limitaciones de modelado, porque solo pueden modelar sistemas cuyo control de flujo no contenga ramas, sin embargo, se pueden modelar actividades paralelas.

Desde el punto de vista formal, las gráficas marcadas se definen de la siguiente manera:

Definición 18 *Una gráfica marcada es una gráfica dirigida G(V,E)*
Donde
V es un conjunto de vértices.
E es un conjunto de aristas.

Definición 19 *Una marcación es una función que asigna a cada arista e∈E un entero positivo.*

Definición 20 *En este caso decimos que un vértice es disparable si el número de fichas en cada arista de entrada es positivo.*

Definición 21 *El disparo de un vértice consiste en la eliminación de una ficha de cada arista de entrada y agrega una ficha a cada arista de salida.*

Definición 22 *Una marcación es justa si no hay aristas a las que se les asigne más de una ficha y no existe una sucesión de disparos que asigne dos o mas fichas a una arista.*

De la definición de gráfica marcada tenemos las siguientes propiedades:

Propiedad 1 *El número de fichas de un circuito dirigido no cambia por el disparo de un vértice; es decir, el número de fichas en un circuito dirigido es invariante al disparo de los nodos.*

Una de las propiedades importantes que se trata de encontrar en las redes de Petri, es si una marca es o no viva; en este caso, una marca es viva si todo vértice es disparable o puede ser disparable mediante una sucesión de disparos: la propiedad que me permite verificar esta propiedad es la siguiente:

Propiedad 2 *Una marcación es viva si y solo si el número de fichas de todos los ciclos dirigidos es positivo.*

Propiedad 3 *Una marcación que es viva permanece viva después de disparar.*

Otra de las propiedades importantes que necesitamos conocer de una red de Petri es si es o no acotada; la forma de deducir esta propiedad en las gráficas marcadas se muestra en la siguiente propiedad.

Propiedad 4 *Una marcación viva es justa si y solamente si toda arista en la gráfica pertenece a un ciclo dirigido con un número de fichas igual a uno.*

de esta y las propiedades anteriores deducimos las siguiente propiedad.

Propiedad 5 *Si una gráfica tiene una marcación viva y segura, entonces para cualquier arista de la gráfica se puede encontrar un ciclo que pase a través de la arista.*

Entre otras propiedades que podemos encontrar sobre gráficas vivas y seguras tenemos.

Propiedad 6 *Si la gráfica que resulta de la eliminación de direcciones de la gráfica $G(V,E)$ es conexa y si a G se le puede asignar una marcación viva y segura entonces G es fuertemente conexa.*

Propiedad 7 *Para toda gráfica finita, dirigida y fuertemente conexa existe una marcación viva y segura.*

Otra de las propiedades que se tratan de buscar en una red de Petri es la reversibilidad; en las gráficas marcadas esta propiedad se encuentra si se cumple la siguiente.

Propiedad 8 *Si $\dot{\sigma}$ es una sucesión de disparos, para una gráfica cuya gráfica sin direcciones es conexa y la sucesión lleva a la marca inicial M_0, entonces todos los vértices han disparado igual número de veces.*

Para mayores referencias sobre este modelo véase [MS92, FCP71]

3.8.2 Máquinas de estados

Otra de las subclases de redes de Petri son las máquinas de estados y es una red de Petri tal que cada transición tiene una sola entrada y una sola salida, es decir,

Definición 23 *Una máquina de estados es una red de Petri tal que para toda $t \in T$ tal que $|{}^\bullet t| = 1$ y $|t^\bullet| = 1$.*

Definición 24 *Dado una red de Petri $N = (P,T,F,W)$*
Una fuente o abrazo mortal estructural es un subconjunto de lugares tal que las transiciones de entrada pertenecen a las transiciones de salida.

Una trampa es un subconjunto de lugares tal que las transiciones de salida pertenecen a las transiciones de entrada.

Las máquinas de estados permiten el modelado de decisiones (conflictos) y la reentrancia. Notemos que este concepto visto como una subclase de red es más general que los clásicos diagramas de estados, porque en este caso, estos pueden tener más de una ficha. Con estas máquinas, entre las propiedades de los sistemas que podemos modelar encontramos:

- No determinismo.
- Se pueden modelar sistemas finitos.

De los modelos creados con esta clase podemos probar las siguientes propiedades:

Propiedad 9 *Una máquina de estados es viva con M_0 si y solamente si la gráfica es fuertemente conexa y por lo menos M_0 tiene una ficha.*

Claramente para encontrar si una máquina de estados es viva nos toma tiempo exponencial.

Propiedad 10 *Una máquina de estados es consistente(conserva las fichas) si y solamente si la grájica es fuertemente conexa.*

Propiedad 11 *Las Máquinas de estados son estructuralmente acotadas y por lo tanto estructuralmente vivas si y solamente si son fuertemente conexas.*

Estas propiedades también se pueden decidir en tiempo polinomial.

Una propiedad que es válida tanto para máquinas de estados y gráficas marcadas es la siguiente.

Propiedad 12 *Sea $\Sigma = (P, T, F, W, M_0)$ un sistema de red ya sea (gráfica marcada o máquina de estados) viva y acotada. Las siguientes aseveraciones son equivalentes.*

1. *M es alcanzable desde M_0 (i.e. existe σ tal que $M_0[\sigma\rangle M$).*
2. *$M = M_0 + C\vec{p}, \vec{p} \geq 0$ donde $M \epsilon N^n$ marca todas las trampas de mínimas.*
3. *$B^T M = B^T M_0$ donde B es una base para los anuladores izquierdos (i.e $B^T C = 0$) y $M \epsilon N^n$ marca todas las trampas mínimas).*

En general por ser finitas, tienen un gran poder de decisión, pero son de uso limitado en la modelación de sistemas; en este caso no se pueden modelar sistemas que tengan sincronización de procesos o compartición de recursos. Para mas información sobre este tema véase [MS92]

Capítulo 4

Herramientas para prueba de protocolos.

En este capítulo presentamos diferentes formas de analizar la correctez de una especificación. Sabemos que una especificación bien estructurada facilita la verificación. La validación de protocolos de comunicación tiene diferentes aspectos de importancia práctica, tienen diferente naturaleza y son tratados por diferentes métodos.

- Si se trata de probar que el protocolo satisface los requerimientos del usuario, la mejor forma de validarlo es que el usuario pruebe el modelo simulado o un prototipo.
- Si el objetivo es probar que el protocolo ofrece el servicio especificado entonces se pueden investigar propiedades abstractas antes de efectuar la implantación de la especificación.
- Otro objetivo puede ser probar la implantación de una parte del protocolo (entidad de protocolo). El propósito puede ser probar que la entidad de protocolo implementada satisface la especificación o que puede interoperar con las otras implementaciones (prueba de interoperabilidad).

Cualquier método que incremente la confianza de la correctez de una especificación, puede ser usado como método de validación, por ejemplo como métodos de validación también podemos considerar los siguientes:

- En algunos casos, el único método disponible es implementar el protocolo y probar la implementación.
- Algunas veces la simulación es el mejor método en el sentido de que este requiere menor esfuerzo y facilita la validación en el proceso de diseño.

En este capítulo, vamos a mencionar diferentes métodos para la validación de protocolos, pero sólo vamos a profundizar mas sobre los métodos formales y en particular en una técnica de validación de protocolos especificados con redes de Petri.

4.1 Simuladores de redes de Petri.

Los simuladores o métodos por simulación se llaman métodos dinámicos y recorren probando el modelo de la red bajo ciertas convenciones. En este caso se pueden detectar algunos errores; si no detectamos problemas durante el proceso de simulación nos dará cierta confianza acerca de la validez del modelo. Aunque en general, los métodos de simulación no prueban propiedades aunque son de gran ayuda para entender el sistema modelado. En particular, los métodos de simulación son bastante útiles cuando se asocia tiempo a la evolución de la red o cuando queremos saber la respuesta del sistema descrito en un cierto ambiente que también esta especificado en la simulación. Los simuladores sólo validan ciertos caminos seleccionados dentro de todos las posibles ejecuciones; sin embargo se pueden aplicar a especificaciones de tamaño real.

4.2 Técnicas formales.

Desde el punto de vista formal, en general un sistema se puede ver como una tripleta

$$< S, A, \Sigma >$$

Donde
S → es un conjunto de estados.
A → un conjunto de acciones.
Σ → un conjunto de comportamientos en $SxAxS$ de la forma

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots \tag{4.1}$$

para $s_i \epsilon S$ y $a_i \epsilon A$

Para dar una descripción de un sistema, se deben definir el conjunto de estados S, las acciones A y los comportamientos Σ.

Hay dos enfoques para describir el conjunto Σ, un enfoque constructivo y un enfoque axiomático. En el enfoque constructivo, a Σ se le describe mediante un programa, donde a Σ se le define como el conjunto de todos los posibles comportamientos que se obtienen por la ejecución del programa. El programa se puede escribir en un lenguaje de programación convencional o en términos de un método formal, tal como redes de Petri o máquinas de estados finitos.

En el enfoque axiomático, a Σ se le describe mediante un conjunto de axiomas; en este caso Σ se define como el conjunto de todas las secuencias de la forma 4.1 que satisfacen los axiomas.

Las acciones se pueden escribir en un sistema formal tal como lógica temporal o en un una notación matemática menos formal.

Descripciones axiomáticas nos llevan directamente a un método de razonamiento. Si S es un conjunto de axiomas que describen Σ y C es una propiedad expresada con el mismo sistema formal que Σ, entonces el sistema satisface C si y solo si la formula $S \vdash C$ es válida. Para el lector interesado en técnicas formales y el enfoque axiomático consultese [AJSn88, R.90, LL90].

Dependiendo de la herramienta que se utilice para especificar el sistema, esta restringe a probar ciertas propiedades; es decir, no hay herramienta que permita probar todas las propiedades

importantes que posee un sistema, además cada técnica tiene sus propios métodos de prueba. En particular, si las especificaciones de Σ se efectúan con máquinas de estados finitos o con redes de Petri, una forma de probar ciertas propiedades del sistema es mediante el análisis de alcanzabilidad, que se basa en el espacio de estados alcanzables; es decir, se basa en el análisis del conjunto de todos los posibles estados a los que puede llegar el sistema. En este caso nos vamos a restringir a describir una forma de validar un modelo especificado con redes de Petri.

4.2.1 Gráfica de estados alcanzables de redes de Petri.

En esta sección vamos a describir el uso de la teoría de gráficas para probar la correctez de un modelo especificado mediante el uso de redes de Petri. En primer lugar vamos a ver como transformamos un modelo de redes de Petri a una gráfica y una ves que tengamos la gráfica, veremos cuales son las propiedades que podemos describir acerca del modelo.

El método que vamos a utilizar para analizar las redes de Petri es un método por enumeración, el cual se basa en la construcción de una gráfica denominada **gráfica de estados alcanzables**. Esta gráfica representa las marcaciones y el disparo de las transiciones. Si el sistema de red es acotado, la gráfica de estados alcanzables es acotada y podemos probar diferentes propiedades cualitativas; pero si el sistema de red no es acotado la gráfica es infinita y por lo tanto imposible de construir.

Construcción de la gráfica de estados alcanzables.

Antes de describir el proceso para construir la gráfica de estados alcanzables veamos algunas definiciones.

Definición 25 *Decimos que una marcación es alcanzable en un sistema de red* $< N, M_0 >$. *si existe una sucesión* σ *aplicable a* M_0 *tal que* $M_0[\sigma > M$.

Si a partir de la marcación $< N, M_0 >$ podemos encontrar todas las marcaciones alcanzables denotadas por $A(N, M_0)$ entonces $M \in A(N, M_0)$.

