INDICE

Capítulo 1: Evolución de los Protocolos
Ingeniería de Protocolos y Servicios

1.1. COMIENZOS DE LOS PROTOCOLOS DE COMUNICACIONES

• 458 A.C. *Agamenón:* caída de Troya por Atenas
- Distancia aprox. 500 Km.
- Información preestablecida de antemano.

• Siglo II A.C. *Polibio:* en Grecia se empleaba un método a base de antorchas. Los mensajeros podían deletrear palabras y con ello enviar mensajes. Si el mensajero sostenía tres antorchas en su mano izquierda y dos en su derecha, el receptor era capaz de interpretarlo como cierta letra. Cambiando la cantidad de antorchas, se cambiaba la letra.

El método de Polibio no es considerado por muchos autores como verdadera criptografía, pues su interés no reside en ocultar la información, sino en transmitirla de un modo más eficaz. Aún así es interesante su estudio, pues ciertas ideas se usan en criptografía.

Los métodos de sustitución se basan en asignar a cada letra otro ente, que puede ser también otra letra, o un número o un símbolo especial. La cifra de Polibio es históricamente la primera que emplea métodos de sustitución.

Para explicar el funcionamiento en castellano hay que recurrir a un truco, debido a que en dicho idioma se emplean más de 25 letras, cosa que en latín y en griego no ocurre. Para codificar un mensaje primeramente se forma la siguiente tabla:

	A	**B**	**C**	**D**	**E**
A	a	b	c	d	e
B	f	g	h	i	j
C	k	l	m	n	o
D	p	r	s	t	u
E	v	w	x	y	z

La letra *a* se cifrará como AA, la *b* como AB, ... Se ha eliminado la letra *q*, lo cual no redunda en el contenido del mensaje, siempre que se sustituya dicha letra por la *k.* Tampoco se ha incorporado la *ñ* pues se suele llevar mal con los ordenadores.

Ejemplo:

Texto claro: cifradepolibio
Texto cifrado: ACBDBADBAAADAEDACECBBDABBDCE

Una de las ventajas del cifrado de Polibio es que emplea únicamente 5 letras para escribir cualquier mensaje, pero tiene un grave problema: el mensaje cifrado tiene el doble de longitud que el texto claro.

En resumen:
- Nuevo método de señalización formado por 2 conjuntos de antorchas (=5^2 símbolos).
- Alfabeto formado por 5 bloques con 5 letras cada uno.
- El establecimiento de la comunicación se iniciaba con 2 antorchas.

• 1793 *Claude Chappe,* Ingeniero francés, usó el término *telegrafía* para designar una serie de estaciones equipadas con brazos que formaban un sistema de torres de semáforos y telescopios mecánicos. Cada estación se situaba a unos 16 km; de forma que se pudieran enviar mensajes en un código predeterminado.

En las torres, sobre una plataforma se montaba un mástil de madera, en cuyo extremo superior se colocaba horizontalmente un travesaño (denominado regulador), que podía modificar su posición mediante cuerdas y poleas. En el extremo del brazo horizontal había otros brazos verticales también móviles. Así se podían conseguir un gran número de figuras geométricas que desde la torre siguiente eran visualizadas por medio de un anteojo.

Ante el éxito de esta primera línea se creó en Francia una extensa red telegráfica óptica, llegando a tener hasta 556 semáforos.

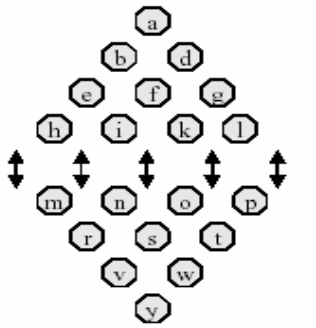
En la actualidad, el término de *telegrafía* se aplica a todos los sistemas de transmisión de mensajes escritos o imágenes por medio de señales eléctricas desde un lugar a otro por medio de conexiones alámbricas o inalámbricas.

• 12 junio 1837 *William Cooke*: científico inglés que realizó estudios de anatomía y fisiología en Paris. Abandonó sus estudios de medicina después de ver un modelo de telégrafo del barón Schilling, y aplicó todas sus energías en el desarrollo de un nuevo telégrafo. En menos de tres semanas acabó su primer galvanómetro, que consistía en tres agujas magnéticas rodeadas de múltiples bobinas, dirigidas mediante tres circuitos diferentes. El movimiento de las bobinas bajo la acción de la corriente eléctrica permitía 36 señales diferentes, correspondientes a las letras del alfabeto y los números.

Para perfeccionar su invento se trasladó a Londres, y su encuentro con Wheatstone, quien también trabajaba en la construcción de un telégrafo eléctrico, le permitió a Cooke avanzar de forma decisiva en su invención.

La patente compartida de ambos científicos se realizó el *12 de junio de 1837*, constaba de cinco agujas con las que se podían efectuar 20 combinaciones que correspondían a los signos más usados. Posteriormente lo perfeccionaron con solo dos agujas y así se lo presentaron a las compañías ferroviarias. Antes del final de ese mismo mes, ya trabaja en la instalación de una línea telegráfica entre

las estaciones de Euston y Camden Town, a lo largo de la línea entre Birmingham y Londres.
Tantas eran las solicitudes que en 1845 empezó su andadura la compañía *Electric Telegraph Company*, de la cual Cooke fue nombrado presidente.



Una de las razones por las que se usó el telégrafo eléctrico de *Cooke* en el ferrocarril fue para proteger tramos peligrosos (túneles, ...). Lo que nadie se esperaba eran los numerosos accidentes por malentendidos.

Ejemplo: Túnel de Clayton (protocolo inadecuado)

• En cada extremo del túnel había un telégrafo, un semáforo y un operador.

• Los semáforos pasaban a rojo automáticamente cuando pasaba un tren.

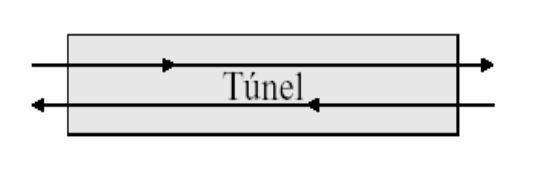
• Si falla el semáforo, se dispara la alarma y se usan banderas roja y blanca para señalizar.

• Para poner el semáforo en verde, el operador debía comprobar antes que el tren que acababa de entrar hubiese salido del túnel.

• Hay 2 vías, una en cada sentido.

• Telégrafo con los mensajes predefinidos:
- Tren en el túnel
- ¿Ha salido ya el tren?
- Túnel libre.
- Túnel ocupado.

• Funcionamiento: Cuando un tren entra en el túnel se manda “tren en túnel”. Cuando el tren sale (si sale) se responde “Túnel libre”. Entonces el operador puede poner el semáforo en verde de nuevo.



• Agosto 1861

- El tren se pasa el semáforo, y éste no se pone rojo, por lo tanto, suena la alarma. Se transmite “tren en túnel” y se utiliza bandera roja para parar trenes.

- Un segundo tren (rápido) pasa el semáforo verde, pero consigue en el último momento ver la bandera roja.

- Un tercer tren, viendo la bandera roja, para a la entrada del túnel.

- El operador transmite (2º tren) “Tren en túnel”. (Hay 2 trenes en el túnel y este es un evento no contemplado por el protocolo. Por tanto hay 2 mensajes seguidos “Tren en túnel” que no tienen sentido).

- El operador transmite “¿Ha salido ya el tren?”

- Al salir el primer tren se responde “Túnel libre”.

- El operador no puede saber si ha salido uno o dos trenes. No sabe si debe esperar 2 mensajes seguidos de “Túnel libre”.

- El operador decide que ambos trenes ya deben haber salido del túnel, permitiendo la entrada del tercer tren.

- El conductor del segundo tren vio la bandera roja. Paró en medio del túnel y, al cabo de un tiempo decidió dar marcha atrás y salirse del túnel para no correr riesgos.

- Resultado: 21muertos y 176 heridos.

1.2. LAS PRIMERAS REDES

A continuación se muestra su evolución histórica:
• 1851: Bolsa de París y Londres interconectadas por telégrafo. Introducción del código Morse moderno.

• 1875: más de 300.000Km. de línea telegráfica.

• 1925: Redes "Telex" (*telegraph-exchange*) en pleno funcionamiento.
Paralelamente:
- Radio: *Guglielmo Marconi* 1897 -radiotelégrafo-
- Teléfono: *Alexander Graham Bell* 1876

PROTOCOLOS MAESTRO-ESCLAVO

Aparecen en la década de 1950 y su escenario típico es en la comunicación de ordenadores con periféricos. El inicio de la comunicación siempre parte del maestro, que es el que se encarga también de la transferencia de datos, la recuperación frente a errores, la sincronización y la gestión de la conexión. La comunicación entre maestro y esclavo se realiza mediante un sondeo (*polling).*

En un sistema de *polling,* se controlan las comunicaciones en una red dirigidas por un ordenador central, y se organizan de manera que es éste el que les pregunta secuencialmente a todos los ordenadores de la red si tienen algo que comunicar, y les insta a que lo hagan en caso afirmativo. Ningún otro componente de la red toma en ningún momento la iniciativa de la comunicación.

La Ingeniería de Protocolos, deberá resolver la secuencia de escenarios de comunicaciones que se darán a partir de la evolución del modelo maestro-esclavo.

A continuación se muestra dicha evolución histórica:
• 1946: Universidad de Pensilvania: ordenador ENIAC (30 ton.)

• 1950: Las ejecuciones automatizadas en *Mainframes* generan aumento en la robustez de los nuevos protocolos.
Configuración de un *mainframe* y varios periféricos "tontos".

• 1956: Primeras transmisiones de datos entre ordenadores a larga distancia a través de línea telefónica.

• Años 60: *mainframes* se interconectaban (redes de datos): La relación maestro-esclavo perdía el sentido.

PROTOCOLOS *PEER TO PEER*

Son los protocolos más antiguos y elementales usados para la comunicación mediante una línea de datos entre dos ordenadores.
En una red peer-to-peer no existe una jerarquía entre los ordenadores. Todos los ordenadores son iguales y además son conocidos como pares (peers). Normalmente, cada ordenador funciona como un cliente y como un servidor, y no hay uno asignado a ser un administrador responsable de la red en su totalidad. El usuario de cada ordenador determina cuales de sus datos serán compartidos en la red.

El **tamaño** original de las redes Peer-to-Peer es pequeño, normalmente, menos de 10 ordenadores en la red.

El **coste** de las redes Peer es relativamente bajo, debido a que cada ordenador funciona como un cliente y un servidor, y no hay necesidad de un potente servidor central, o de los otros componentes requeridos para una red de alta capacidad.

El **software** de red no necesita tener el mismo nivel de prestaciones y seguridad que el software diseñado para redes de servidor dedicado.
Uno de los problemas que surgen, es el de la negociación sobre quién es el responsable de los distintos aspectos de la comunicación.

La Ingeniería de Protocolos debe resolver la utilización compartida de los recursos en una red de pares.

A continuación se muestra su evolución histórica:
• 1961: *American Airlines: SABRE system* (reservas de vuelos)

• 1969: *U.S. Dept. of Defense*: ARPA (*Advanced Research Projects Agency*).