Una de las principales limitaciones de este método es su complejidad computacional denominado problema de explosión de estados. El número de marcaciones puede ser exponencial con respecto al tamaño de la red. Sin embargo el análisis por alcanzabilidad a sido uno de los métodos mas efectivos para verificar la correctez de protocolos de comunicación basados en modelos de transición de estados. Debido a su efectividad y fácil automatización, se han construido bastantes herramientas para verificación de protocolos basados en el método de análisis por alcanzabilidad. Aunque es un método muy restringido se han propuesto diferentes estrategias [FLL88b] para evitar la intratabilidad del problema en el momento de construir las herramientas.

Definición 26 *La gráfica de alcanzabilidad asociada al sistema* $< N, M_0 >$ *es una gráfica* $EA(N, M_0)$ *en la cual cada nodo representa una marcación alcanzable desde* M_0 *y cada arista representa el disparo de una transición. Existe una arista etiquetada como* t_k *que va del nodo que representa al estado* M_i *al que representa el estado* M_j *si y solamente si estando en* M_i *el disparo de* t_k *nos lleva al estado* M_j; *es decir,* $M_i[t_k > M_j$.

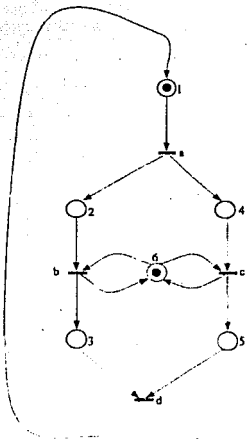

Figura 4.1: Gráfica de un modelo en redes de Petri.

Si la gráfica marcada es acotada, el proceso de construcción termina cuando se han explorado todos los posibles disparos de las marcaciones alcanzables.

Consideremos el sistema de red de la figura 4.1, su gráfica de estados se muestra en la figura 4.2.

Entre las propiedades que podemos describir del sistema, tenemos las siguientes.

Propiedad 13 *Un sistema de red es reversible si y solamente si su gráfica de estados alcanzable es fuertemente conexa.*

La demostración es muy sencilla y lo único que estamos diciendo es que si el sistema es reversible, cualquier transición puede ser disparada una y otra vez si y solamente si la transición es disparable al menos una ves desde la marcación inicial. Esto es cierto porque la marcación inicial es siempre

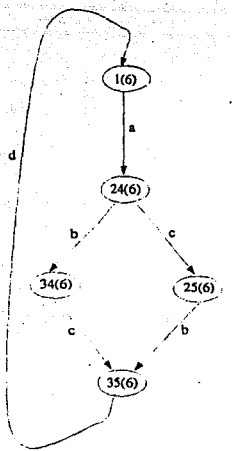

Figura 4.2: Gráfica de estados alcanzables.

alcanzable y la transición se puede disparar nuevamente. Si el sistema no es reversible pero es acotado, la propiedad de vivacidad es la siguiente.

Propiedad 14 *Sea $< N, M_0 >$ un sistema acotado. La transición* t *es viva en $< N, M_0 >$ si y solamente si* t *etiqueta al menos una arista de todas las componentes fuertemente conexas de la gráfica de estados alcanzables que contienen su cerradura transitiva.*

Como los sistemas pueden ser no acotados, la construcción de la gráfica de alcanzabilidad puede no terminar. Para evitar que se continúe construyendo la gráfica de estados alcanzables tenemos la siguiente condición de abandono.

Propiedad 15 *El lugar* p *es acotado en $< N, M_0 >$ si y solamente si no existe un M_j alcanzable desde M_i tal que $M_j \geq M_i$ y $M_j(p) > M_i(p)$. Por lo tanto el sistema $< N, M_0 >$ es no acotado si y solamente si existe un M_j alcanzable desde M_i tal que $M_j > M_i$.*

Esta técnica basada en la gráfica de estados alcanzables es muy simple desde el punto de vista conceptual, pero como ya mencionamos, es difícil por su complejidad computacional. Otro de los problemas que debemos observar es que la gráfica de estados alcanzables se construye a partir de una marcación inicial y si hay un cambio en el número de recursos o la estructura de la red se debe reconstruir nuevamente toda la gráfica. Para mayor referencia sobre esta técnica consultese [FLL88a, MS92] y para mayores referencias sobre algoritmos para gráficas consultese [Eve79, Gib85].

Como ya hemos visto en los capítulos anteriores las redes de Petri son una herramienta formal muy poderosa para el modelado y validarión de protocolos o sistemas en general; pero algo que es muy importante evaluar es la eficiencia de un modelo o del sistema que estamos modelando y se utiliza el principio descrito en la sección anterior.

4.2.2 Reducción de sistemas de red.

La técnica por alcanzabilidad basada en la gráfica de estados alcanzables es completa para sistemas acotados, pero en la práctica, su complejidad computacional limita su aplicabilidad. La reducción de sistemas de red es una técnica de análisis diferente que permite examinar los modelos de redes, transformando su estructura y su marcación inicial. Este enfoque se basa en la definición de un conjunto de reglas de reducción, de tal forma que el modelo transformado preserva el conjunto de propiedades originales del modelo (vivacidad, acotamiento, reversibilidad, etc.).

El proceso de transformación es iterativo y las reglas de transformación (que preservan las propiedades) se aplican hasta que el sistema es irreducible. El sistema reducido puede ser tan simple que las propiedades a estudiar son triviales. También habrá casos en los que el sistema reducido no es tan simple, en cuyo caso necesitamos otra técnica para probar las propiedades del sistema; por lo cual este método no es un método independiente, sino mas bien es complementario.

Para este método la figura 4.3 muestra gráficamente las reglas de transformación; pero el lector interesado en comprobar y ampliar su conocimiento sobre esta técnica lo invitamos a consultar [MS92].

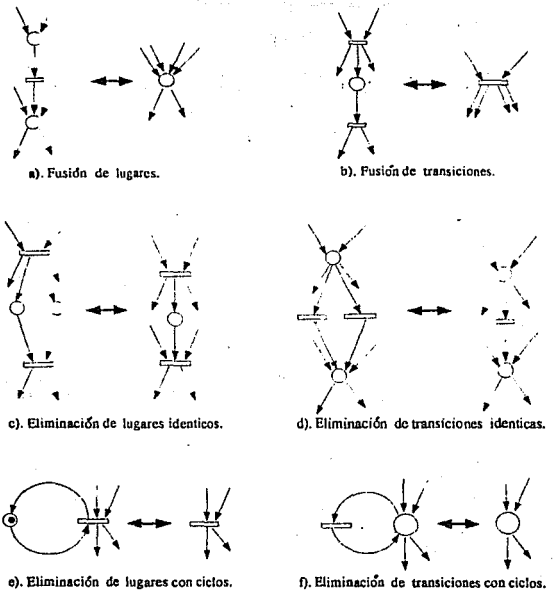

Figura 4.3: Técnicas de transformación de redes de Petri.

4.3 Otras herramientas.

Han aparecido diferentes métodos para la verificación de protocolos, pero otros de los métodos formales mas conocidos tenemos a: las máquinas de estados finitos y al calculo de sistemas comunicantes.

4.3.1 Máquinas de estados finitos.

Uno de los métodos para prueba y especificación de protocolos son las máquinas de estados finitos, que se definen como una cuádrupla

$$(S, i, E, T)$$

Donde
S → Es un conjunto de estados.
i → Es un estado inicial.
E → Es un conjunto de eventos (acciones).
T → Es un conjunto de transiciones definidas en $SxExS$.

Las máquinas de estados finitos han sido una de las técnicas formales mas usadas para la especificación y prueba de sistemas; se han hecho algunas extensiones para especificar sistemas concurrentes. Para mayores referencias sobre esta herramienta consultese [PEH90, PER90, BU91].

4.3.2 Algebras de procesos.

Otra de las técnicas, usualmente referida como álgebra de procesos también ha sido útil, tanto como método clásico de formalización como una base para analizar ciertas propiedades de los sistemas. Una de estas teorías, el álgebra de procesos comunicantes CCS [Mil80] ha jugado un papel importante en el desarrollo de lenguajes formales para los protocolos de OSI y hasta ahora se han reportado muchas aplicaciones de CCS para verificar aspectos de sincronización de protocolos reales.

Capítulo 5

Arquitectura y herramientas de implantación.

Para ubicarnos dentro del contexto en el cual se implementa el sistema vamos a definir y aclarar ada término acerca de las redes locales, topologías, arquitecturas y protocolos.

El sistema esta montado sobre una red local con una topología de tipo bus conectada por cable coaxial y la arquitectura y protocolo que usa para accesar el medio es el definido en el estándar IEEE 802.3.

En primer lugar, el sistema esta montado sobre una red local, la cual se define como una red de comunicaciones que permite la interconexión de una variedad de dispositivos que comuniran datos dentro de una área pequeña.

Como ya se vio en el capítulo 1, la topología es la forma como están conectadas las máquinas que pertenecen a la red; en este caso la topología en la cual se implementa el sistema es de tipo bus como se muestra en la figura 5.1.

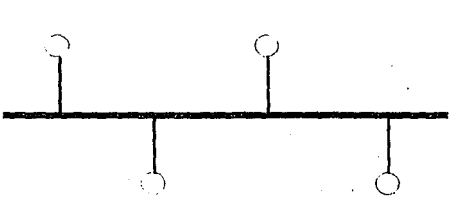

Figura 5.1: Topología de tipo Bus

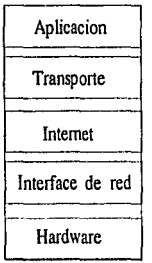

Figura 5.2: Arquitectura del Departamento de Defensa de E.U.

5.1 Arquitectura

Como todos los nodos en el bus comparten un medio de trasmisión común, solo una estación puede trasmitir a la vez, por lo tanto se requiere de una mecanismo para controlar el acceso en donde se determine la máquina que va a trasmitir. En este caso, el protocolo que utiliza es el (Carrier Sense Multiple Access with Colition Detection(CSMA/CD)), definido en el estándar IEEE 802.3; en la parte superior del nivel de enlace usualmente conocido como (Logical Link Control) se encuentran las funciones que se encargan de aceptar trasmisiones y liberar recepciones a las estaciones de trabajo conectadas. Entre estas funciones encontramos:

- Proveer puntos de acceso al servicio; estos puntos de acceso son la interfase con los demás niveles.
- En trasmisiones, ensamblar datos en bloques que contenga direcciones y campos para detección de error.
- Manipular la comunicación en el enlace.

En la figura 5.1, se muestra la arquitectura de la red local sobre la cual esta implementado el sistema, pero solo abarca los niveles físico y de enlace del modelo OSI; En realidad, la arquitectura global en la cual se implementa es la establecida por el (Departamento de Defensa de los Estados Unidos (DOD)) mostrada en la figura 5.2, la arquitectura de la red local a la que nos referimos en los párrafo anteriores se absorbe en el nivel de interfase de red como se muestra en la figura 5.3, en donde mostramos las arquitecturas del modelo OSI y la del (DOD).

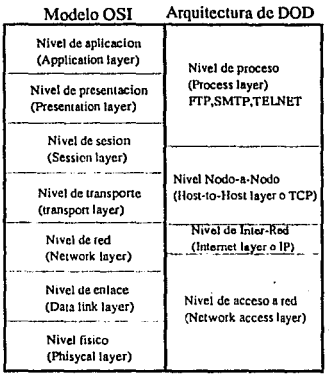

Figura 5.3: Arquitecturas del DOD y la del modelo OSI.

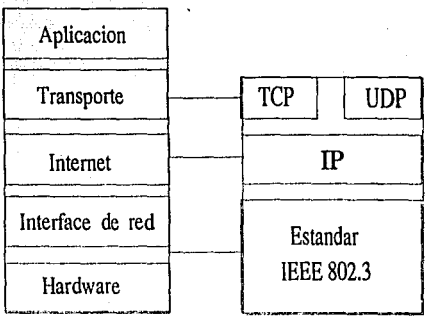

Figura 5.4: Posición de los diferentes protocolos en la arquitectura del DOD

La implementación del sistema hace uso del protocolo IP (Internet Protocol) en el nivel de "internet", del protocolos TCP (Trasmition Control Protocol) y de UDP (User Datagram Protocol) que se encuentran en el nivel de transporte de la arquitectura del DOD, como lo muestra en la figura 5.4. Para mayores referencias sobre este tema consultese [WS90a, WS90b, aDLS88, aDLS91]. Por ahora vamos a describir las caracteristicas y el uso de cada un de estos protocolos.

5.1.1 El protocolo IP y su funcionamiento

El protocolo IP fue desarrollado como parte de un proyecto de "internet" de DARPA y prové un servicio sin conexión entre estaciones, es decir, IP no establece una conexión lógica y no garantiza la llegada de todos los datos enviados, ni tampoco un orden correcto. Que el protocolo ofrezca este tipo de servicio tiene las ventajas de ser flexible porque requiere muy poco de la red, también es robusto porque cada datagrama se direcciona sobre la red independientemente; además ofrece software de aplicación sin conexión.

Todos los paquetes enviados por una máquina viajan por las diferentes redes pasando por los "Gateways" que reciben los mensajes vía un protocolo de acceso a red como se muestra en la figura 5.5; básicamente, el protocolo IP en el "gateway" efectúa una decisión de direccionamiento.

Hay diferentes aspectos que considerar en el protocolo IP, entre estos encontramos: direccionamiento, direccionamiento, tiempo de vida de los datagramas, fragmentación y reensamble, control de errores y control de flujo.

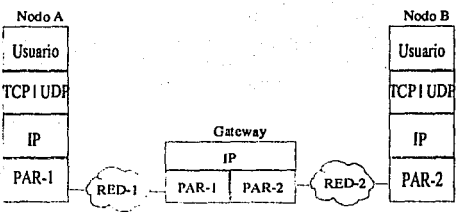

Figura 5.5: Flujo de los mensajes entre redes

Aquí el direccionamiento se puede especificar de diferentes formas:

- La aplicación puede direccionar a una red mediante un número único, en cuyo caso el nombre y la dirección son la misma.
- La lógica de "internet" en donde el nodo puede trasladar un nombre de red en una dirección de red.
- Utilizando un esquema de direccionamiento global en donde hay un único identificador para cada nodo en la red. Para propósitos de direccionamiento cada "gateway" necesita deducir la dirección de la red de la dirección del nodo.

Usualmente los "gateways" reciben paquetes "internet" de la forma **red.nodo**, donde **red** trae la dirección de una red y en **nodo** trae un nombre y una dirección.

Los desarrolladores de **Ethernet** han propuesto otro nivel de direccionamiento que permita identificar puntos de acceso al servicio (PAS) individuales en los nodos; con este nivel de direccionamiento, un identificador del protocolo "internet" va a ser de la forma **red.nodo.PAS**, con el cual un protocolo "internet" se puede ver como un protocolo entre procesos y no como un protocolo entre nodos.

La ventaja de este último nivel de direccionamiento es que el nivel superior puede simplificarse, lo cual daría lugar a la inclusión de dispositivos con procesadores de poca capacidad. Hay ciertos problemas, quizá el mas notorio se relaciona al uso de los puertos que permiten el acceso a servicios comunes. Si los puertos fueron implementados como (PAS) en el nivel de IP, la asignación de puertos conocidos deberá ser centralizado.

El direccionamiento generalmente se efectúa mediante el uso de una tabla en cada nodo y "gateway" en donde se especifica para cada posible destino de la red y el "gateway" por donde se va a enviar el datagrama IP. La tabla de direccionamiento puede ser estática o dinámica. En las tablas estáticas puede haber rutas alternativas para los casos en que haya "gateways" no disponibles. Una tabla dinámica es más flexible en respuesta a situaciones de error y congestión.

Las tablas de direccionamiento también pueden ser usadas para soportar otros servicios "internet" tales como seguridad y prioridad.

Otro punto que incluye el protocolo IP es el tiempo de vida de un datagrama, en este caso, cada datagrama puede etiquetarse con un tiempo de vida y una vez que este tiempo se termine, el datagrama es descartado; esto se puede lograr mediante una máquina de reloj global.

En cuanto a fragmentación y reensamble se definen ciertos mecanismos para ensamblar y desensamblar bloques de información de tal forma que no afecte el tipo de red en donde se este ejecutando.

En cuanto al control de error, este protocolo no garantiza la llegada de todos los datagramas. Cuando un datagrama es descartado por un "gateway", este debe intentar regresar información a la fuente si es posible. El protocolo "internet" de la fuente debe usar esta información para modificar su estrategia de trasmisión y notificar a los niveles superiores la perdida del datagrama. Los datagramas se pueden descartar por diferentes razones, incluyendo la terminación del tiempo de vida, por congestión o por error de bits.

En cuanto al control del flujo, en el nivel de "internet" se tiene un control de la velocidad de recepción en los "gateways" y los nodos receptores; para el tipo de servicio sin conexión que hemos descrito, los mecanismos de control de flujo están limitados.

El protocolo IP ofrece dos primitivas de servicio hacia las capas superiores, que son: SEND y DELIVER. Estas primitivas tienen un conjunto de parámetros que son usados por los protocolos superiores como TCP y UDP, para mayores referencias sobre este protocolo consultese [aDLS88, aDLS91].

5.1.2 Protocolo UDP(User Datagram Protocol)

En la sección anterior describimos el protocolo IP, que permite la transferencia de datagramas entre diferentes computadoras y "gateways" y cada datagrama es dirigido a través de la red basándose en las direcciones "Internet". En el protocolo de la capa "internet" (IP) una dirección destino identifica una computadora y no hay mas distinción como por ejemplo que usuario o que programa de aplicación en esa computadora va a recibir el datagrama. En esta sección se extiende el protocolo "internet" en el que se agregan mecanismos para distinguir múltiples destinos dentro de cierta computadora; lo cual permite que múltiples aplicaciones ejecutándose en una cierta estación permitan enviar y recibir datagramas independientemente.

En las computadoras de múltiple procesamiento parece natural pensar en que el último destino de un datagrama sea un proceso (un programa en ejecución), pero si vemos que los procesos se crean y se destruyen dinámicamente y además que es difícil que los emisores identifiquen un proceso remoto; aparte que sería bueno tener procesos que reciban datagramas sin tener que informarles a los emisores; por estas y otras razones se considera que un proceso no es el ideal para el destino

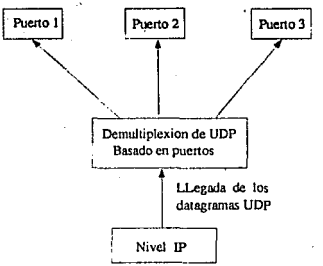

Figura 5.6: Multiplexado y demultiplexado de mensajes

final de un datagrama; a cambio se consideró que en cada máquina existiera un conjunto de puntos abstractos como destino, a los cuales los llamaron **Puertos de protocolo**, los cuales son identificados por enteros positivos. Los sistemas operativos poseen mecanismos para especificar y acesar estos puertos.

Para efectuar la comunicación con un puerto remoto, el emisor necesita conocer la dirección "internet" de la máquina destino y el número de puerto de protocolo destino dentro de esta máquina.

Dentro de cada mensaje, el emisor puede enviar el número de puerto destino y el número de puerto fuente; esto hace posible que el proceso receptor del mensaje pueda contestarle al emisor.

Dado que este protocolo esta sobre el protocolo (IP) y esta dedicado a transportar mensajes (UDP) de una máquina a otra; es igual de inseguro y ofrece un servicio sin conexión como IP. Este tampoco usa mensajes de confirmación para asegurar la llegada del mensaje, no garantice el orden de los mensajes y no utiliza mensajes para controlar la velocidad de flujo. Esto implica que los mensajes pueden ser perdidos, duplicados o llegar en orden incorrecto, sin embargo los paquetes pueden llegar más rápido de lo que los procesos pueden recibirlos.

Una de las funciones fundamentales del protocolo UDP es que permite multiplexar o demultiplexar múltiples objetos. Este acepta datagramas UDP del nivel IP y demultiplexa basándose en los puertos UDP como se muestra en la figura 5.6

Para este protocolo ya hay un conjunto de puertos establecidos dentro de los cuales ya se encuentran algunos ocupados; para efectuar comunicación entre máquinas se eligen puertos que no han sido utilizados; para mayores referencias véase [aDLS88, aDLS91]

5.1.3 El protocolo TCP(Un servicio de conexión confiable).

En niveles bajos, la comunicación entre computadoras ofrece un servicio no confiable; cuando hay errores de trasmisión de datos, cuando el hardware de la red falla o cuando la red esta sobrecargada, los paquetes pueden perderse o destruirse.

En los niveles superiores, los programas de aplicación con frecuencia necesitan enviar grandes volúmenes de datos de una computadora a otra. La programación de un sistema para transferir información usando una conexión no confiable se vuelve tedioso y los programadores necesitan construir aplicaciones que permitan detectar y corregir errores. Como consecuencia se ha diseñado un protocolo de red de uso general que ofrezca un servicio de comunicación confiable, lo cual permite a los programadores el desarrollo de sistemas de comunicación de manera más fácil y eficiente.

En el protocolo TCP se especifica un formato de los datos e información que dos computadoras intercambian para lograr una transferencia confiable; además se especifican los procedimientos que cada computadora usa para asegurar que los datos lleguen correctamente. Se especifica también la forma como el software de TCP distingue entre los diferentes destinos de una cierta máquina y además la forma de recuperarse de errores tales como perdida y duplicación de paquetes. Otras de las especificaciones son la forma de inicio y terminación del protocolo.

Al igual que el protocolo UDP, el TCP se encuentra sobre el protocolo "internet", como se muestra en la figura 5.4

En una máquina especifica el TCP permite la ejecución concurrente de múltiples programas de aplicación y demultiplexa el trafico de TCP entre los diferentes programas de aplicación.

De igual forma que el UDP, el TCP incorpora objetos abstractos denominados puertos que identifican el destino final dentro de una máquina.

A diferencia de UDP, el TCP es un protocolo orientado a conexión que requiere de dos puntos finales para efectuar la comunicación. Antes de que el trafico de TCP pueda pasar a través de "internet", los programas de aplicación en ambos extremos deben ponerse de acuerdo para la conexión deseada. Para lograrlo, el programa de aplicación en un extremo efectúa una apertura pasiva indicando al sistema operativo que va a aceptar una conexión; en este momento, el sistema operativo asigna un número de puerto para un extremo de la conexión. El programa de aplicación en el otro extremo debe contactar al sistema operativo haciendo una petición de apertura activa para establecer conexión. Los dos módulos de TCP se comunican para verificar que la conexión se ha establecido. Una vez establecida la comunicación los módulos de software de TCP en cada extremo pueden iniciar a transferir información. La idea que se utiliza para decidir que un dato se ha perdido y por lo tanto se tiene que volver a trasmitir es que cada vez que se envía un mensaje se corre un reloj y espera una confirmación del destino; si el tiempo se termina antes de que se reciba confirmación de la llegada del dato, el protocolo supone que el dato fue perdido o dañado y por lo tanto se retransmite.