• 1985: 1.200 nodos

• 1987: 25.000 nodos (Internet)

• Ag. 2001: 120.205 millones de hosts

1.3. PROTOCOLO VISTO COMO LENGUAJE

• Abril 1967: se produjo por vez primera el uso del término “protocolo” aplicado a un procedimiento de comunicación de datos

Protocolo: acuerdo sobre cómo intercambiar información en un sistema distribuido.

Define:

- Formato de mensajes válidos (sintaxis)
- Reglas de intercambio de datos (gramática)
- Vocabulario de mensajes válidos con su significado (semántica)

Debe ser consistente y completo: en cualquier circunstancia debe describir de manera no ambigua qué está permitido y qué está prohibido.

- Consistente: no hace nada incorrecto.
- Completo: considera todos los casos posibles.

1.4. ESTANDARIZACIÓN DE PROTOCOLOS

Existen bastantes entidades dedicadas a la estandarización de protocolos. Dos importantes son:

- ISO (*International Standards Organization*).
- CCITT (*Comité Consultatif International Télégraphique et Télephonique*), actualmente parte de la ITU (*International Telecommunications Union*).

Estándar Internacional no significa que algo funcione, significa que mucha gente que se ha puesto de acuerdo en utilizarlo cree que funciona. Es necesario disponer de metodología adecuada para diseñar, describir y comprobar un protocolo.

Para estandarizar es necesario un formato común para la especificación de protocolos: Lenguaje de Especificación de Protocolos.

• Técnicas de Descripción Formal:

• Fuerte base matemática: soportan herramientas para verificar, validar y probar los protocolos, o analizar la conformidad de las implementaciones con la especificación correspondiente.
• Abstractos: especificaciones independientes de la implementación.
• Soporte a la especificación y al diseño de protocolos: permiten definir protocolos de manera no ambigua, consistente y completa.

Hay 3 (referidos como “las 3 FDT “-Formal Description Technique-) de interés:

• SDL (*Specification and Description Language*).
Recomendación Z101-Z104 del CCITT. Pensado originalmente para diseñar sistemas de telecomunicaciones.
Dispone de 2 versiones equivalentes: Gráfica y textual.

• Lotos (*Language of Temporal Ordering Specifications*).
Álgebra de procesos basada en CCS (*Calculus of Communication Systems*). IS8807. ISO TC97/SC21/WG21

• Estelle. IS9074. ISO TC97/ SC21/WG21.

• Existe un problema en el salto que hay desde la especificación hasta la implementación final. Una posible solución será usar herramientas automáticas, tales como Telelogic TAU SDL, técnicas semiformales, como Promela, o implantación manual a través de distintos lenguajes de programación.

1.5. RECAPITULACIÓN: Tareas del diseño de un protocolo

• Seleccionar un medio de transmisión y una tecnología.

• Definir y describir un vocabulario.

• Definir la codificación de los mensajes.

• Definir la evolución de una transmisión (negociación, inicio, terminación, reinicialización, ...), es decir, las reglas de procedimiento.

• Tener en cuenta las situaciones críticas, y tratar de evitar los errores, sobre todo los más complejos. Buscar un conjunto de reglas consistente y completo.

• Estructurar el protocolo en capas si es necesario.

Todo protocolo es incorrecto
hasta que se demuestre lo contrario.

Capítulo 2: Estructura de los Protocolos
Ingeniería de Protocolos y Servicios

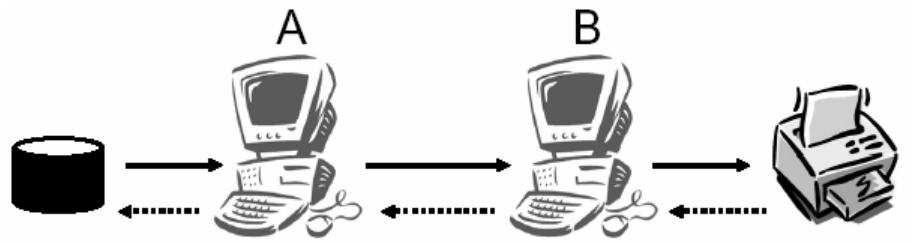
2.1. INTRODUCCIÓN

¿Qué es un protocolo?
- Es un conjunto de normas y reglas organizadas, convenidas de mutuo acuerdo entre todos los participantes y previas a una comunicación.

- Su misión es hacer que la comunicación entre todos los ordenadores de una red que están usando ese protocolo sea compatible. Estos protocolos son estandarizados por organizaciones, y los fabricantes lo tienen en cuenta para la realización de dispositivos tele-informáticos.

- La gramática del protocolo debe ser consistente y completa, es decir, bajo cualquier circunstancia las reglas deben dictar de manera no ambigua lo que está permitido y lo que está prohibido.

Por ejemplo, para imprimir un documento del disco de A en la impresora de B:

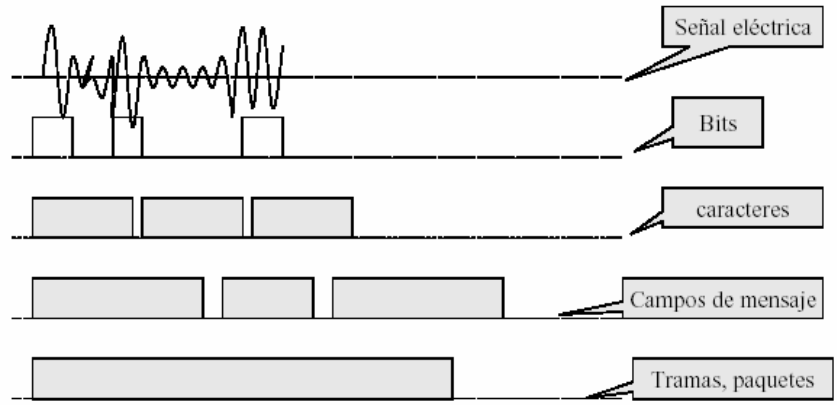


- La información irá en un sentido y la comunicación en dos.
- Las señales de control: Previo acuerdo del significado.

Por tanto, a todas las reglas, formatos y procedimientos preacordados entre A y B se le llamará protocolo.

Existen acuerdos previos sobre:
- Inicialización y terminación de un intercambio de datos.
- Sincronización de emisores y receptores.
- Detección y corrección de errores de transmisión.
- Formateo y codificación de los datos.

Nótese que cada punto puede separarse en varios niveles de abstracción.



Durante el diseño de protocolos, existen dos tipos de errores que son difíciles de evitar:

1. Diseño de un conjunto incompleto de reglas.
2. Diseño de reglas contradictorias.

2.2. LOS 5 ELEMENTOS DE UN PROTOCOLO

La especificación de todo protocolo debe constar de 5 partes distintivas.
Toda especificación debe incluir:
1.- El *Servicio* que proporciona el protocolo.
2.- Las *Suposiciones* sobre el entorno donde se ejecuta el protocolo.
3.- El *Vocabulario* de los mensajes usado para implementar el protocolo.
4.- La *Codificación* (formato) de los mensajes del vocabulario del protocolo.
5.- Las *Reglas de procedimiento* que controlan la consistencia del intercambio de mensajes.

El último elemento es el más difícil de diseñar y de verificar.

2.3. EJEMPLO: PROTOCOLO DE LYNCH (1968)

ESPECIFICACIÓN DEL SERVICIO:
- Transferir archivos como secuencia de caracteres por la línea telefónica evitando errores de transmisión, suponiendo que pueden detectarse todos los errores.
- Es una transferencia de archivos full-duplex.
- Se envían reconocimientos positivos y negativos para el tráfico de A a B mediante la línea de B a A (y viceversa).
- Cada mensaje tiene dos partes, una de mensaje, y otra de control que se aplica al tráfico en el canal contrario.

SUPOSICIONES SOBRE EL ENTORNO:
- El entorno consta de dos usuarios del servicio y un canal de transmisión.
- Cada usuario pide un archivo y espera la vuelta.
- Se supone que el canal distorsiona arbitrariamente el mensaje, pero no pierde, inserta, duplica, ni reordena los mensajes.
- Se parte de la existencia de un módulo de nivel inferior que atrapa las distorsiones y reparte mensajes no distorsionados de tipo “err”.

VOCABULARIO DEL PROTOCOLO:
Se definen 3 tipos de mensaje:
• ack: mensaje combinado con reconocimiento positivo.
• nack: mensaje combinado con reconocimiento negativo.
• err: mensaje combinado con error de transmisión.

El vocabulario puede ser expresado como un conjunto:

V = { ack, nack, err }

FORMATO DE LOS MENSAJES:
- Cada mensaje consta de un código de control que identifica el tipo de mensaje y un campo de datos con el código del carácter (suponemos que ambos son de tamaño fijo).

- La forma general de cada mensaje puede ser representado de manera simbólica como una simple estructura de dos campos:

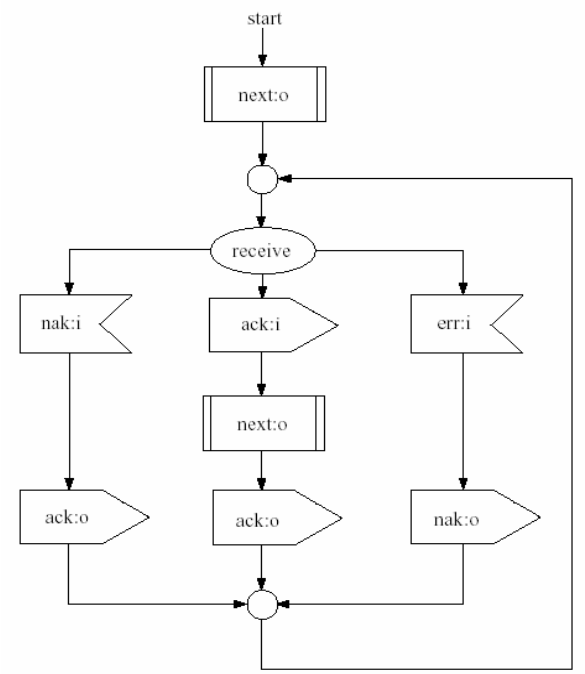
{etiqueta de control (ack, nack, err), datos};

- En C, de manera más detallada tendríamos:

```
enum control { ack, nack, err };
struct message {
enum etiqueta de control;
unsigned char datos;
};
```

REGLAS DE PROCEDIMIENTO
De manera informal:
1.- Si la recepción anterior no tenía errores, el próximo mensaje en el canal contrario llevará un reconocimiento positivo (ack).
Si la recepción tuvo errores, llevará un reconocimiento negativo (nack).
2.- Si la recepción previa llevaba un reconocimiento negativo, o la recepción anterior fue errónea, se retransmitirá el mensaje anterior; de otro modo, se prepara otro mensaje para una nueva transmisión.

De manera formal, se pueden usar las siguientes técnicas:
- Diagramas de flujo (subconjunto de SDL)
- Diagramas de transición de estados.
- Expresiones algebraicas, etc.



ERRORES DE DISEÑO

1. La transferencia de datos en un sentido, sólo puede continuar si se lleva a cabo la transferencia de datos en sentido contrario.

Posible solución: cuando no existan datos que enviar en un sentido, enviar mensajes de relleno.

2. Las dos reglas de procedimiento especifican la transferencia normal de datos, pero no precisan nada acerca de los procedimientos de inicio y terminación.

Posible solución: uno de los dos procesos inicia enviando un mensaje falso de error.
Problema cuando los dos inician simultáneamente (Figura 1). El procedimiento de terminación, sin embargo, requiere de mensajes extras de control.