Al igual que UDP, TCP tiene la posibilidad de usar puertos dinámicos y estáticos y los puertos de UDP y TCP son independientes; los programadores pueden elegir usar el mismo número de puerto para un servicio de acceso desde UDP y TCP. Para mayores referencias véase [aDLS88, aDLS91].

5.2 Herramientas de programación.

Las facilidades para la comunicación entre procesos y entre redes fue una de las grandes aportaciones al UNIX en la versión de Berkeley 4.2BSD. La idea básica de esta interfase es hacer la comunicación entre procesos similar a la entrada y salida de archivos. En UNIX un proceso tiene un conjunto de descriptores de entrada y salida, de los cuales uno puede leer y escribir. Los descriptores pueden referirse a archivos normales, a dispositivos o a canales de comunicación. Los **pipes** son otra forma de descriptores que se han usado en UNIX desde hace tiempo y la trasmisión de datos entre procesos se efectúa en una dirección, con la restricción de que los procesos y el **pipe** deben tener el mismo padre.

La identificación de la comunicación entre procesos con la entrada y salida de archivos le dio cierta relevancia al UNIX, ya que al principio la comunicación solo se daba entre procesos locales. Con el UNIX 4.2BSD de Berkeley estos mecanismos se expandieron para incluir la comunicación entre procesos de diferentes máquinas.

Básicamente, las herramientas fuertes que se están utilizando son:

- Una herramienta que le permite a un proceso (denominado proceso padre) la creación de otros procesos (denominados procesos hijos), los cuales solo difieren en los siguientes aspectos:
  - El proceso padre y el proceso hijo tienen diferentes identificadores de proceso.
  - El valor que regresa la función que crea el proceso hijo (fork) es diferente en los dos procesos. En el proceso padre, el valor que regresa es el número de identificador de proceso del hijo, mientras que en el proceso hijo es cero.
  - El proceso hijo hereda todos los archivos abiertos y **pipes** del padre; sin embargo el proceso hijo tiene su propia copia de los descriptores de archivo y **pipe**.

    La forma como se efectúa la llamada a la función para crear proceso es:

    $$proceso = fork()$$

    A la variable *proceso* se le asigna **cero** o **uno** dependiendo si el proceso es el hijo o el padre.

- Otra de las herramientas que estamos utilizando son los **pipes**, que es un mecanismo que permite la comunicación entre procesos, este mecanismo nos permite sincronizar procesos y ofrece un canal unidireccional para la comunicación asíncrona y es usado por los procesos para comunicarse y sincronizarse. Para enviar información de un proceso a otro, un proceso envía por un extremo del **pipe** y otro proceso lee por el otro extremo y el **pipe** automáticamente controla la información entre estos.

  Los **pipes** son similares a los archivos, pero se tienen las siguientes diferencias:

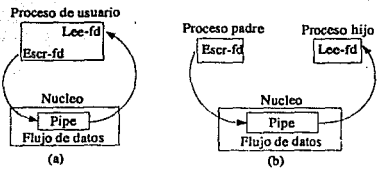

Figura 5.7: (a).- Estructura de la comunicación entre procesos mediante **pipes** con un solo proceso (b).- Estructura de la comunicación con dos procesos.

1. Al intentar escribir en un **pipe** lleno, el **pipe** hace que el proceso de escritura se espera hasta que el **pipe** se vacié; es decir, hasta que el proceso de lectura lea del otro extremo del **pipe**. De igual forma, un intento de lectura de un **pipe** vacio hace que el proceso espere hasta que el **pipe** contenga información; es decir, hasta que el proceso de escritura escriba información.
2. Los **pipes** se deben accesar secuencialmente mientras que los archivos se pueden accesar aleatoriamente.

La función que permite la creación de **pipes**, es

$$pipe(pip)$$

Donde *pip* es un arreglo de dos enteros. Esto genera una imagen como la que se muestra en la figura 5.7(a).

En general se crean procesos para escritura y lectura y se puede crear un **pipe** de tal forma que se produce una imágen como la que se en la figura 5.7(b).

Aparte de los **pipes** existen otros mecanismos de comunicación que ofrece UNIX, tales como FIFO, colas de mensajes y semáforos; el lector interesado sobre **pipes** y estos otros mecanismos de comunicación consultese [Ste89].

- Otra de las herramientas que se esta utilizando es una estructura de datos denominada **sockets** que permite la comunicación entre procesos locales a los cuales les llaman procesos del dominio UNIX y se denotan por AF_UNIX y la comunicación entre procesos de la red a los cuales les denominan procesos del dominio "Internet" y se denotan como AF_INET, entre otros.

Los **sockets** tienen un tipo abstracto asociado, el cual describe la semántica de comunicación y las propiedades tales como confiabilidad, orden y prevención de duplicación de mensajes que

se determinan por el tipo. Los diferentes tipos de **sockets** que existen son: SOCK_DGRAM, SOCK_STREAM, SOCK_RAW, SOCK_RDM y SOCK_SEQPACKET.

En este caso solo se utilizan SOCK_DGRAM y SOCK_STREAM. Los SOCK_DGRAM modela la semántica de los datagramas en la red de comunicación; en este caso los mensajes pueden ser perdidos, duplicados o pueden llegar fuera de orden. Un **socket** de tipo datagrama puede enviar y recibir mensajes de múltiples entidades. Los **sockets** de tipo SOCK_STREAM modela la comunicación basada en circuitos virtuales y la comunicación se debe establecer antes de iniciar el intercambio de mensajes.

Cada tipo de **socket** tiene un protocolo asociado y se usa dentro del dominio para suministrar la semántica requerida por el tipo de **socket**. En particular dentro del dominio "Internet", los SOCK_DGRAM pueden ser implementados por el protocolo UDP y los SOCK_STREAM pueden ser implementados por el protocolo TCP. Para el protocolo UDP existe una primitiva no confiable para la difusión, en este caso solamente se utiliza una dirección de difusión; la sintaxis y la estructura se llenan de la misma forma como si se efectuara comunicación punto a punto.

Para los **sockets** existen una serie de funciones o llamadas al sistema que permiten crear y establecer la comunicación entre los diferentes nodos, pero no los describiremos, para el lector interesado consultese [mic90, Ste89, aDLS88, aDLS91] y para mayores referencias de programación en **C** y **UNIX** consultese [KP87, KR87, GEH85].

Capítulo 6

Especificación e implementación de un protocolo por difusión tolerante a fallas.

6.1 Introducción.

Con la aparición de las redes de computadoras, el área de computación que más importancia ha adquirido son los Sistemas Distribuídos. Lo que caracteriza a un sistema distribuído no es solo la distribución de sus partes, sino también que sus partes se puedan comunicar. El hardware en un Sistema Distribuído permite la emisión de mensajes entre procesadores y por lo general el sistema operativo extiende esta facilidad permitiendo la comunicación entre procesos de diferentes máquinas. El sistema operativo también puede ofrecer un tipo de circuito virtual para la comunicación entre procesos y puede incluir protocolos que aseguren un cierto grado de confiabilidad en la comunicación, sin embargo, desde el punto de vista de la programación, estas facilidades no dejan de ser de bajo nivel. Esto ha llevado a la búsqueda de abstracciones más apropiadas y de más alto nivel que permitan la comunicación entre procesos. Algunos investigadores sugieren una comunicación a través de un tipo de memoria global compartida ocultándole al programador la parte de la distribución; esta abstracción tiene la ventaja de que cualquier programa escrito en un sistema no distribuído se puede trasladar a uno distribuído de manera transparente.

Una abstracción de alto nivel muy usada para comunicación entre procesos son los llamados "Remote Procedure Call(RPC)". Un proceso se comunica con otro utilizando una interfase que es similar a una llamada a un procedimiento. La ventaja de esta abstracción es que esto simplifica la programación distribuída, porque permite la llamada a un proceso remoto como si se efectuara entre procesos locales. Una de las desventajas es que está pensada para modelos tipo "cliente-servidor"; por lo tanto, los "RPC" no son una abstracción conveniente cuando un programa distribuído se compone de un cierto número de procesos que tienen un alto grado de interdependencia entre sí y la comunicación entre ellos refleja su interdependencia. En este tipo de programas la comunicación que se requiere es una comunicación de un proceso a un conjunto de procesos, en este caso, lo que

se requiere es un tipo de servicio que permita a un proceso enviarle a un conjunto de procesos; esto es lo que llamamos **protocolos por difusión** en el capítulo dos.

En los Sistemas Distribuídos frecuentemente se necesita resolver problemas difíciles que van desde el manejo de replicación de datos hasta la reconfiguración dinámica en respuesta a fallas. Todos estos problemas se reducen a otras primitivas que resuelven problemas de consistencia, las cuales se pueden implementar usando **protocolos por difusión confiables o tolerantes a fallas** que también fueron definidos en el capítulo dos.

Una de las áreas donde es natural el uso de primitivas para difusión es en las bases de datos distribuídas, en donde es necesario la replicación de estructuras de datos en múltiples lugares para tolerar fallas y reducir el tiempo de acceso.

Otra área en la cual se ha estado aplicando los protocolos por difusión es para crear memorias compartidas distribuídas, lo cual permite el desarrollo de lenguajes de programación para efectuar paralelismo en sistemas multicómputo.

En esta tesis lo que se vamos a especificar es un **protocolo por difusión de control centralizado tolerante a fallas.**

6.2 Criterio de selección.

Como se ha visto en los capítulos anteriores, tenemos dos clases de protocolos: los protocolos centralizados y los protocolos distribuídos. Dentro de cada clase de protocolos hay una infinidad de estos que se pueden especificar; unos más difíciles de implementar que otros, unos más eficientes que otros. En general: lo que se busca es especificar un protocolo fácil de implementar y eficiente. La facilidad de implementación y eficiencia dependen del nivel (del modelo OSI) donde este montado. En este caso el protocolo va estar montado sobre el nivel de transporte del modelo OSI y la comunicación se va ha hacer utilizando el protocolo (TCP/IP) y (UDP/IP). Es claro que un protocolo entre más confiable se requiera, en general es menos eficiente; el objetivo principal de este trabajo es desarrollar un protocolo **confiable y eficiente.** Como veremos en este protocolo si la comunicación es confiable, entonces el protocolo es muy eficiente; sin embargo si hay ciertas fallas, este degrada en eficiencia. Este protocolo se eligió porque es eficiente, es confiable y es factible de implementar.

6.3 Especificación del servicio que ofrece el protocolo.

Sabemos que en el modelo (OSI) se especifican los servicios entre las diferentes capas, indicando el servicio que se ofrece hacia la capa superior e indicando el servicio que obtiene de la capa inferior. En este caso vamos a indicar cual es el servicio que ofrece hacia la capa superior y los servicios que se usan del nivel inferior, pero antes de especificar el protocolo vamos a definir todos los términos usados en la especificación.