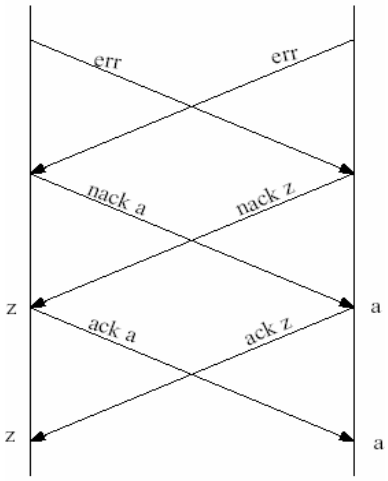


Figura 1. Deficiencias en el diseño

3. El receptor no es capaz de decidir si el elemento recibido correctamente y temporalmente almacenado en *i*, debe ser o no aceptado.

Agregando las siguientes reglas de aceptación:

- Los datos en un mensaje ack o nack son aceptados.
- Los datos en un mensaje err no son aceptados.

Las reglas anteriores no resuelven el problema. Considere la secuencia de eventos de la Figura 2. Debemos hacer notar que, a pesar de que la especificación es sencilla, es sumamente difícil encontrar el error.

- La especificación del protocolo anterior es simple.
- La descripción informal es convincente.
- Pocos dudarían de la corrección del protocolo.

Pero:
- La especificación del protocolo está incompleta.
- Su implementación, por muy bien implementada que estuviera, generaría errores durante el intercambio de información.

Conclusión: Aun para el protocolo más sencillo, una buena disciplina de diseño y herramientas automatizadas para el análisis son indispensables.

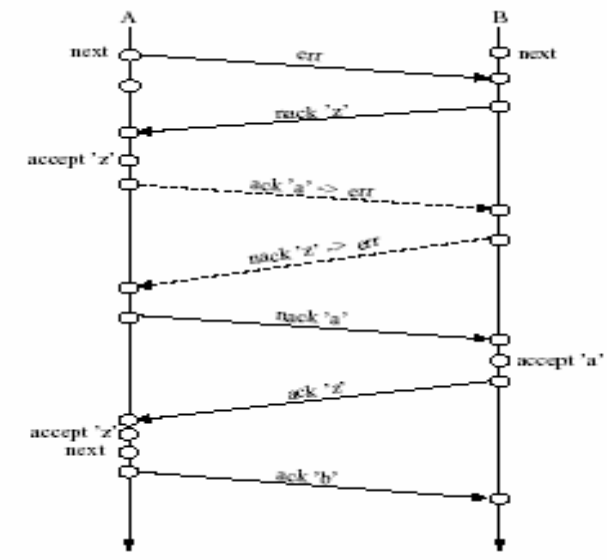


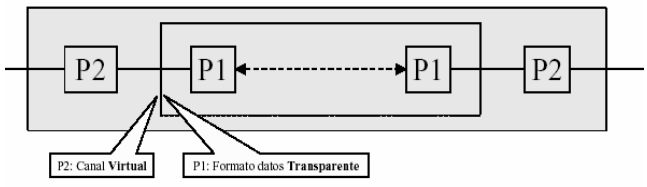
Figura 2. Diagramas de tiempo

2.4. SERVICIO Y ENTORNO

Servicio: Qué ofrece un protocolo.
Entorno: En qué se basa un protocolo.

Ejemplo
Protocolo que codifique caracteres en 7 bits y detecte errores añadiendo un bit de paridad.
Servicios: codificación de datos y detección de errores

Separación en:
• Módulo codificador/decodificador (P2)
• Módulo añadir/comprobar paridad (P1)



Virtual: es algo que parece que existe pero, en realidad, no existe.
Transparente: es algo que, en realidad, existe pero que parece que no existe.

• Cada capa implementa un protocolo y ofrece un servicio
• P1: Datos 8 bits
• P2: Datos 7 bits

En lugar de usar el hardware de red directamente, las redes usan módulos de software que ofrecen interfaces de alto nivel para desarrollar aplicaciones.

Las familias de protocolos aparecen: porque, en lugar de tener un solo protocolo gigante que especifique todos los detalles de todas las formas posibles de comunicación, el problema de la comunicación entre computadores se divide en subpartes. Así, los protocolos son más fáciles de diseñar, analizar, implementar, y probar. Esta partición del problema da origen a un conjunto de protocolos relacionados llamados *Familias de Protocolos*.

Una de las herramientas más importantes en la Ingeniería de Software en general es el modelo de capas. Esto consiste en una división del problema de la comunicación en partes llamadas capas. La familia de protocolos puede diseñarse especificando un protocolo que corresponda a cada capa.

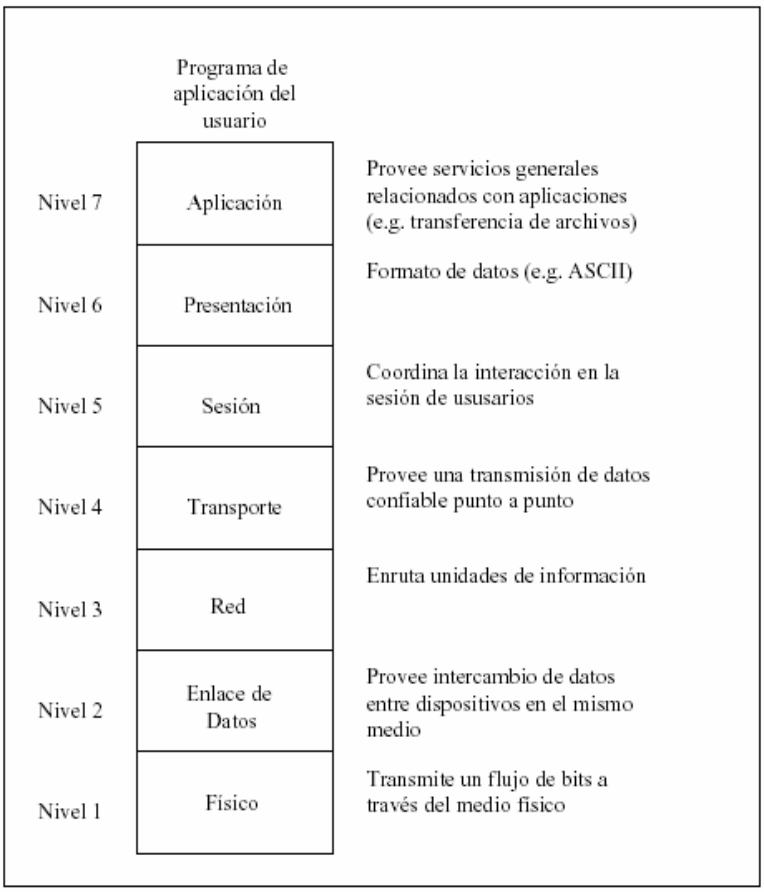
La Organización Internacional de Normalización ISO (*International Standards Organization)* definió uno de los modelos más importantes y más utilizado, el modelo de siete capas.

Las ventajas que ofrece un diseño jerarquizado son:

- Cada capa realiza un conjunto de funciones, resolviendo un problema diferente de la comunicación.
- Cada capa se sustenta en la capa inmediatamente inferior.
- Cada capa proporciona servicios a la capa inmediatamente superior.
- Los cambios en una capa no implicarán cambios en las otras capas.

La comunicación física se lleva a cabo entre las capas de nivel 1.

Modelo de Referencia OSI (*Open Systems Interconnection*) de la ISO *(International Standards Organization).*



Nivel Físico

• Contiene todas las funciones relacionadas con la transmisión de bits sobre una conexión física.
• Especifica el medio mecánico (cable, fibra, radio-enlace, conectores asignaciones de pines, conmutadores, ...).
• Especifica la codificación de bit.
• Esconde estos detalles a capas superiores.

Nivel de Enlace

• Utiliza los servicios del nivel físico.
• Corrección (tratamiento) de errores.
• Multiplexación de flujos de datos.
• Control de flujo punto a punto.
• Unidad de datos: Trama.
• Convierte el canal físico en un canal fiable.

Nivel de Red
• Direccionamiento y encaminamiento.
• Evitar cuellos de botella.
• Reducción de la congestión.
• Iniciar y terminar conexiones.

Nivel de Transporte
• Interconecta procesos de usuario de manera transparente a través de una red.
• Control de flujo extremo a extremo.

Nivel de sesión
• Mecanismos para controlar el diálogo entre las aplicaciones de los sistemas finales.
• Sincronización de puntos de comprobación.
• Muchas aplicaciones no necesitarán este nivel.
• Determina el tipo de servicio que se proporciona al usuario:
- Tipo de diálogo
- Agrupamiento
- Recuperación

Nivel de presentación
• Define el formato de los datos que se van a intercambiar entre las aplicaciones y ofrece un conjunto de servicios de transformación de datos.
• Define la sintaxis usada entre entidades de aplicación y proporciona los medios para la selección y modificación de la representación empleada.
• Codifica datos en modo estándar (reales, enteros, caracteres, ...) y realiza funciones de compresión y cifrado de datos.

Nivel de aplicación
• Proporciona un medio para que los procesos de aplicación accedan al entorno OSI.
• Funciones de administración y mecanismos útiles para admitir aplicaciones distribuidas.
• Aquí residen las aplicaciones de uso general:
- Terminales virtuales de red.
- Transferencia de archivos.
- Correo electrónico.
- Búsqueda de directorios.
- .....

Surgieron críticas hacia el modelo OSI,
- Momento poco adecuado:
  - Tardó mucho en desarrollarse.
- Tecnología inadecuada:
  - Algunos niveles vacíos, otros muy densos.

- Muy dependiente de la arquitectura SNA de IBM.
- Muy complejo, difícil de implementar e ineficiente.

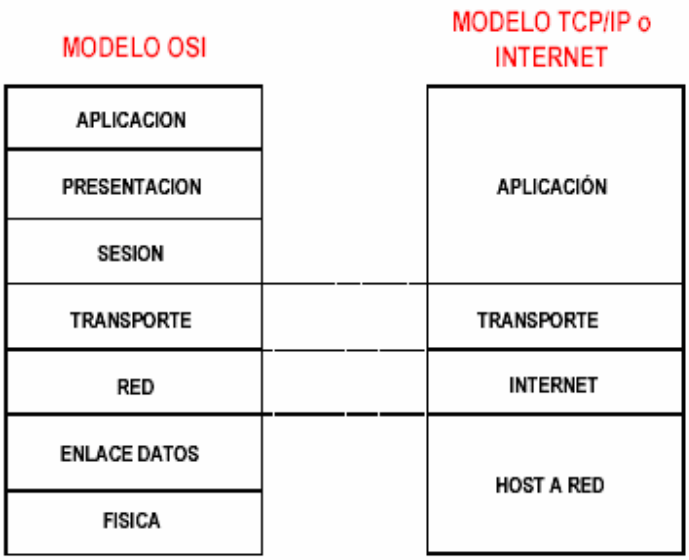
- Implementaciones inadecuadas:
  - Enormes y lentas.

- Política inadecuada:
  - Muy ligado a instituciones gubernamentales.

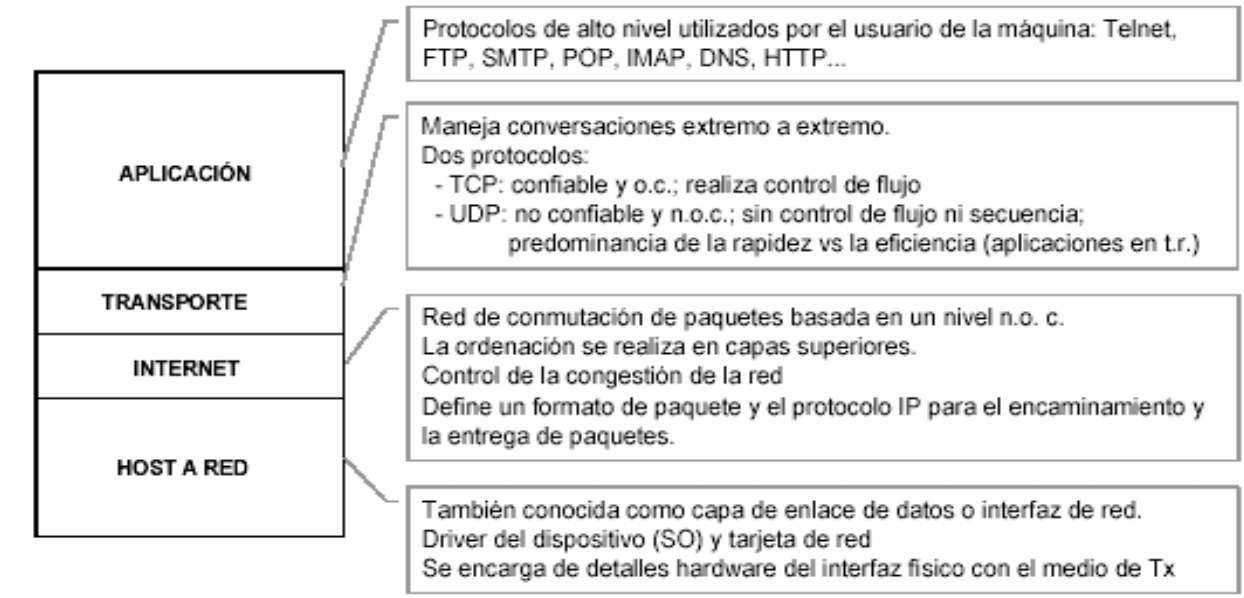
MODELO TCP/IP

• ARPAnet, red experimental del DoD *(Department of Defense)* americano, pasó a usarse en ambiente universitario: sobre líneas telefónicas alquiladas.
• Se unieron redes satélite y radio y aparecieron los primeros problemas de interconexión.
• Se crea el modelo de referencia TCP/IP en 1974
- Capacidad de conexión de múltiples redes de una manera sencilla.
- Exigencia de permanencia de la comunicación mientras funcionasen los hosts extremos.
• En los 90, ante la no implementación de OSI, el DoD ordenó la implementación de TCP/IP en todas sus adquisiciones.

COMPARATIVA DE CAPAS: MODELO OSI vs MODELO TCP/IP



CAPAS DEL MODELO TCP/IP



Surgieron críticas hacia el modelo TCP/IP,

• A la hora de implementar no distingue claramente servicio, interfaz y protocolos
  - Mala guía para diseño de nuevas redes

• No es un modelo general
  - No describe cualquier pila de protocolos
  - Se trata de una implementación concreta

• No se distinguen las capas física y de enlace de datos
  - No es un modelo apropiado a seguir

COMPARATIVA MODELO OSI vs TCP/IP

¿En qué se parecen?

- Describen una arquitectura jerárquica en niveles.
- La funcionalidad de las capas guardan “cierta” correspondencia.

¿En qué se diferencian?

- OSI se fundamenta en los conceptos de Servicios, Interfaces y Protocolos, mientras que en TCP/IP se obvian.
- En OSI se ocultan mejor los protocolos → mayor modularidad e independencia.
- OSI se desarrolló teóricamente antes de la implementación de los protocolos, mientras que en TCP/IP primero se implementaron los protocolos y el modelo no era más que su descripción.
- La cantidad de capas de cada modelo es diferente en ambos.

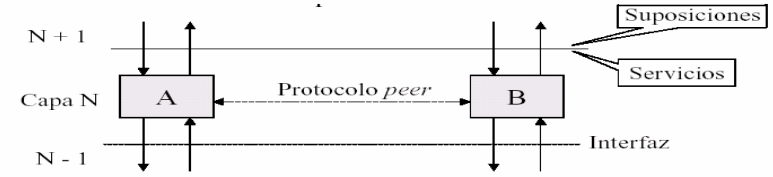
- En OSI, a nivel de red se permite comunicación orientada a conexión y no orientada a conexión y a nivel de transporte sólo orientada a conexión.
- En TCP/IP, a nivel de red se permite sólo la comunicación no orientada a conexión y a nivel de transporte se permiten ambos.

ESTRUCTURACIÓN EN CAPAS

¿Por qué usamos un diseño estructurado?
- Reduce la complejidad del desarrollo.
- Estandariza interfaces.
- Facilita la técnica modular.
- Asegura la interoperabilidad de la tecnología.
- Acelera la evolución.
- Simplifica la enseñanza y el aprendizaje.

En resumen:

- Un nivel o capa define un grado de abstracción de un protocolo, agrupando funciones relacionadas y separando las independientes.
- Una interfaz separa (y une) dos niveles distintos de abstracción.



Las características generales de las capas son las siguientes:

- Cada una de las capas desempeña funciones bien definidas.
- Los servicios proporcionados por cada nivel son utilizados por el nivel superior.
- Existe una comunicación virtual entre dos mismas capas, de manera horizontal.
- Existe una comunicación vertical entre una capa de nivel N y la capa de nivel N + 1.

2.5. VOCABULARIO Y FORMATO

Partiendo de las capas inferiores, en primer lugar aparece la capa de enlace . Ésta agrupa los bits en paquetes discretos denominados tramas *(frames)* que son los que envía por la línea. La utilización de tramas simplifica el proceso de detección y eventual corrección de errores. Para identificar el principio y final de una trama la capa de enlace puede usar varias técnicas:

- En niveles bajos de abstracción, los tres métodos principales de formato son:
    - Orientados a Bit.
    - Orientados a Carácter.
    - Orientados a nº de Bytes *(Byte-count oriented).*
- En niveles altos:
    - Cabeceras y Colas *(Headers and Trailers).*

ORIENTADOS A BIT

Consiste en utilizar una determinada secuencia de bits para indicar el inicio de una trama. Generalmente se usa para este fin la secuencia de bits 01111110, que se conoce como byte indicador *('flag byte').* El receptor está permanentemente analizando la trama que recibe buscando en ella la presencia de un flag byte, y en cuanto lo detecta sabe que ha ocurrido un inicio (o final) de trama. Aunque el flag byte tiene ocho bits el receptor no realiza el análisis byte a byte sino bit a bit, es decir, la secuencia 01111110 puede suceder entre dos bytes y el receptor lo interpretaría como flag byte; esto permite el envío de tramas cuya longitud no sea múltiplo de ocho.

Queda por resolver el problema de que los datos a transmitir contengan en sí la secuencia 01111110; en este caso se usa la técnica conocida como relleno de bits o inserción de bit cero *('bit stuffing').* Consiste en que el emisor, en cuanto detecta que el flujo de bits contiene cinco bits contiguos con valor 1, inserta automáticamente un bit con valor 0. El receptor por su parte realiza la función inversa: analiza el flujo de bits entrante y en cuanto detecta cinco unos contiguos seguidos de un 0 lo suprime en la reconstrucción de la trama recibida. De esta forma la secuencia 01111110 no puede nunca aparecer como parte de los datos transmitidos sino solamente como delimitador de tramas. Si las cosas van mal y el receptor pierde noción de donde se encuentra bastará con que se ponga a la escucha de la secuencia 01111110, que le indicará el principio o final de una trama.
La trama a transmitir incluye casi siempre, además de los datos, alguna información de control de errores, por ejemplo un código CRC. Es importante notar que el relleno de bits (o de bytes) debe aplicarse a la trama inmediatamente antes de transmitirla y después de haber calculado el CRC, ya que de lo contrario la trama no sería interpretada correctamente si el CRC contuviera por azar el carácter o secuencia delimitadora de trama. Esto se ve más fácilmente desde el punto de vista del receptor: Éste debe desempaquetar la trama, detectando donde empieza y acaba y deshaciendo la inserción de ceros.

Ejemplo: Patrón de bits de seis unos seguidos rodeado de ceros.

- Emisor: Después de 5 "1" inserta un "0".
- Receptor: Después de 5 "1", si sigue un cero se elimina, sino es final de trama.

Datos de usuario a transmitir	01110111110101111111110100010
Transmisión	011111100111011111001011111011110100010011111110

Esta técnica de *bit stuffing* se usa en el protocolo HDLC (*High level Data Link Control*) de la capa 2 de la OSI.

De manera simplificada, HDLC es un protocolo de comunicaciones de datos punto a punto entre dos elementos. Proporciona recuperación de errores en caso de perdida de paquetes de datos, fallos de secuencia y otros. Mediante una red de conmutadores de paquetes conectados con líneas punto a punto entre ellos y con los usuarios se constituye la base de las redes de comunicaciones X25.

HDLC es un protocolo de propósito general, que opera a nivel de enlace de datos. Ofrece una comunicación confiable entre el transmisor y el receptor.
Cada dato que se envía, es encapsulado en una trama HDLC, esto añadiéndole un header y un trailer.
-El *header* contiene una dirección HDLC y un campo de control HDLC.
-El *trailer* contiene un campo de CRC *(Ciclic Redundancy Check).*

Cada trama es separada por un delimitador o bandera con valor hexadecimal 7E (0111 1110)

Las grandes ventajas de la familia de protocolos basada en HDLC son:

- Independencia del código utilizado: se trata de enviar conjuntos de bits que en principio pueden configurar información en cualquier código.
- Gran eficiencia en la transmisión: la relación existente entre los bits de información y los bits de control es muy alta.
- Gran fiabilidad en las transmisiones: se dispone de métodos de control para la detección y recuperación de errores con gran eficacia.

ORIENTADOS A CARÁCTER

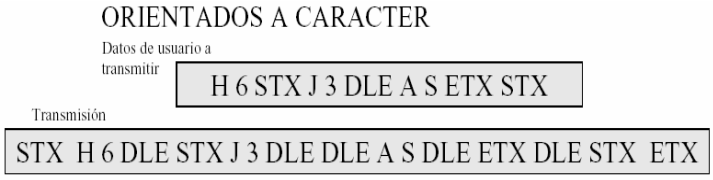
Este método usa caracteres especiales para marcar el inicio y final de cada trama, normalmente los caracteres ASCII STX para el inicio y ETX para el final (STX es *Start of Text* y ETX es *End of Text*). De esta forma si ocurre un error o incidente grave el receptor sólo tiene que esperar al siguiente STX o ETX para saber en que punto se encuentra.

Cuando se usa este sistema para transmitir ficheros binarios es posible que aparezcan en el fichero caracteres STX o ETX, lo cual provocaría una interpretación incorrecta de un principio o final de trama por parte del receptor. Para evitar esto se usa una técnica conocida como relleno de caracteres

('character stuffing'): el emisor cuando ve que ha de transmitir un carácter STX o ETX que proviene de la capa de red intercala en la trama otro carácter DLE; el receptor, cuando recibe DLE, ya sabe que ha de quitar DLE y pasar el siguiente carácter a la capa de red.