Definición 27 *Las fallas que se van a considerar en este caso son:*

1. *Perdida de mensaje, el cual puede presentarse por falla de la línea o por falta de espacio en el "buffer" del nodo receptor. Esta falla será detectada por el nodo receptor del mensaje.*

2. *Otra de las fallas que vamos a considerar es la caída de nodos, en cuyo caso consideramos que si un nodo deja de funcionar, este no emite mensajes erróneos. Se va a considerar que un nodo falla si después de un número de intentos(MaxF), no hay respuesta de parte del nodo.*

3. *Vamos a suponer que si falla el nodo líder, ninguno de los otros falla al mismo tiempo.*

4. *Se pueden soportar fallas de mas de un nodo al mismo tiempo, pero sin incluir el líder.*

Definición 28 *Un servicio de difusión es confiable si ofrece un servicio que:*

1. *Garantice la recepción de todos los mensajes por todos los nodos que no fallen y*

2. *Los mensajes deben ser usados en el mismo orden en cada uno de los nodos.*

3. *Si algún nodo falla, el protocolo debe ser capaz de recuperar los mensajes perdidos por dicho nodo.*

Este protocolo no ofrece una primitiva general para cualquier ambiente, sino que el servicio que ofrece está restringido a un grupo (G) después de que se ha generado un medio ambiente. La primitiva que se ofrece al usuario en ese medio ambiente va a ser "env mnsj', la cual va a ser transparente al usuario. El protocolo va a garantizar que el mensaje "mnsj" llegue a todos los nodos que no fallen y que pertenecen al grupo; para ofrecer esta primitiva, el protocolo se divide en dos etapas:

1. Creación de un medio ambiente.

2. Ejecución del protocolo de cooperación en el medio ambiente establecido.

6.3.1 Supuestos sobre el medio ambiente en donde se implementa el protocolo.

Vamos a suponer que tenemos una sistema de comunicación que permite la difusión de mensajes(una red de tipo Bus), en cuyo caso, si un nodo envía un mensaje (m_1) y después (m_2), entonces se difundirá por el medio primero m_1 y después m_2; es decir, se tiene un mecanismo de tipo FIFO. Pero esto no garantiza la llegada de m_1 antes de m_2, porque puede ser que m_1 se pierda por falla, lo cual implicaría la reemisión de m_1, en cuyo caso llega después de m_2.

Para poder establecer el ambiente, vamos a suponer que en cada nodo se tienen procesos para enviar y recibir datos y además el proceso de aplicación que en este caso vamos a suponer que es el receptor final de los datos.

En el nodo líder aparte de estar los procesos para enviar y recibir mensajes, debe haber un proceso que sea capaz de atender a las peticiones de los nodos para la repetición de un dato.

Todos los procesos que se ejecutan en los nodos y los que se ejecutan en el líder van ha estar presentes en cada nodo del grupo, aunque en cada nodo habrán ciertos procesos desactivados.

Un punto importante es que si un nodo falla y se recupera antes de que el líder se actualice, éste puede continuar en el proceso de cooperación ejecutando el protocolo.

Si el nodo no se recupera después de la actualización, éste debe salir de la lista del grupo de cooperación y no vuelve a entrar.

Observemos que al introducir tolerancia a fallas, el protocolo corre en un ambiente dinámico, por lo tanto el ambiente inicial puede cambiar; lo que se debe garantizar es la autoestabilización del protocolo cuando se presenten fallas(protocolo autoestable).

En este caso, en la estabilización del protocolo se debe incluir la recuperación de la información de los nodos que hayan fallado incluyendo el líder.

El comportamiento esperado de cada uno de los nodos es el siguiente:

- El líder es capaz de enviar y recibir mensajes, atender al resto de los procesadores para reenviar los mensajes que estos hayan perdido y al mismo tiempo debe estar resolviendo el problema que todos los demás nodos están resolviendo.
- Todos los procesadores que no son el líder son capaces de enviar y recibir mensajes y resolver el problema al mismo tiempo.
- Cada nodo es responsable de verificar que a la aplicación se le envíen los mensajes correctos y en el orden correcto, porque el líder solo etiqueta y envía, pero no verifica que el mensaje haya llegado a los nodos.

6.3.2 Comportamiento general del protocolo.

Como vimos, el protocolo se divide en dos niveles:

- Creación de medio ambiente.
- Protocolo de comunicación que incluye el mantenimiento de integridad del medio ambiente.

En la creación del medio ambiente se efectúan los siguiente pasos:

- Elección de un grupo de cooperación.
- Elección de líder.

Una vez establecido el medio ambiente, se puede iniciar el protocolo de comunicación que se basa en los siguientes procesos:

- De manera asíncrona cualquiera de ellos puede decidir difundir un mensaje; esto se hace enviando el mensaje al líder. En la comunicación entre el nodo y el líder se usa un protocolo confiable que garantice la recepción correcta del dato a enviar. En este caso, el líder confirma la llegada del dato, en la cual envía el número de secuencia asignado al dato y el nodo lo almacena en un "buffer" local.

- El líder almacena el mensaje en un buffer de "Historia", en donde van ha estar almacenando todos los mensajes que reciba para difundir.
- El líder difunde el mensaje a todos los nodos del grupo de cooperación usando comunicación no confiable.
- Cuando cada nodo recibe el mensaje verifica que sea el mensaje que estaba esperando.
- Si el mensaje es el correcto entonces lo pasa a la aplicación; pero no confirma si lo ha recibido o no.
- Si el mensaje no es correcto lo almacena y pide al líder el reenvío del dato esperado.
- Una vez que el líder sabe que todos los nodos han recibido un cierto dato, este lo saca del buffer de "Historia". Esto lo sabe porque en cada mensaje que se envía al líder, se envía el número de secuencia del dato hasta el cual ya ha recibido este nodo. El líder encuentra el mínimo de todos los números de secuencia que indican hasta cual dato ha recibido cada nodo y lo envía en el mensaje que difunde.

Algoritmo para detección de fallas.
Como hemos visto, las fallas pueden ser de los nodos o del líder. El líder detecta si falla algún nodo y los nodos detectan si falla el líder.

1. **Detección de la falla de un nodo.**
   Si el buffer de "Historia" del líder se llena, puede ser porque un nodo esta lento (en cuyo caso no es una falla) o porque falló un nodo.

   Cuando el buffer de "Historia" se llena, el líder comienza una etapa de actualización, en la que envía un mensaje a todos los nodos del grupo de cooperación (G) para que se actualicen; si en este llamado(después de un tiempo determinado) algún nodo no contesta, entonces se infiere que ha fallado y lo saca de la lista del grupo de cooperación y envía la nueva lista al resto de nodos. Observemos que en este caso no se pierden mensajes porque los que había enviado el nodo se encontraban en el buffer de "Historia" del líder. En el proceso de actualización los nodos dejan de enviar nuevos datos hasta que todos terminan de actualizarse.

2. **Detección de falla del líder.**
   La falla del líder se detecta cuando después de un cierto número de intentos un nodo no puede entablar comunicación; esto puede ser en el momento que el nodo quiere enviar un dato o en el momento que pide retransmisión. Cuando el nodo detecta la falla del líder, inicia una fase de reestructuración y el proceso que sigue es:

   - Envía un mensaje al resto de nodos informando la falla del líder.
   - Todos los nodos que reciben el mensaje le contestan y se forma un nuevo grupo de cooperación.
   - Se selecciona un nuevo líder y se difunde la lista del nuevo grupo.

- Una ves establecido este nuevo ambiente, los nodos vuelven a enviar los datos que tenían en los "buffers" y el proceso continua ejecutándose de forma normal.

Observemos que en este caso tampoco se pierden mensajes porque estos son desechados hasta que se avisa que han llegado a todos los nodos. Todos los mensajes que se guardan en los nodos son aquellos que no habían llegado a todos los nodos.
Nota: Si hay algún nodo que solo es receptor, cada cierta frecuencia éste enviará al líder un mensaje avisando hasta que dato ha recibido.

6.4 Análisis del protocolo.

En esta sección vamos a efectuar el análisis del protocolo en donde se encontrará el número de mensajes que se envían por cada etapa del protocolo, incluyendo:

1. Elección del grupo de cooperación.
2. Elección del líder.
3. Envío de datos.
4. Fallas de los nodos.

Para efectuar el análisis vamos a considerar que tenemos los siguientes datos

- M → Es el número de nodos que participan en el protocolo incluyendo el líder.
- MaxF → Es el máximo número de intentos de comunicación para decidir que un nodo ha fallado.
- MaxB → Es la capacidad máxima del buffer del líder donde se almacenan los datos que se han difundido.
- El número de mensajes que se envían por cada dato en el protocolo confiable son dos.
- El número de mensajes que se envían por cada difusión es uno.

Número de mensajes para la elección del grupo de cooperación
En este caso se utiliza la difusión; por tanto se envía 1 mensaje para ver los nodos dispuestos a cooperar y 1 mensaje por cada nodo que contesta, por lo tanto el total de mensajes es M, el nodo que inicia no responde con ningún mensaje.

Número de mensajes para la elección del líder
En este caso se usa comunicación confiable. Como el líder en este caso es el primero de la lista de los nodos dispuestos a cooperar, entonces solo se gastan dos mensajes por cada nodo que contestó; este se gasta en el momento de difundir la lista al grupo dispuesto a cooperar. Por lo tanto son

2(M-1).

Número de mensajes enviados para difundir un dato.

Para difundir un dato se efectúan 2 pasos:
El nodo que desea difundir el dato lo envía al líder usando el protocolo confiable y después el líder difunde el dato al grupo (G) usando la difusión; por lo tanto el número de mensajes enviados por cada dato son 3.

Número de mensajes enviados por falla de nodos diferentes de líder.

Por cada nodo que falla se envían (1+ MaxF) mensajes para decidir que ha fallado.

Una vez que se ha detectado la falla de algún nodo se envían:
2(M-2) + 2(M-2)=4(M-2) mensajes si solo falla un nodo.
2(M-3) + 2(M-3)=4(M-3) mensajes si fallan dos.

Los primeros (M-2) o (M-3) son los mensajes que gastan para enviar respuestas (de que siguen activos) y los otros son los que se gastan para difundir la lista.

Por lo tanto el número de mensajes que se envían si fallan $k < M$ nodos son:

$$1 + k(MaxF) + \underbrace{4(M-k-1)}_{Nodos\ que\ contestan} + \underbrace{4(M-k-1)}_{En\ difundir\ la\ nueva\ lista.}$$

En este caso no se gastan mensajes recuperando datos porque no se pierden datos.

Número de mensajes enviados por falla del líder.

Para que un nodo decida que el líder ha fallado debe efectuar MaxF intentos, por lo tanto se gastan MaxF mensajes para detectar falla del líder.

Como cada nodo se comunica con el líder independientemente de los demás nodos, entonces puede ser que haya mas nodos tratando de comunicarse con el líder al mismo tiempo, pero no pueden usar mas de MaxF mensajes.

El peor caso es cuando todos los nodos tratan de comunicarse con el líder al mismo tiempo y entonces el número de mensajes es a lo mas (MaxF)(M-1).

El mejor caso es que solo un nodo intente comunicarse con el líder y en este caso el número de mensajes es MaxF.

Una vez que se ha detectado la falla del líder, se envía un mensaje a todos los nodos anunciando la falla del líder(en este caso se gastan 2(M-2) mensajes); se envía la nueva lista al resto de nodos incluyendo el nuevo líder(se gastan 2(M-2) mensajes).

Cuando ya se tiene el nuevo líder, se inicia la etapa de actualización y en este caso en la recuperación de la información se envían a lo mas 2(MaxB)+(MaxB) datos porque es el máximo de

datos que puede almacenar el líder. En este caso el máximo número de mensajes enviados es a lo mas $3(MaxB)$.

A pesar de la tolerancia a fallas, el número de mensajes enviados no aumenta demasiado; aunque si se presentan demasiadas fallas el protocolo se puede volver lento.

Resumiendo:

- En el mejor de los casos, el número de mensajes por cada dato a difundir es

$$2 + 1 = 3$$

y por cada dato perdido se gastan otros 2 mensajes.

- El peor caso se puede presentar cuando:
  1. Fallan k nodos, en cuyo caso el número de mensajes necesarios para la recuperación son:

$$1 + k(MaxF) + 8(M - k - 1)$$

  2. Si falla el líder, el número de mensajes para:
     - Detectar la falla son:
       - El peor caso (M-1)(MaxF).
       - El mejor caso (MaxF).
     - Elección de nuevo líder son: 4(M-2).
     - Recuperación de información 3(MaxB).