El principal problema que tiene el uso de STX y ETX es su dependencia del código de caracteres ASCII. Este método no resulta adecuado para transmitir otros códigos, especialmente cuando la longitud del carácter no es de 8 bits. Tampoco es posible enviar con este sistema tramas cuyo tamaño no sea múltiplo de ocho bits.

Ejemplo:



ORIENTADOS A Nº DE CARACTERES *(BYTE-COUNT)*

En este tipo de protocolos, para indicar donde empiezan y terminan los datos se utiliza un campo, en un lugar conocido (después de STX) de la trama, que indica la longitud en bytes de la misma.
Dejan de ser necesarios tanto el carácter ETX como las técnicas de bit stuffing y character stuffing.

STX	longitud en bytes de la trama	.....

CABECERAS Y COLAS *(HEADERS AND TRAILERS)*

De manera genérica, los métodos de formato de datos de alto nivel se componen de 2 estructuras de datos - cabecera y cola - envolviendo los datos.

Formato = {cabecera, datos, cola}

- cabecera = { tipo, destino, nº secuencia, longitud, ... }
- cola = { checksum, origen, ... }

2.6. REGLAS DE PROCEDIMIENTO

Concurrencia:
• Las reglas son interpretadas por un conjunto de procesos que interaccionan.
• No siempre una interacción es reproducible.
No existe un método que a priori garantice un conjunto de reglas sin ambigüedades.
Existen herramientas que verifican la corrección y consistencia de las reglas.

2.7. DISEÑO ESTRUCTURADO DE PROTOCOLOS

SIMPLICIDAD: Protocolos ligeros *(Light-Weight)*
Un protocolo bien estructurado debería estar formado por un conjunto de piezas conocidas, que hagan una función y la hagan bien.
Se consigue facilidad de implementación y comprensión del funcionamiento.

MODULARIDAD: Jerarquía de funciones
Un protocolo que desempeñe funciones complejas debería construirse de piezas que interactúen de manera simple y bien definida. (Abierto, extendible, modificable).
Cada pieza puede desarrollarse, implementarse, verificarse y mantenerse por separado.
Funciones ortogonales deben separarse en módulos que puedan desconocerse entre sí.

ESPECIFICACIÓN ADECUADA
• Sobre-especificado: Partes de código nunca se ejecutan.
• Sub-especificado (incompleto): situaciones inesperadas.
• Acotado: No desborda el sistema.
• Auto-estabilizado: Si un error modifica el estado del protocolo, éste debe volver a un estado deseable en un número finito de transiciones.
• Auto-adaptativo: Puede modificarse de acuerdo al entorno.

ROBUSTEZ
• No es complicado diseñar protocolos que funcionan en circunstancias normales. Deben funcionar en circunstancias especiales, en cualquier condición y respondiendo a cualquier secuencia de eventos.
• Un diseño robusto se escala sin gran esfuerzo.
• No sobrediseñar: El diseño mínimo es deseable, eliminando lo innecesario.

CONSISTENCIA
Hay circunstancias típicas donde fallan los protocolos:
• Bloqueos *(Deadlocks):* No es posible ejecutar. Todos los procesos esperan unas condiciones que nunca se darán.
• *Livelocks:* Secuencias que se repiten infinitamente sin que el protocolo progrese.

• Terminación incorrecta: Protocolo finaliza sin cumplir las condiciones de terminación adecuadas.

2.8. DIEZ REGLAS DE DISEÑO

Los principios anteriores conducen a estas reglas:
1.- Definir correctamente el problema: Enumerar todos los criterios de diseño, requerimientos y limitaciones *(constraints).*
2.- Definir el servicio a realizar para todo nivel de abstracción antes de escoger estructuras (que, luego cómo).
3.- Diseñar la funcionalidad externa antes. La interna después. Primero considerar la solución como una caja negra que va a interactuar con el entorno. Después decidir cómo organizar internamente la caja negra (normalmente un conjunto de cajas negras que ...).
4.- Mantener el protocolo simple: Los problemas que parecen complejos suelen ser problemas simples juntos.
Identificar problemas simples, separarlos y solucionarlos.
5.- No conectar asuntos independientes.
6.- No introducir restricciones innecesarias. Un buen diseño suele ser extendible, más bien soluciona una clase de problemas que no un caso particular.
7.- Antes de implementar un diseño, verificar el mismo mediante un prototipo de alto nivel comprobando que se satisfacen los criterios de diseño.
8.- Implementar el diseño, medir y, si es necesario, mejorar el comportamiento.
9.- Comprobar que la versión optimizada es equivalente al diseño de alto nivel que fue verificado.
10.- No saltarse las reglas del 1 al 7 (con frecuencia, son las de diseño).

Capítulo 3: CONTROL DE ERRORES
Ingeniería de Protocolos y Servicios

3.1. INTRODUCCIÓN

Los errores de transmisión en su mayoría se deben a la comunicación y no a los errores *hardware* de los ordenadores.
• P(error) circuitos internos < 10^{-15}
• P(error) fibra óptica < 10^{-9}
• P(error) cable coaxial < 10^{-6}
• P(error) línea telefónica < 10^{-4}

Las señales emitidas suelen sufrir dos tipos de deformación; atenuación (reducción de su amplitud); y desfase, siendo esta última la que más afecta a la transmisión.

Otros factores que afectan a la señal son:
- Distorsión Lineal: limitaciones en el ancho de banda del canal.
- Distorsión no Lineal: Eco, diafonía, ruido blanco (por los componentes eléctricos de los transformadores), ruido impulsivo.

El efecto de estos errores se puede solucionar de manera parcial con aislamiento de cables, filtros, canceladores de eco,…pero no se eliminan.
El resto de errores son tarea del protocolo de comunicaciones.

Para expresar los errores se usa:
• Media a largo plazo de la probabilidad de error.
• Porcentaje de tiempo en que la probabilidad de error media no excede un umbral.
• Porcentaje de segundos libres de error.

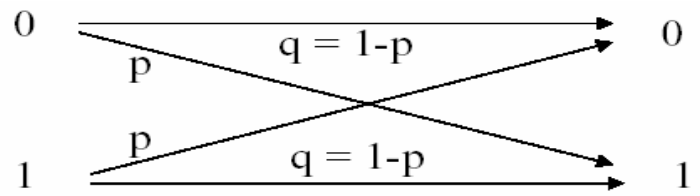
Los dos últimos indican la calidad global de una línea. No existe ningún método que detecte todos los errores posibles: se pretende reducir los errores al nivel deseado.

Un esquema de control de errores debe adaptarse a las características del canal donde va a ser usado:
• Si un canal sólo tiene inserciones, no sirve un protocolo que proteja contra eliminaciones.
• Si un canal produce errores simples, puede ser más adecuado usar un protocolo más simple.
• Si el error del canal es menor que el de la circuitería, el mecanismo de control sólo degrada rendimiento del sistema y disminuye fiabilidad del protocolo.

3.2. MODELOS DE ERROR

Definiciones para el canal:
• Discreto: Reconoce un número finito de valores (0 y 1).
• Sin Memoria: La probabilidad de un error es independiente de los errores anteriores.
• Simétrico: La probabilidad de error es la misma para las distintas señales.

CANAL DISCRETO SIN MEMORIA SIMÉTRICO
No contempla las ráfagas de errores.
Probabilidad de error a largo plazo = p:
• Prob. 1 o más errores en mensaje de n bits = $1-(1-p)^n$



CANAL DISCRETO SIMÉTRICO
Si n bits fueron erróneos es probable que el siguiente sea erróneo (memoria).
Se conoce como Intervalo Libre de Errores (EFI, *Error Free Interval*) a una serie de transmisiones seguidas sin error.
Con la probabilidad de error media a largo plazo = b, en función de la distribución de errores se tiene:

- Lineal:
  $Prob(EFI>=n) = (1-b)^n$, con $n >= 0$

- Poisson:
  $Prob(EFI>=n) = e^{-b(n-1)}$, con $n >= 1$

- Benoit Mandelbrot
  $Prob(EFI>=n) = (n^{(1-a)} -(n-1)^{(1-a)})e^{-b(n-1)}$ con $0 <= a < 1$ y $n >= 1$

3.3. TIPOS DE ERRORES DE TRANSMISIÓN

Existen distintos tipos de errores de transmisión de datos, los más importantes son:
• Inserción
• Eliminación
• Duplicación
• Distorsión
• Reordenación
Se dan por falta de sincronización, mal control de flujo,...

A mayor velocidad de transmisión, un mismo error puede afectar a más bits, por ejemplo:
Un ruido de 1/100 segundos puede afectar:

- Si se transmite a 1 Kbps, a 10 bits.
- Si se transmite a 1 Mbps, a 10.000 bits.

Existen distintos métodos de detección de errores como VRC (V*ertical Redundancy Check*) y LRC *(Longitudinal Redundancy Check*), CRC (*Cyclic Redundancy Check)*, suma de comprobación ... que más tarde estudiaremos.

3.4. REDUNDANCIA

La trama que se transmite de un ordenador a otro está formada por *m* bits de datos y *r* bits redundantes, de comprobación. La trama tiene pues una longitud n=m+r, y forma lo que en teoría de la codificación se denomina una palabra codificada o una *codeword* de n bits.

La eficiencia de un código viene dada por la relación m/n; a la diferencia 1-m/n se le denomina *redundancia*, es decir, la diferencia entre la información máxima que podría proporcionar el alfabeto empleado y la que proporciona realmente.

En esencia cualquier mecanismo de control de errores se basa en la inclusión de un cierto grado de redundancia, lo cual requiere un compromiso entre eficiencia y fiabilidad.

El objetivo esencial de los códigos de detección o corrección de errores consiste en optimizar los algoritmos de cálculo de los bits de control para que sean capaces de detectar el máximo número de errores posible con un número razonable de bits adicionales.

- Los códigos de corrección de errores se denominan corrección de errores hacia delante o FEC (*Forward Error Control),* y corrigen en el receptor.
- Los códigos de detección de errores se denominan corrección de errores hacia atrás o por realimentación (*Feedback error control)* y al detectar un error piden retransmisión.

• Se denomina *error residual* al número de bits erróneos no corregidos en relación al total de bits enviados.
• *Tasa de error residual*: relación entre el número de bits erróneos y los bits totales. Lo mismo que con bits, se puede establecer una tasa para caracteres o bloques.
Si f es la fracción de errores que van a ser detectados, y p la probabilidad de error, la probabilidad de error después de la detección es p(1-f)
• Un esquema de control de errores pretende bajar (imposible eliminar) la probabilidad de error a una cierta cota.

• *Code rate*: d/(d+e)
    d - bits mensaje
    e - bits redundancia
• Hay que elegir el método de control de errores adecuado, en función de p (y los costes de retransmisión).

3.5. TIPOS DE CÓDIGOS

Existen 2 tipos básicos:
• **CÓDIGOS DE BLOQUE:** Las palabras código tienen igual longitud. Cada mensaje genera de manera estática una palabra código. En la palabra a transmitir se pueden distinguir los datos y la redundancia añadida. Los códigos de bloque suelen tener limitada la capacidad de corrección de errores alrededor de 1 o 2 símbolos erróneos por palabra de código. Estos códigos se usan en canales con baja probabilidad de error.

Refinando se tienen:

- Códigos lineales: Un código es lineal sí y solo sí para cualquier par de palabras del mismo código, la combinación lineal de las mismas produce otra palabra código válida.

- Códigos cíclicos: Son una subclase de los códigos de bloque lineales.
Tienen una fuerte estructura algebraica, lo que les proporciona una gran capacidad de detección de errores. Son fáciles de implementar en hardware y usan representaciones polinómicas, por ejemplo, $x^4+x^2+x = 10110$.
Se denominan cíclicos porque para cada palabra válida de código, todos sus desplazamientos cíclicos también lo son.

En resumen, son fáciles de codificar y cumplen las siguientes propiedades:
1º *Linealidad:* la suma módulo-2 de dos palabras del código es otra palabra código.
2º *Cíclicos:* cualquier desplazamiento cíclico de una palabra del código también pertenece al código.

Denotaremos un código cíclico mediante un par (n,k), donde:
n es la longitud de las palabras de código
k es la longitud de una palabra original.

- Códigos sistemáticos: mensaje original + bits de comprobación.
Se añade un grupo de bits obtenidos por alguna operación sobre el conjunto de datos a transmitir. El receptor aplica la operación inversa para saber si hay error en la transmisión:
- Paridad
- Hamming
- CRC

• **CÓDIGOS DE CONVOLUCIÓN**: Se diferencian de los códigos de bloque en su forma estructural y las propiedades para corregir errores. Los códigos de convolución son adecuados para usar sobre canales con mucho ruido, es decir, aquellos en los que existe una alta probabilidad de error.

Los códigos de convolución son códigos lineales, donde la suma de dos palabras de código cualesquiera también es una palabra de código, y al contrario que con los códigos lineales, se prefieren los códigos no sistemáticos.
La longitud de las palabras suele ser constante y no existe distinción entre cuales son los bits de datos y cuales los de redundancia.
El sistema tiene memoria, es decir, la codificación actual de la palabra código depende del mensaje actual y de códigos anteriores.
El codificador cambia su estado con cada mensaje procesado. Utilizan una máquina de estados para generar la secuencia de bits a transmitir.

Un código de convolución queda especificado por tres parámetros (n,k,m):

n es el número de bits de la palabra codificada
k es el número de bits de la palabra de datos
m es la memoria del código o longitud restringida

3.6. BIT DE PARIDAD

Es el ejemplo más sencillo de código de detección de errores. El bit de paridad se elige de forma que mantenga la paridad (par o impar) de la *codeword*. El código formado con un bit de paridad tiene una distancia de 2, ya que cambiando un bit de cualquier codeword el resultado es ilegal, pero cambiando dos vuelve a serlo. Con una distancia 2 es posible detectar errores de 1 bit, pero no es posible detectar errores múltiples, ni corregir errores de ningún tipo. A cambio tiene un overhead mínimo, ya que supone solamente añadir un bit a cada codeword. Por este motivo el bit de paridad se usa en situaciones donde la fiabilidad es muy alta y la codeword muy pequeña, como en algunos sistemas de memoria RAM o de grabación de datos en soporte magnético.
Si q=1-p es la probabilidad de transmitir correctamente un bit:

- prob. de transmitir correctamente un mensaje de n bits es $q(n+1)$
- prob. un solo error en n+1 bits es $(n+1)pq^n$ (binomial)
- prob. error residual: $1- q^{(n+1)} - (n+1)pq^n$
- ej. n=7 y p= 10^{-4} → p.e.r. de $2{,}8 \cdot 10^{-7}$

3.7. CORRECCIÓN DE ERRORES

Los códigos de corrección de errores siempre tienen una eficiencia menor que los de detección para el mismo número de bits, y salvo que el medio de transmisión tenga muchos errores no sale rentable. Por este motivo, los códigos correctores sólo se usan cuando el medio físico no es suficientemente fiable y no es posible emplear códigos detectores, como ocurre en los casos siguientes:

- El canal de comunicación es simplex, es decir, la comunicación sólo es posible en un sentido; en este caso el receptor no dispone de un mecanismo que le permita pedir retransmisión.

- Se realiza una emisión broadcast o multicast; aunque fuera posible pedir retransmisión en este caso sería inaceptable que el emisor tuviera que atender todas las solicitudes que se le planteen.

- El funcionamiento en tiempo real de la aplicación no toleraría el retardo introducido por un mecanismo de reenvío.

La capacidad reparadora de los códigos correctores disminuye cuando aparece una gran cantidad de errores contiguos.

Un esquema de corrección de errores usará un subconjunto reducido de combinaciones de bits para codificar mensajes, escogiéndolas de manera que para convertir un mensaje válido en otro haya que equivocarse en muchos bits. La corrección se realiza escogiendo el código válido más “parecido” al código recibido (menos bits de diferencia).

Ejemplo: Código de repetición. Cada bit se transmite 5 veces.

Mensaje “0” -> Código 00000
Mensaje “1” -> Código 11111

De las 2^5=32 combinaciones de 5 bits se han escogido 2.
Code Rate = d/d+e = 1/1+4 = 0.2

Suponiendo que se quiere transmitir el mensaje 01000001 (A):

00000 11111 00000 00000 00000 00000 00000 11111

Si en recepción se obtiene:

00000 11110 00011 00111 01111 00000 00000 11111

Corrección (corrige 1 o 2 bits): 01011001 (Y) –FEC
Híbrido (corrige 1 bit, detecta 2 o 3 bits): 01xx1001
Detección (detecta 1,2,3 o 4 bits): 0xxxx001.

El *code rate* de un código corrector suele ser más bajo que el de un código detector. Así, se usarán cuando la comunicación receptor-emisor sea complicada, como:
• Retardo de transmisión muy elevado (ej. sonda espacial)
• Ausencia de canal de retorno (ej. emisora *broadcast*)
• Tasa de error elevada (incluso repeticiones con errores)

Ejemplo:
Extensión del bit de paridad para corregir errores

- LRC (*Longitudinal Redundancy Check*): Redundancia horizontal de 1 bit por mensaje
- VRC (V*ertical Redundancy Check*): Redundancia vertical de 8 bits por n mensajes.

Usando código ASCII de 7 bits y n=4

	LRC
H = 72d = 1001000	0
O = 79d = 1001111	1
L = 76d = 1001100	1
A = 65d = 1000001	0
VCR 0001010	0 paridad cruzada

de 40 bits se puede corregir 1.
Code rate = 28/(12+28) = 0.7

Ejemplo (J. H. Van Lint, 1971):
• Tirar una moneda a un ritmo de A veces por segundo: Secuencia aleat.
• Canal broadcast binario simétrico sin memoria de capacidad 2A bits/s y P(error)=$2 \cdot 10^{-2}$
• 1 bit para codificar cara o cruz
Transmisión tal cual: 2% de los resultados son incorrectos

• Solución 1:
Codificar cada resultado con 2 bits. Así los receptores detectarán la mayor parte de los errores
P(error no detectado) = $(2.10^{-2})^2$ = 0,04 %
Problema: No hay retransmisión para corregir los errores. Es necesario un esquema corrector de errores
P(error) = 1 - P(no error en 2 bits) = 3,96 % (peor)

• Solución 2:
Codificar 2 resultados con 4 bits.

Resultado	Código transmisión	Códigos válidos en recepción
cc	0000	0000 1000 0100 0010
cx	1001	1001 0001 1101 1011
xc	0111	0111 1111 0011 0101
xx	1110	1110 0110 1010 1100

Este código resiste errores de 1 bit en las posiciones 1, 2 o 3.
P(error) = 1 - P(no error) = 1 - [$q^4 + 3pq^3$] = 2,12 %

• Transmitiendo a 2A y *code rate* = 0.5 (como antes) se ha bajado de 3,96 % a 2,12 %
• Se puede iterar (mejorar) el proceso con 8 bits para codificar 4 resultados, escogiendo adecuadamente 24 códigos de los 28 posibles.

DISTANCIA DE HAMMING
Es el número de posiciones de bit en que dos palabras difieren. Si dos codewords están separadas por una distancia d serán necesarias d conversiones de un bit para transformar una en la otra.

La distancia de Hamming de un código determina su capacidad de detección y corrección de errores. Para detectar d errores (es decir, d bits erróneos en la misma trama) es preciso que la distancia sea como mínimo de d+1; de esa manera la codeword errónea no coincidirá con ninguna otra codeword válida y el receptor puede detectar la anomalía. Si se quiere un código capaz de corregir d errores es preciso que la distancia de Hamming sea como mínimo 2d+1, ya que entonces la codeword errónea recibida sigue estando más cerca de la codeword original que de cualquier otra.
Así por ejemplo, si la distancia Hamming del código usado en la corrección de errores de un protocolo determinado es de 5, entonces el protocolo podrá corregir hasta 2 errores en una trama y detectar hasta 4.

A esto se le conoce como *Decodificación por Máxima Verosimilitud.*

3.8. CÓDIGOS DE BLOQUES LINEALES

Para transmitir n mensajes es necesario $\log_2 n$ (=m) bits, redondeando m al entero mayor más cercano.
Para proteger los m bits se pueden usar c bits adicionales, resultando en $2^{(m+c)}$ posibles códigos, de los cuales se usarán n.
Se escogen n códigos tal que difieran el máximo número de bits entre ellos.

Se pretende corregir 1 bit: Distancia Hamming mínima de 3 bits
¿Cuantos bits de redundancia c habrá que añadir a m?
• Para cada palabra código de m+c bits se pueden generar justamente m+c códigos de un solo bit erróneo
• Para proteger de un solo error, cada palabra código (de las 2m) necesita por lo menos m+c+1 códigos
Igualando: $(m+c+1)\cdot 2^m = 2^{(m+c)}$
Se obtiene: $m+c+1 = 2^c$

CÓDIGOS DE HAMMING
Es un método general propuesto por R. W Hamming en el que si se añaden junto al mensaje bits detectores y correctores de error y si esos bits se pueden ordenar de modo que diferentes bits de error producen diferentes resultados, entonces los bits erróneos podrían ser identificados.

En un conjunto de siete bits, hay sólo siete posibles errores de bit, por lo que con tres bits de control de error se podría especificar además, qué bit fue el que causó el error.

Hamming estudió los distintos esquemas de codificación existentes, y generalizó sus conclusiones. Para empezar, desarrolló una nomenclatura para describir el sistema, incluyendo el número de los bits de datos y el de los bits detectores y correctores de error en un bloque. Por ejemplo, la paridad incluye un solo bit para cualquier palabra de datos, así que las palabras del Código ASCII que son de siete bits, Hamming las describía como un código (8.7), esto es, un total de 8 bits de los cuales 7 son datos.

Hamming también estudió los problemas que surgían al cambiar dos o más bits a la vez y describió esto como "distancia", ahora llamada distancia de Hamming en su honor, y que fue explicada anteriormente.

Hamming estaba interesado en solucionar simultáneamente dos problemas:
- Aumentar la distancia tanto como sea posible
- Aumentar al máximo los bits de información.

Durante los años 40 desarrolló varios esquemas de codificación que mejoraron notablemente los códigos existentes. La clave de todos sus sistemas era intercalar entre los bits de datos los de paridad.

Hoy en día, el código de Hamming se refiere al (7.4) que Hamming introdujo en 1950. El código de Hamming agrega tres bits adicionales de comprobación por cada cuatro bits de datos del mensaje.
El algoritmo de Hamming (7.4) puede corregir cualquier error de un solo bit, y detecta todos los errores de dos bits.