6.5 Discusión y comparación

Ha habido diferentes diseños de protocolos por difusión; algunos de estos utilizan la comunicación punto a punto que por ninguna razón van a ser más eficientes que los que utilizan comunicación por difusión. En este caso, el protocolo utiliza la ventaja que ofrecen las redes "ethernet"(comunicación por difusión). Para poder hacer comparaciones vamos a describir los protocolos con los cuales se va a comparar.

1. **Protocolo de Birman y Joseph(BJ)** [Mul89]
   La forma de operar de este protocolo es que a cada mensaje le asigna una etiqueta de tiempo y estos mensajes se liberan en el orden como se les asocian las etiquetas.

   Cuando un sitio recibe un nuevo mensaje, lo almacena en una cola de pendientes. Después envía un mensaje al que se lo envió con una propuesta de etiqueta para que sea difundido. El nodo emisor colecciona todas las etiquetas de tiempo propuestas, tanto por él como las del resto de los nodos que están cooperando; selecciona la más grande y envía este valor a

los nodos receptores. Esta es la etiqueta que se le asigna al mensaje. Cuando un receptor recibe esta etiqueta, se la pone al mensaje que estaba en la cola de pendientes y marca el mensaje como liberable y la cola de pendientes es reordenada de acuerdo a las etiquetas de tiempo asignada a cada dato y el que esta al final de la cola es liberado; este proceso se repite hasta que la cola se vacía. Este protocolo requiere del nivel de transporte para efectuar comunicación confiable punto a punto.

Análisis de este protocolo

Si consideramos que el protocolo (BJ) utiliza comunicación confiable en la emisión punto a punto, entonces el número de mensajes para poder aceptar un dato serían los siguientes:

- 2(M-1) para enviarlo por primera vez.
- 2(M-1) en las respuestas de los receptores.
- 2(M-1) para enviarlo por segunda vez.

Por lo tanto el número de mensajes para enviar un dato son por lo menos: 6(M-1).

En cuanto a espacio de memoria tenemos que:

- Por cada mensaje se requieren M localidades.
- En este protocolo se considera espacio de memoria variable: la liberación de la memoria va a depender de la frecuencia a la que se intercambien los mensajes. Entre menos mensajes se envíen, la memoria se liberará más rápido.

2. **Protocolo Chan y Maxemchuck (CM)** [CM84].

Este protocolo no requiere de comunicación confiable punto a punto, solo utiliza comunicación no confiable.

En este protocolo, a un miembro del grupo se le asigna una ficha y al nodo se le denomina "Token site". Este "Token site" asigna una etiqueta de tiempo a cada mensaje a difundir y los mensajes son enviados a todos los receptores en el orden que fueron etiquetados. Esto asegura que todos los mensajes por difusión sean liberados en el mismo orden a todos los miembros del grupo.

Este protocolo necesita que la ficha sea transferida periódicamente de un sitio a otro. La lista de todos los posibles sitios que pueden recibir la ficha es denominado "token list" y se encuentra en cada uno de los nodos que están en esta lista. El "token site" pasa la ficha al siguiente sitio en esta lista.

El protocolo funciona perfectamente mientras el número de fallas sea menor que el tamaño de la lista. Los sitios entran en una etapa de reformación cuando se detecta que la lista ha cambiado, ya sea porque un nodo ha fallado o porque un nodo desea entrar en la lista de cooperación.

En este protocolo un mensaje es enviado a la aplicación y la memoria es liberada solamente cuando la ficha ha pasado por todos los nodos de la lista. Al final de la primera ronda, sabemos que el mensaje ha sido recibido por todos los nodos y entonces se comienzan a liberar los mensajes.

Análisis del protocolo.
Para transferir la ficha en este protocolo se utiliza comunicación confiable; esta transferencia se efectúa cada cierta cantidad de mensajes, digamos (m), por lo tanto el número de mensajes gastados para liberar (m) mensajes son: Por cada mensaje se gastan 2 en la comunicación punto a punto mas 1 en la difusión. Por cada ronda se gastan 2(M). Es decir, se gastan (3m+2M) mensajes para poder liberar (m) mensajes.

Uso de memoria:
Si consideramos que cada (m) mensajes la ficha se envía al siguiente nodo en la lista, entonces el espació de memoria requerido es del orden de $\frac{3(Mm)}{2}((Mm)^2 - 1)$.

La especificación descrita (PE) en esta tesis está basada en las ideas del articulo de Kaashoek F. & Tanenbaum (KT) [MFK89], pero le hemos agregado tolerancia a fallas de nodos(líder o cualquier otro).

La eficiencia es idéntica a la que ofrece el protocolo descrito en este articulo cuando no hay fallas y aún cuando hay perdida de mensajes.

En el articulo de (KT) no se considera la posibilidad de fallas(como por ejemplo, caída de nodos); pero en este trabajo se consideran fallas hasta de todos los nodos excepto el líder o del líder pero de ningún otro nodo; en otro caso no es posible recuperar todos los mensajes perdidos.

Los tres protocolos (PE), (KT), (CM) y (BJ) dependen de un nodo central para ordenar los mensajes.

A diferencia del protocolo de (KT) que solo utiliza un "buffer" de historia para almacenar los mensajes, el protocolo especificado (PE), (BJ) y el de (CM) utilizan mas memoria para almacenar los mensajes enviados; esto se debe al hecho de soportar fallas. Para ser mas precisos, el protocolo especificado(PE) utiliza el doble de memoria de la que utiliza el protocolo de (KT). El protocolo (PE) utiliza mucho menos memoria que el (CM); de hecho el espacio de memoria que ocupa (PE) es constante y por lo tanto acotado.

En cuanto al número de mensajes el (PE) y (KT) son iguales cuando no existen fallas; sin embargo, en el (PE), el número de mensajes aumenta cuando se presenta alguna falla. Con respecto al protocolo (CM), en el (PE) el número de mensajes es menor cuando no hay fallas, aunque en presencia de muchas fallas en ambos protocolos, el número de mensajes aumentará demasiado, con la desventaja de que en el (PE) sería mas difícil de recuperarse.

Un punto importante en el (PE) es que la eficiencia no esta determinada por la velocidad de trasmisión, sino por la velocidad de procesamiento de los nodos. En este caso en el protocolo de (CM) el envió de un mensaje genera 2(n-1) interrupciones, mientras que en el (PE) solo se generan (n+1).

Si comparamos el protocolo (PE) con el (BJ), tenemos que el número de mensajes para difundir un dato en (PE) es menor que en el (BJ). En cuanto a memoria tenemos que el espacio de memoria requerido para cada mensaje es menor en el (PE) que en el (BJ); aunque en el (BJ) el espacio de memoria puede ser menor si la frecuencia de intercambio de mensajes es pequeña.

Hacer comparaciones con otros protocolos como los citados en [MFK89] no es tangible por el hecho de que estos utilizan enfoques diferentes.

6.6 Especificación formal.

En la especificación formal del protocolo utilizamos las redes de Petri por ser herramienta formal y por su poder de expresión gráfica. En este caso no se va ha especificar cada detalle por el hecho de que se vuelve demasiado complejo el esquema, solamente vamos a especificar de manera global el comportamiento de éste.

En la gráfica(6.1) se muestra la forma como el usuario envía un mensaje a un proceso que posteriormente es enviado al líder. El nodo espera un cierto tiempo para que el líder le confirme la llegada del mensaje. Si una vez transcurrido este tiempo, el líder no responde entonces se inicia una etapa de actualización, en la cual se selecciona un nuevo líder, se actualizan los nodos y se vuelve a asignar permiso de emisión a todos los nodos. En la etapa donde contestan y se actualizan los nodos (mostrada en la gráfica) podría desglosarse a mas detalle, pero se vuelve menos legible el esquema y por eso lo dejamos a este nivel.

Observemos también que si el líder recibe el dato, lo envía al "buffer" de historia y lo difunde. Por el lado de la difusión tenemos los procesos que reciben los datos en los nodos y el proceso que envía a la aplicación, en cuyo caso cuando todos los nodos han confirmado la recepción de un dato, éste se puede eliminar del "buffer" de historia. En este caso los procesos descritos en cada transición también se podrían extender a detalle pero no lo haremos.

Por otro lado, vemos que cuando el "buffer" de historia llega al máximo, se dispara una transición que desabilita la emisión de datos de los nodos y habilita las transiciones para que cada nodo inicie la etapa de actualización. En este caso podría ser que halla falla del nodo; en tal caso se actualiza la lista y se vuelve a regresar el permiso de emisión a los nodos. Si no hay falla de ningún nodo entonces todos confirman que se han actualizado y se vuelve a regresar el permiso de emisión a los nodos. Aquí también se pueden extender las transiciones de actualización y recepción de lista a una red de Petri con más detalle pero tampoco lo haremos.

Para poder validar el protocolo veamos que las partes de las que consta(falla del líder, falla de un nodo, etapa de actualización y proceso normal sin fallas) son independientes una de la otra y vamos a suponer además que el proceso sin fallas es absorbido en el momento de que se inicie algún otro. Las partes en las que se divide el modelo general son:

- Falla del líder representado en la gráfica 6.2.
- Falla de un nodo representado en la gráfica 6.3.
- Proceso de actualización representado en la gráfica 6.4.
- Proceso general sin fallas representado en la gráfica 6.5.

Creemos que cada parte se explica por si sola y no vamos a extendernos a mas detalle porque se volvería demasiado complejo y quizá difícil de entender; por ahora veamos al proceso de prueba para este protocolo.

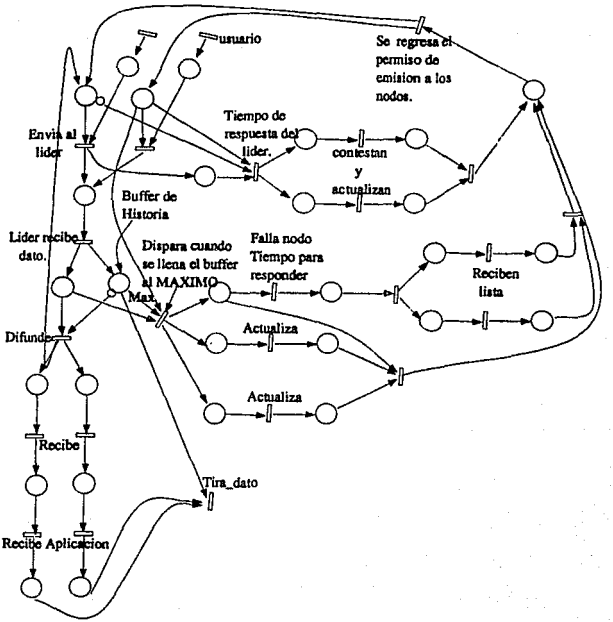

Figura 6.1: Especificación general del protocolo.

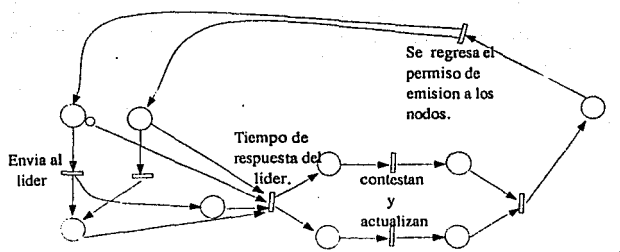

Figura 6.2: Modelo de falla del líder

6.7 Validación del protocolo.

Como hemos visto, la validación de un modelo puede tener diferentes aspectos, pero en este caso vamos a validar el modelo de acuerdo a las técnica de especificación formal que hemos descrito. En este caso lo que vamos a hacer es transformar cada parte en un modelo mas reducido utilizando la técnica descrita en el capítulo dedicado a pruebas de protocolos y una vez transformado, encontraremos la gráfica de estados alcanzables y entonces veremos si presenta o no las propiedades descritas.