Para un ambiente en el que el ruido pueda cambiar como máximo 2 bits de 7, el código Hamming (7.4) es generalmente el de pérdida mínima. El medio tendría que ser muy ruidoso para que se perdieran mas de 2 bits de cada 7 (casi el 45% de los bits transmitidos), y habría que considerar seriamente cambiar a un medio de transmisión mas fiable.

Lo explicaremos mediante un ejemplo sencillo:
Consideremos la palabra de datos de 7 bits "0110101".
Para ver cómo se generan y utilizan los códigos Hamming para detectar un error, observe las tablas siguientes.
- **d** para indicar los bits de datos
- **p** para los de paridad.

En primer lugar los bits de datos se insertan en las posiciones apropiadas y los bits de paridad calculados en cada caso usando la paridad par.

	P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7
	1	2	3	4	5	6	7	8	9	10	11
	2^0	2^1	2^0 2^1	2^2	2^2 2^0	2^2 2^1	2^2 2^1 2^0	2^3	2^3 2^0	2^3 2^1	2^3 2^1 2^0
Palabra de datos (sin paridad)			0		1	1	0		1	0	1
P1	1		0		1		0		1		1
P2		0	0			1	0			0	1
P3				0	1	1	0				
P4								0	1	0	1
Palabra de datos (con paridad)	1	0	0	0	1	1	0	0	1	0	1

La nueva palabra de datos (con los bits de paridad) es ahora "10001100101". Consideremos ahora que el bit de la derecha, por error, cambia de 1 a 0. La nueva palabra de datos será ahora "10001100100"; cuando se analice el modo en que se obtienen los bits de paridad en los códigos de Hamming se observarán variaciones en la paridad, lo que significará que hay error.

P1 → D1+D2+D4+D5+D7
P2 → D1+D3+D4+D6+D7
P3 → D2+D3+D4
P4 → D5+D6+D7

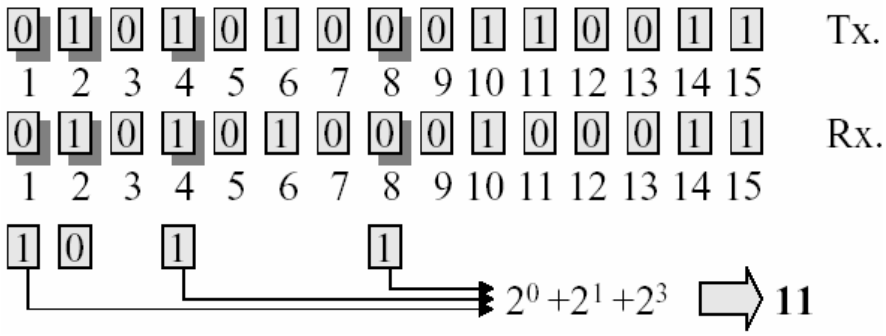
	P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	Prueba de paridad	Bit de paridad
Palabra de datos recibida	1	0	0	0	1	1	0	0	1	0	0	1	
P1	1		0		1		0		1		0	Error	1
P2		0	0			1	0			0	0	Error	1
P3				0	1	1	0					Correcto	0
P4								0	1	0	0	Error	1

El paso final es evaluar los bits de paridad (recuerde que el fallo se encuentra en **D7**). El valor entero que representan los bits de paridad es 11, lo que significa que el bit décimo primero de la palabra de datos (bits de paridad incluidos) es el erróneo y necesita ser cambiado.

	P4	P3	P2	P1	
Binario	1	0	1	1	
Decimal	8		2	1	Suma=11

Cambiando el bit décimo primero 10001100100 se obtiene de nuevo 10001100101. Eliminando los bits de paridad de Hamming se vuelve a obtener la palabra de datos original 0110101.
Observe que en la comprobación de la paridad no se tienen en cuenta los bits de paridad. Si el error se produjera en uno de ellos, en la comprobación sólo se detectaría un error, justo el correspondiente al bit de paridad causante del mismo.
Finalmente, cuando cambien dos bits, en la comprobación de paridad se obtendrá un valor decimal superior a 11, detectándose el error; sin embargo no se podrá saber las posiciones de los dos bits que cambiaron.

Ejemplo: Código de Hamming
La palabra original que quiere ser transmitida es “00100110011” así que 4 bits de redundancia protegerán 11 de información (Code Rate 0.73)



Lo demostraremos paso a paso siguiendo el procedimiento expuesto anteriormente:

	P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	2^0	2^1	$2^0 2^1$	2^2	$2^2 2^0$	$2^2 2^1$	$2^2 2^1 2^0$	2^3	$2^3 2^0$	$2^3 2^1$	$2^3 2^1 2^0$	$2^3 2^2$	$2^3 2^2 2^0$	$2^3 2^2 2^1$	$2^3 2^2 2^1 2^0$
Palabra Tx	0	1	0	1	0	1	0	0	0	1	1	0	0	1	1
Palabra Rx	0	1	0	1	0	1	0	0	0	1	0	0	0	1	1

P1 → D1+D2+D4+D5+D7+D9+D11
P2 → D1+D3+D4+D6+D7+D10+D11
P3 → D2+D3+D4+D8+D9+D10+D11
P4 → D5+D6+D8+D9+D10+D11

	P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	Error
P1	**0**		**0**		**0**				**0**		**0**		**0**		**1**	**1**
P2		**1**	**0**			**1**	**0**			**1**	**0**			**1**	**1**	**1**
P3				**1**	**0**	**1**	**0**					**0**	**0**	**1**	**1**	**0**
P4								**0**	**0**	**1**		**0**	**0**	**1**	**1**	**1**

El paso final es evaluar los bits de paridad (recuerde que el fallo se encuentra en **D7**). El valor entero que representan los bits de paridad es 11, lo que significa que el bit décimo primero de la palabra de datos (bits de paridad incluidos) es el erróneo y necesita ser cambiado.

	P4	P3	P2	P1	
Binario	**1**	**0**	**1**	**1**	
Decimal	**8**		**2**	**1**	**Suma=11**

REPRESENTACIÓN MATRICIAL

Si reordenamos la palabra código del ejemplo anterior escribiendo los datos seguidos de la redundancia:
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 C1 C2 C3 C4

Sabiendo que:
C1→ 1 → 2^0 // BIT DE CONTROL
C2→ 2 → 2^1 // BIT DE CONTROL
D1→ 3 → 2^0+2^1
C3→ 4 → 2^2 // BIT DE CONTROL
D2→ 5 → 2^2+2^0
D3→ 6 → 2^2+2^1
D4→ 7 → $2^2+2^1+2^0$
C4→ 8 → 2^3 // BIT DE CONTROL
D5→ 9 → 2^3+2^0
D6→ 10 → 2^3+2^1
D7→ 11 → $2^3+2^1+2^0$
D8→ 12 → 2^3+2^2
D9→ 13 → $2^3+2^2+2^0$
D10→ 14 → $2^3+2^2+2^1$
D11→ 15 → $2^3+2^2+2^1+2^0$

Obtenemos:
C1 = D1 + D2 + D4 + D5 + D7 + D9 + D11
C2 = D1 + D3 + D4 + D6 + D7 + D10 + D11
C3 = D2 + D3 + D4 + D8 + D9 + D10 + D11
C4 = D5 + D6 + D7 + D8 + D9 + D10 + D11

Reescribiendo:
D1 + D2 + D4 + D5 + D7 + D9 + D11 + C1 = 0
D1 + D3 + D4 + D6 + D7 + D10 + D11 + C2 = 0
D2 + D3 + D4 + D8 + D9 + D10 + D11 + C3 = 0
D5 + D6 + D7 + D8 + D9 + D10 + D11 + C4 = 0

C^t : Matriz traspuesta de la palabra de datos.

$$\left(\begin{array}{cccccccccccc|cccc} 1&1&0&1&1&0&1&0&1&0&1&&1&0&0&0\\ 1&0&1&1&0&1&1&0&0&1&1&&0&1&0&0\\ 0&1&1&1&0&0&0&1&1&1&1&&0&0&1&0\\ 0&0&0&0&1&1&1&1&1&1&1&&0&0&0&1 \end{array}\right) \cdot C^t = \begin{pmatrix} 0\\0\\0\\0 \end{pmatrix} \Rightarrow H \cdot C^t = \mathbf{0}$$

H: Matriz de control de paridad

Los errores de transmisión se pueden formalizar como un vector de error **E** que se añade a la palabra código.
En recepción:

$$H \cdot (Ct + \mathbf{E}) = \mathbf{s}$$

El vector **s** se llama síndrome. Al tratarse de códigos de bloque lineales, si coincidiera con una palabra código, el error **E** no sería detectado (**s** = **0**)

RÁFAGAS DE ERRORES (*Bursts*)
Hasta ahora se ha visto la corrección de un bit erróneo, pero si los errores vienen a ráfagas lo anterior no sirve, y la mayor parte de las veces los errores no se producen de forma aislada.
La extensión de los códigos de Hamming a errores múltiples resulta compleja, pero existen mecanismos correctores tolerantes a ráfagas:

- Códigos de entrelazado -k datos de n bits →matriz k*n
  -se Tx por columnas →corrige ráfagas <k
- -Reed-Solomon -CDs, DVDs, códigos de barras, comunicaciones inalámbricas, TV digital...

Esta forma de corregir ráfagas de errores hace que:

- Aumente el aprovechamiento del canal.
- Disminuya el error residual.

ENTRELAZADO DE CÓDIGOS
Es un método general contra ráfagas de errores, en el que se crea una matriz antes del envío con las palabras código y se cambia el orden de transmisión de los datos.
En recepción se reconstituye dicha matriz, es decir, si los mensajes son de n bits y se esperan ráfagas de errores de longitud k bits, se pueden almacenar k mensajes a transmitir en búferes de tamaño k · n, escribiendo en el búfer por filas, y leyendo

del mismo para transmisión por columnas. De esta forma es posible detectar y corregir errores.
El rendimiento de un código de control viene dado por el número de bits de cada bloque.

Ejemplo: Códigos de 7 bits y ráfagas de 3 bits
Código: A=a1 a2 a3 a4 a5 a6 a7,
B=b1 b2 b3 b4 b5 b6 b7,
C=c1 c2 c3 c4 c5 c6 c7

Transmite: a1 b1 c1 a2 b2 c2 a3 b3 c3 a4 b4 c4 a5 b5 c5 a6 b6 c6 a7 b7 c7
Recibe: a1 b1 c1 a2 b2 c2 a3 xx xx xx b4 c4 a5 b5 c5 a6 b6 c6 a7 b7 c7

Código Rx: A=a1a2 a3 xx a5 a6 a7,
B=b1 b2 xx b4 b5 b6 b7,
C=c1 c2 xx c4 c5 c6 c7

Corregible si, por ejemplo, se trataba de un código de Hamming 3,4

CÓDIGOS REED-SALOMON
El codificador Reed-Salomon toma un bloque de información digital y añade bits redundantes. Los errores ocurren durante la transmisión o almacenamiento de información por varios motivos (ruido o interferencias, ralladuras en los discos compactos ...)

El decodificador Reed-Salomon procesa cada bloque e intenta corregir los errores y recuperar la información original. El número y tipo de errores que pueden ser corregidos depende de las características del código Reed-Salomon.