En primer lugar, la gráficas que representan la falla del líder 6.2 y la que representa la falla de un nodo 6.3 pueden transformarse en la gráfica 6.7. La gráfica que representa el proceso de actualización 6.4 se transforma en 6.8. Para estas gráficas, la gráfica de estados alcanzables es idéntica y se representa en la figura 6.9.

Observemos que la gráfica de estados alcanzables (6.9) es fuertemente conexa, por lo tanto es reversible, viva y libre de abrazos mortales.

Por otro lado, la gráfica del proceso general se puede transformar como la que se muestra en la figura 6.6. En este caso no vamos a desplegar su gráfica de estados alcanzables, pero se puede ver que no es reversible, es acotada y es viva.

6.8 Implementación.

Para poder implementar un protocolo, no basta con describir el algoritmo, sino que también es importante describir tanto el tipo de mensajes que se van a intercambiar como el formato que van a poseer.

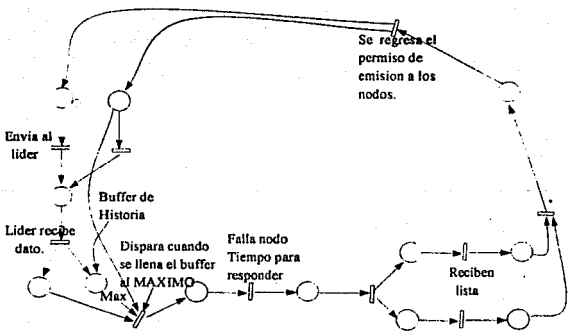

Figura 6.3: Modelo de falla de un nodo

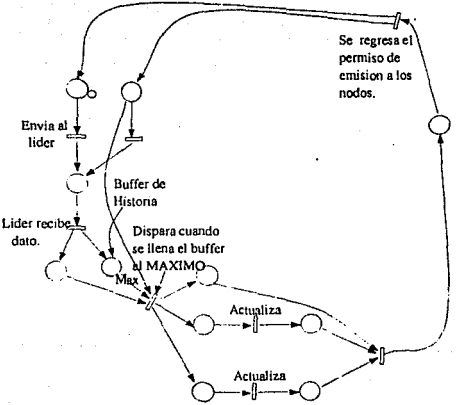

Figura 6.4: Modelo de la etapa de actualización

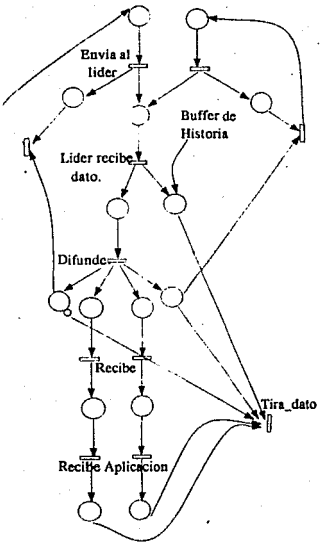

Figura 6.5: Modelo general sin fallas

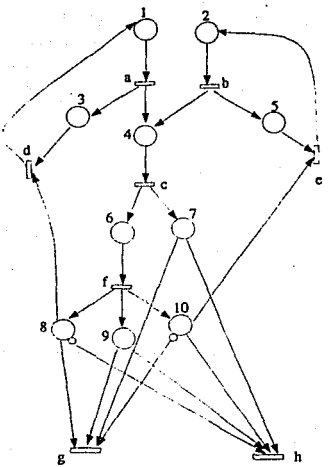

Figura 6.6: Transformación del modelo general

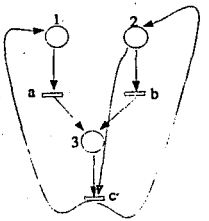

Figura 6.7: Transformación del modelo de falla del líder

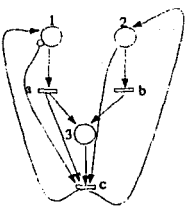

Figura 6.8: Transformación del modelo de la etapa de actualización

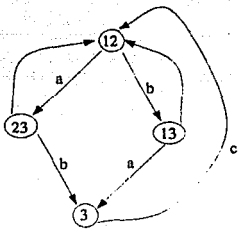

Figura 6.9: Gráfica de estados alcanzables de los modelos: falla del líder, falla de un nodo y etapa de actualización

En esta sección vamos a describir cual es el tipo de mensajes que se van a intercambiar en el protocolo y además el formato de estos; después vamos a describir desde el punto de vista funcional la forma como se lleva a cabo el intercambio de mensajes en el protocolo en general.

6.8.1 Vocabulario.

Como se ha observado, los tipos de mensajes que se envían en el protocolo dependen tanto de la actividad que se este realizando como del nodo que este enviando. En este caso vamos a considerar que el mensaje tiene la siguiente estructura:

$$mensaje = \{TIPO, Estructura - con - el - mensaje\} \tag{6.1}$$

y los tipos de mensajes se dividen en tres, los mensajes que van de:

- NODO → LIDER
  - DATO → Dato a difundir.
  - REENV → Indicando que se repita un dato especifico.
- NODO → NODO
  - LIDER → Indicando elección de nuevo líder.
  - CONFL → Confirmación del nodo.
  - NUEL → Nueva lista.
- LIDER → NODO

- DATO → Dato difundido.
- ACTUAL → Indica al nodo que deje de enviar y se actualice.
- CONTIN → Le indica al nodo que continúe el protocolo.
- NUELIS → Indica que no es un nuevo mensaje, sino una nueva lista.

Observemos que los mensajes no son solamente para comunicación de datos, sino también se necesitan mensajes para mantener el medio ambiente (el mismo grupo de cooperación y el mismo líder) en todos los nodos; todos estos nuevos mensajes que no tienen nada que ver con los datos, se hacen necesarios por el hecho de tener tolerancia a fallas. Veamos el formato y la información que debe enviarse junto con el mensaje.

6.8.2 Formato.

En general, la tolerancia a fallas es un problema difícil de resolver, un protocolo tolerante a fallas se vuelve complejo tanto en tiempo como en número de mensajes; en este caso la estructura del mensaje también se vuelve un poco compleja porque en este caso tratamos de minimizar el uso del canal de trasmisión. Como vimos (6.1) muestra la estructura del mensaje que se envía, y el TIPO pertenece al conjunto

`{DATO, REENV, LIDER, CONFL, NUEL, ACTUAL, CONTIN, NUELIS}`.

La estructura del dato cuando se envía del nodo al líder contiene los siguientes campos
{ *Tipo,*
Identificador del dato que se envía.
Identificador del nodo que envía.
Identificador del último dato recibido.
dato }

El identificador del último dato recibido va a ser usado por el líder para poder desocupar el "buffer" donde almacena todos los datos que se han enviado para difundir.

Cuando el líder difunde un dato, la estructura que se considera es la siguiente
{ *Tipo*
Identificador del dato que ha sido recibido por todos los nodos,
Numero de secuencia del dato.
Identificador para detectar duplicados.
dato }

El identificador del dato que ha sido recibido por todos los nodos es usado en los nodos para vaciar el "buffer" de los datos que ha enviado cada nodo.

En un "buffer" cada nodo almacena los datos que ha enviado; la condición para sacar un dato del "buffer" es que todos los nodos ya hayan recibido dicho dato; esto se sabe cuando el líder envía el identificador del dato, que indica hasta que dato ya han recibido todos los nodos. La estructura de los "buffers" en el grupo de cooperación se ve como en la figura 6.10.

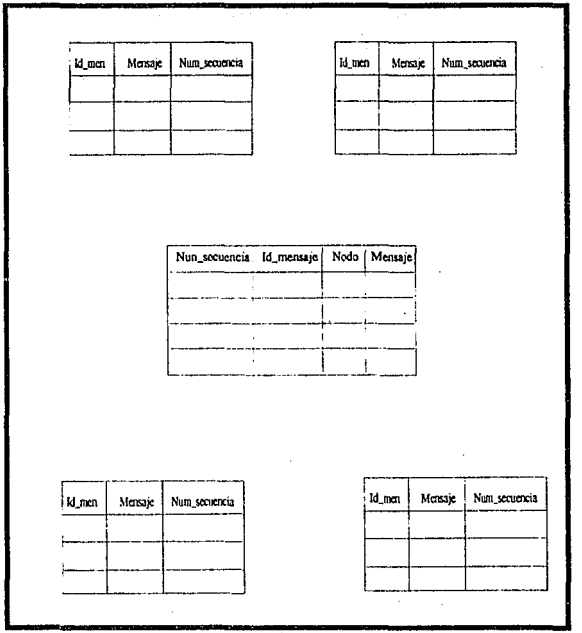

Figura 6.10: "Buffers" para el intercambio de mensajes

6.8.3 Descripción de la comunicación entre procesos.

Desde el punto de vista funcional, en cada nodo tenemos cuatro procesos: El proceso emisor de mensajes, el proceso receptor de mensajes, el proceso que denominamos proceso de aplicación y el proceso de usuario, el cual permite al usuario enviar datos.

Estos procesos, se ejecutan independientemente uno de otro, sin embargo en la implantación del protocolo cada nodo debe avisar al líder hasta el dato que ha recibido; pero eso solo lo conoce el proceso receptor, por lo tanto el proceso receptor tiene que enviarle un mensaje al emisor, de tal forma que en el momento que este envíe un dato al líder, se envíe este valor. La comunicación entre el proceso receptor y el emisor se efectúa a través de un "socket" de tipo UNIX.

Por otro lado, el nodo emisor envía un dato al líder para que lo difunda, pero cuando se envía, no se conoce el número de secuencia que el líder le asigna al dato enviado; por lo tanto no se puede almacenar en el "buffer" del nodo emisor hasta que se conoce el número de secuencia que le asigna el líder. Este número de secuencia se conoce hasta que el líder confirma la llegada del dato, pero este dato es recibido por el proceso receptor del nodo emisor. Para que el proceso receptor reconozca que el dato recibido es el que había enviado el proceso emisor, este necesita conocer el identificador del dato que le había asignado el emisor. Por lo tanto es necesario que el proceso emisor envié al receptor esta información. La comunicación entre el nodo emisor y el líder se efectúa utilizando "sockets" de tipo DATAGRAMA; la comunicación del emisor al receptor se hace mediante "pipes" y del receptor al emisor se hace mediante "sockets" de tipo UNIX.

Otros procesos que se comunican, son el proceso receptor con el proceso de aplicación; esto se hace en el momento que el proceso receptor ha recibido el dato esperado. Cuando se recibe el dato esperado, lo envía a la aplicación; esta comunicación se efectúa a través de un "pipe".

También se establece la comunicación entre el proceso emisor y el proceso de usuario para que desde el exterior se puedan difundir datos. Esta comunicación se hace utilizando "sockets" de tipo UNIX.

En el líder se tienen dos procesos importantes: El proceso que recibe los mensajes a difundir y el proceso que reparte las tareas, que también se comunica con el proceso de aplicación. En la figura 6.11 en la que denotamos como (usr) a los procesos de usuario, (envía y recibe) a los procesos emisor y receptor y como (apl) a los procesos de aplicación, se muestra el proceso descrito.

6.9 Resultados, observaciones y perspectivas.

El objetivo planteado desde el principio fue la especificación, validación e implantación de un protocolo por difusión tolerante a fallas.

Estos objetivos planteados se lograron totalmente y de hecho para efectuar las pruebas, se creo un ambiente distribuído que permite el intercambio de mensajes utilizando el protocolo implantado.

Como resultados, ya se menciono en secciones anteriores, que se obtuvo un protocolo eficiente y tolerante a ciertas fallas.

Como perspectivas considero importantes los siguientes puntos:

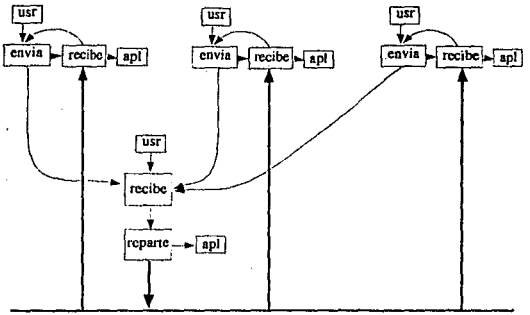

Figura 6.11: Esquema de comunicación entre los procesos.

- **Tolerancia a fallas** : En este aspecto considero que el protocolo se puede extender a tolerar fallas de un nodo y el líder a la vez; esto se lograría replicando la información en mas de un nodo. Si tenemos la información replicada por lo menos una vez, haría que el protocolo tolere fallas del líder y de otro nodo; este proceso se podría seguir extendiendo.

  Es claro que entre mas fallas tolere el protocolo, este se vuelve mucho más complejo, pero aumenta su seguridad.

- **Distribución** : Como se mencionó en el capítulo 2, los protocolos centralizados tiene ciertas desventajas. Como el protocolo que especificamos es de control centralizado, entonces una de las grandes desventajas es su capacidad de crecimiento modular. Para lograr este crecimiento, es necesario hacer una especificación usando un enfoque que considere control distribuído.

- En la programación considero que hubiese sido de gran ayuda el poder contar con una estructura en la que se pueda efectuar la comunicación entre procesos a través de una memoria compartida. En este caso solo se cuenta con un mecanismo de comunicación mediante el intercambio de mensajes y considero que no es lo mejor para hacer programación distribuída.

Capítulo 7

Conclusiones generales.

La confiabilidad de los sistemas en general se considera desde diferentes puntos de vista. Desde el punto de vista teórico, en cuyo caso se considera que un modelo de un sistema es correcto si el modelo no presenta inconsistencias y desde el punto de vista práctico decimos que un modelo es correcto si los resultados obtenidos están de acuerdo a lo que esperábamos.

Desde el punto de vista teórico basta demostrar que el modelo es consistente; esto se logra si utilizamos una herramienta formal para la especificación. En nuestro caso utilizamos las redes de Petri como herramienta de especificación formal y el comportamiento y la prueba se describieron las redes de Petri, para esto se utilizó los modelos extendidos y los modelos reducidos.

Si consideramos el punto de vista práctico, debemos garantizar resultados correctos y además que el sistema sea confiable, robusto y eficiente. En este caso se consideró la posibilidad de fallas y para lograr resultados correctos se definió el algoritmo y las reglas que.se deben seguir dependiendo de la situación que se presente: consideramos que al usar técnicas de más alto nivel nos hubiese reducido la complejidad del problema; sobre todo, si ofrece tanto el aspecto formal como el aspecto práctico

El objetivo de cualquier sistema es lograr correctés, confiabilidad, robustés y eficiencia y esto debe ser transparente al usuario. Pero muchas veces se debe sacrificar algunos de los aspectos para mejorar otros. En particular en los algoritmos por difusión debemos hacer un balance entre confiabilidad y eficiencia. Lograr un protocolo completamente confiable es imposible, pero si deseamos lograr un alto grado de confiabilidad tenemos que pagar un precio tanto por el número de mensajes como por el espacio de memoria, generando sistemas ineficientes.

En particular en el protocolo que especificamos hemos sacrificado cierta seguridad por eficiencia que comparado con el de Kaashoek y Tanenbum considerado hasta ahora, de los más eficientes resulta más seguro e igual de eficiente. Este protocolo se puede extender a uno más confiable pero esto implica sacrificio de eficiencia.

Una de las razones más importantes para el desarrollo de este tipo de trabajo es que en los Sistemas Distribuíos la replicación de datos es fundamental para poder tolerar fallas, pero la replicación no es tan simple, sobre todo porque necesitamos garantizar la consistencia de los datos replicados, de tal forma que el resultado de la consulta no dependa de la réplica consultada.

La replicación de información se ha tratado considerando dos tipos de fallas. En un caso se considera que la falla puede ser la partición de la red, en el otro caso, la replicación de datos es tratada utilizando protocolos de comunicación por difusión confiable, los cuales utilizan técnicas para ordenar eventos logrando un alto grado de transparencia de las fallas, aunque no consideran la posibilidad de que la red se particione.

En este caso nosotros consideramos el segundo enfoque, y la confiabilidad fué tratada desde el punto de vista formal y desde el punto de vista práctico.

Uno de los problemas a los cuales nos enfrentamos en el aspecto formal fué la necesidad de extender esta herramienta para poder tener más libertad de modelado, pero esto nos limitó en la validación del modelo. Consideramos que requerimos de herrramientas de alto nivel tanto en el aspecto formal como en el aspecto práctico que nos ofrescan transparencia.

En el aspecto práctico considero que se tienen ciertas herramientas pero no son suficientes para lograr programación eficiente; quizá en este caso hubiese sido mucho más agradable utilizar programación orientada a objetos o herramientas que permitan la distribución más fácilmente.

Bibliografía

[aDLS88] Douglas E. Comer an David L. Stevens. *Internetworking with TCP/IP, vol. I: Design, Implementation and Internals.* Prentice-Hall, Inc., 1988.

[aDLS91] Douglas E. Comer an David L. Stevens. *Internetworking with TCP/IP, vol. II: Design, Implementation and Internals.* Prentice-Hall, Inc., 1991.

[AJSn88] Vasco L. B. Freitas Alexandre J.T. Santos and José C.M. neves. Temporal logic and concurrent login programming in protocol specification, varification and prototyping. *Research into Networks and Distributed Applications*, 1988.

[AMC91] M. balbo A. Marsam and G. Chiola. An introduction to generalized stochastic petri nets. *Microelectron, reliability*, 4(31), 1991.

[And91] Gregory A. Andrews. Paradigms for proces interaction in distributed programs. *ACM computing surveys*, 23(1), March 1991.

[BD87] S. BUDKOWSKI and P. DEMBINSKI. An introduction to estelle: A language for distributed systems. *Computer Networks and ISDN systems*, 14, 1987.

[BK92] E. Best and M. Koutny. Petri net semantics of priority systems. *Theoretical Computer Science*, 96(1), April 1992.

[BOC90] Gregor V. BOCHMANN. Protocol specification for (osi). *Computer Networks and ISDN Systems, The International Journal Of Computer and Telecomunications Networking*, 18(3), April 1990.

[BU91] Barry S. Bosik and Umit Uyan. Finite state machine based formal methods in protocol conformance testing: from theory to implementation. *Computer networks and ISDN Systems*, 1991.

[CM84] Jo-Mei CHANG and N.F. MAXEMCHUK. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3), August 1984.

[Cri88] J. M. Crichlow. *An introduction to Distributed and Parallel Computing.* Prentice-Hall, Inc., 1988.

[EK89] S. Moran E. Korach, S. Kutten. A modular tecnique for the design of eficient distributed leader finding algorithms. *TECNION- Israel Institute of Technology, Computer Science Department; Tecnical Report 581*, 1989.

[Eve79] S. Even. *Graph Algorithms*. Computer Science Press., 1979.

[FCP71] S. Even F. Commoner, A. W. Holt and A. Prueli. Marked directed graphs. *Journal of Computer and System Sciencs*, 5(5), October 1971.

[FLL88a] M. Chu F.J. Lin and M.T. Liu. Protocol verification using reachability analysis: The stata space explosion problem and relief strategies. *ACM computing surveys*, 1988.

[FLL88b] P.M. Chu F.J. Lin and M.T. Liu. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. *ACM Transaction on Computer Systems*, 1988.

[GEH85] NARAIN GEHANI. *C: An Advanced Introduction*. Computer Science Press, 1985.

[Gib85] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[HAL91] FRED HALSALL. *Data Comunications, Computer Networks and OSI*. Addison-Wesley, 1991.

[Heu91] Carlos A. Heuser. *Modelagem Conceitual de Sistemas*. V EBAI, 1991.

[HH91] Vassos Hadzilacos and J. Y. Halpern. Massage-optimal protocols for byzantine agreement. *Research Report(8253(75447))*., July 23 1991.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer protocols*. Prentice Hall, 1991.

[JJS84] A.E.Baratz J.M. Jaffe and A. Segall. Subtle design issues in the implementation of distributed, dynamic routing algoritms. *Research Report(No.47933) of IBM.*, Augost 20 1984.

[KP87] B. W. Kernigan and R. Pike. *El entorno de programación UNIX*. Prentice-Hall, Inc., 1987.

[KR87] B. W. Kernigan and D. M. Ritchie. *El lenguaje de programación C*. Prentice-Hall, Inc., 1987.

[Lam78a] Leslie Lamport. The implementation of reliable distributed multiproces systems. *North-Holland Publishing Company Computer Networks*, 2, 1978.

[Lam78b] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Comunications of the ACM*. 21(7), July 1978.

[LL90] Leslie LAMPORT and Nancy LYNCH. Distributed computing: Models and methods. *Handbook of Theoretical Computer Science*
Volume B
Formal Models and Semantics., 1990.

[MB84] M. A. Marsam and GianFranco Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocesor systems. *ACM Transactions on Computer Systems*, 2(2), May 1984.

[MFK89] Susan Flynn Hummel M. Frans Kaashoek, Andrew S. Tanenbaum. An eficient reliable broadcast protocol. *ACM Operating Systems Review*, 23(4), April 1989.

[mic90] SUN microsystems. *Sun Manual: Network Programming Guide.* SUN microsystems, 1990.

[Mil80] Robin Milner. *A Calculus of Communicating Systems.* Lecture Notes on Computer Science, Springer Verlag, 1980.

[MS92] Univ. de Zaragoza Manuel Silva. Introducing petri nets, draft. *DRAFT*, 1992.

[Mul89] S. Mullender. *Distributed Systems.* ACM press. 1989.

[NT91] Akihito NAKAMURA and Makoto TAKIZAWA. Reliable broadcast protocol for selectively partially ordering pdu's. (spo protocol). *Computer IEEE*, 1991.

[PEH90] Bjorr PEHRSON. Protocol verification for (osi). *Computer Networks and ISDN Systems, The International Journal of Computer and Telecomunications Networking*, 18(3), April 1990.

[PER90] Dominique PERRIN. Finite automata. *Handbook of Theoretical Computer Science*
Volume B
Formal Models and Semantics., 1990.

[Pet77] James L. Peterson. Petri nets. *Computing Surveys*, 9(3), September 1977.

[R.90] Apt K. R. Login programing. *Handbook of Theoretical Computer Science*
Volume B
Formal Models and Semantics., 1990.

[Ste89] W. Richard Stevens. *UNIX Network Programming.* Prentice-Hall, Inc., 1989.

[Tan81] Andrew S. Tanenbaum. *Computer Networks.* PRENTICE-HALL, INC., 1981.

[Tan88] Andrew S. Tanenbaum. *Computer Networks.* PRENTICE HALL, second edition edition, 1988.

[TB88] A. Pisano T. Bologenesi, D. Latella. Towards a graphic syntax for lotos. *Research into Networks and Distributed Aplications*, 1988.

[WS90a] Ph.D. William Stallings. *Handbook of Computer Comunications, Standards: Local Area Networks Standards*, volume 2. HOWARD W. SAMS & COMPANY, second edition edition, 1990.

[WS90b] Ph.D. William Stallings. *Handbook of Computer Comunications, Standards: Local Area Networks Standards*, volume 3. HOWARD W. SAMS & COMPANY, second edition edition, 1990.