3.9. CONTROL DE REDUNDANCIA CÍCLICO (CRC)

El algoritmo de detección de errores más usado en la práctica se basa en lo que se conoce como *códigos polinómicos* (también llamados *códigos de redundancia cíclica* o CRC, *Cyclic Redundancy Check*). La idea básica es añadir a los datos a transmitir unos bits adicionales cuyo valor se calcula a partir de los datos; la trama así construida se envía, y el receptor separa los datos de la parte CRC; a partir de los datos recalcula el CRC y compara con el valor recibido; si ambos no coinciden se supone que ha habido un error y se pide retransmisión.

La aritmética polinómica tiene unas propiedades singulares que la hacen especialmente fácil de programar en sistemas digitales, por lo que es posible implementarla directamente en hardware con lo que se consigue una eficiencia elevada, cosa importante para evitar que la comunicación se ralentice por el calculo del CRC. Veamos algunas de estas propiedades:

Supongamos la siguiente suma de polinomios:

$x^7+x^4+x^3+x+1$	que equivale a:	10011011
$x^7+x^6+x^3+x$	que equivale a:	11001010
---		---
x^6+x^4+1		01010001

Obsérvese como no se arrastra valor a la unidad superior. En la práctica el resultado de la suma es equivalente a haber efectuado un OR EXCLUSIVO bit a bit entre dos cadenas.

En el caso de la resta la situación es idéntica:

$x^6+x^4+x^2+1$	que equivale a:	01010101
$x^7+x^5+x^3+x^2+x+1$	que equivale a:	10101111
---		---
$x^7+x^6+x^5+x^4+x^3+x$		11111010

Ya que al usar módulo 2 la operación de dos valores iguales siempre da 0 y dos diferentes da 1.

En el caso de la división la operación se hace como en binario con la única peculiaridad de que la resta se hace módulo 2, como acabamos de ver.

Veamos paso a paso como se usa todo en una transmisión de datos con un ejemplo concreto:

1. En primer lugar el emisor y el receptor acuerdan un generador polinómico común G(x), por ejemplo x^4+x+1 (que representaremos como 10011 o *g*); el primer y último bits de un generador polinómico siempre deben ser 1. El CRC siempre tiene una longitud de un bit menos que el generador polinómico usado, por lo que en nuestro caso será de 4 bits.

2. Supongamos ahora que el emisor desea transmitir la cadena *c1*, formada por los bits 1101011011, que podemos ver como un polinomio de grado 9 (los datos a transmitir siempre deben tener más bits que el generador polinómico usado). El emisor añade cuatro bits (en principio a 0) al final de los datos a transmitir, formando la cadena *c2* 110010110110000; esto equivale a multiplicar la cadena por 2^4

3. El emisor divide la cadena *c2* por el generador polinómico acordado (10011) usando las reglas de división binaria módulo 2 que antes hemos mencionado, y calcula el resto *r*, que es en este caso 1110.

4. El emisor resta el resto *r* de la cadena *c2*, formando así la cadena *c3* 11010110111110.

Obsérvese que, como la resta es una operación XOR sobre los cuatro últimos bits, en la práctica la resta se hace sencillamente sustituyendo los cuatro últimos bits de *c2* por *r*. Al restar al dividendo el resto, el valor obtenido es divisible por *g*.

5. La cadena *c3* es transmitida al receptor.

6. El receptor recibe la cadena *c3* y la divide por *g.* Si el resultado no es cero la transmisión se considera errónea y se solicita retransmisión.

Este algoritmo, que en principio puede parecer extraño y arbitrario, tiene tres características interesantes:

- Dará un resultado predecible y reproducible, por lo que aplicado sobre unos mismos datos siempre dará el mismo resultado.

- Las operaciones usadas lo hacen muy fácil de implementar en hardware.

- Suministra un mecanismo extremadamente flexible y robusto para la detección de errores, a través de la elección del generador polinómico adecuado.

Los generadores polinómicos más usados forman parte de estándares internacionales, y son los siguientes:

CRC-12: $x^{12} + x^{11} + x^3 + x^2 + 1$

CRC-16: $x^{16}+x^{15}+x^2+1$

CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$

CRC-32 (IEEE-802): $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

CRC-12 se usa en los códigos con longitud de carácter de 6 bits; CRC-16 y CRC-CCITT se usan en conexiones WAN, mientras que CRC-32 se usa en conexiones LAN.

En todos los generadores usados aparece x+1 como factor, ya que esto asegura la detección de todos los errores con un número impar de bits. Un código polinómico de r bits detectará todos los errores a ráfagas de longitud =<r. Un generador como CRC-16 o CRC-CCITT detectará todos los errores simples y dobles, todos los errores con un número impar de bits, todos los errores a ráfagas de longitud 16 o menor 99,997% de los errores a ráfagas de 17 bits y 99,998% de los errores a ráfagas de 18 o más bits.

Cabría pensar en la posibilidad de que un error alterara la trama de tal forma que el CRC de la trama errónea coincidiera con el de la trama correcta; los algoritmos de cálculo de CRCs intentan conseguir que las otras posibles tramas con igual CRC se encuentren muy alejadas (en términos de distancia de Hamming) por lo que tendría que producirse una gran cantidad de errores en la misma trama para que la posibilidad pudiera darse. En todo caso, cualquier protocolo de nivel de enlace fallará si se produce un error que no pueda ser detectado por el CRC.

3.10. CHECKSUM ARITMÉTICO

- Sólo sumas y módulos, es simple, pero con buena detección de errores, menor carga de procesamiento y menor latencia.
- Inferior al CRC.

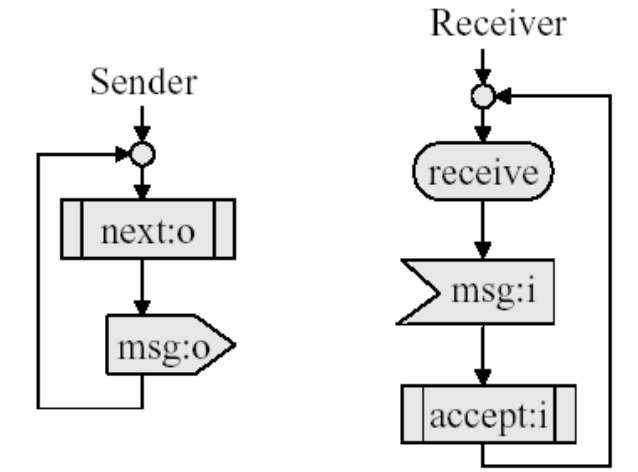
Capítulo 4: Control de Flujo
Ingeniería de Protocolos y Servicios

4.1. INTRODUCIÓN

¿Para qué sirve el control de flujo?
• Asegurar que los datos no se transmitan más rápido de lo que se puedan procesar.
• Optimizar el uso del canal.
• Evitar la saturación de ciertos enlaces del canal.
• Proteger la transmisión contra borrado, inserción, duplicación y reordenamiento de mensajes.

A continuación se verán algunos modelos de control de flujo en orden creciente de dificultad.

MODELO BÁSICO: Protocolo sin control de flujo.
-simplex-



• Funciona si el receptor es más rápido que el receptor.
• Regla de diseño de sistemas concurrentes: No hacer suposiciones de las velocidades relativas de procesos concurrentes.

PROTOCOLO X-ON X-OFF
• El funcionamiento correcto del protocolo, depende de las propiedades de transmisión del canal.
- Suspend: cuando el emisor lo recibe para las transmisiones.
- Resume: cuando el emisor lo recibe reanuda las transmisiones.

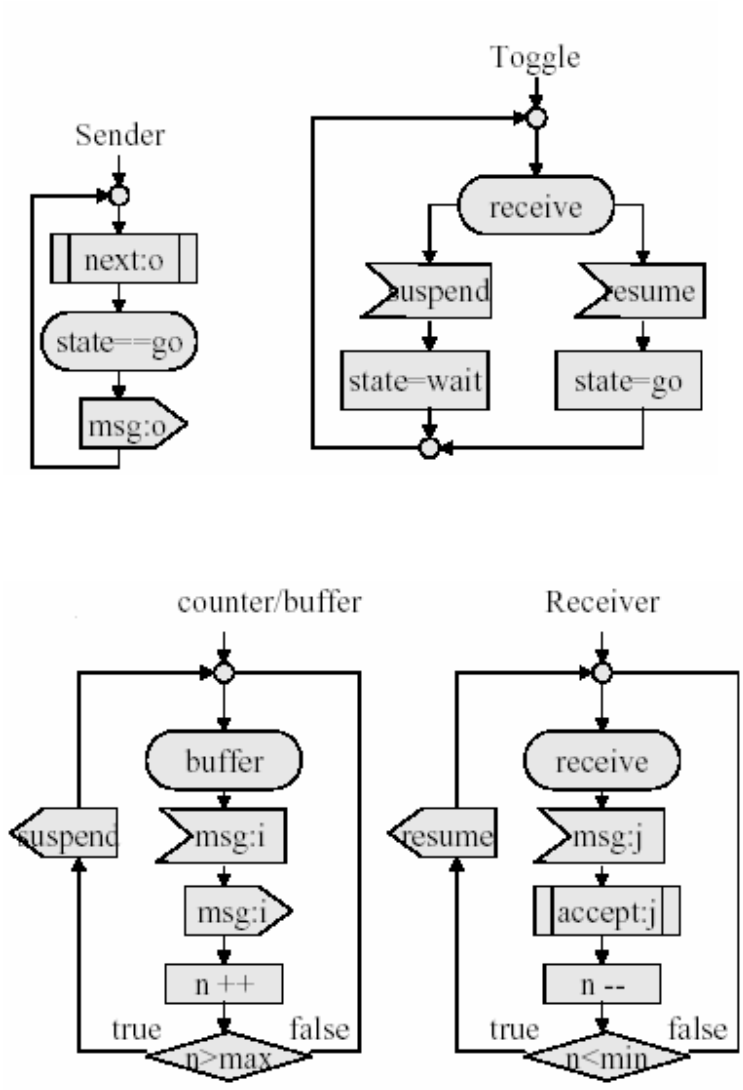
• Utiliza 2 mensajes de control:
Pérdidas: - Mensaje *resume* → procesos bloqueados.
- Mensaje *suspend* → overflow.

Se requiere solucionar los siguientes dos problemas:
- Proteger contra la pérdida de mensajes.
- Proteger contra problemas de overflow.

• No requiere negociación previa.

Así, suponiendo un canal sin errores y el vocabulario:
V = { *msg, suspend, resume* }

Las reglas de procedimiento para el emisor y para el receptor serán:



Los mensajes de datos pasan de *counter/buffer* a *receiver* a través de una cola interna.
Pegas: Son las generadas por la dependencia del protocolo a las propiedades del canal:

- Si un mensaje *suspend* se retrasa o pierde hay *overflow*.
- Si un mensaje *resume* se pierde los 4 procesos colapsan.

PING - PONG

Hay que solucionar 2 problemas:
• Problemas de *overflow* del mecanismo X-ON/X-OFF
• Protección contra pérdidas de mensajes.
Propuesta: Protocolo ping-pong (*Stop and wait*)

PROTOCOLO DE PARADA Y ESPERA *(Stop and Wait)*

Consiste en que el emisor espera confirmación o acuse de recibo después de cada envío o antes de efectuar el siguiente. El acuse de recibo, también llamado ACK (del inglés acknowledgement) sirve tanto para indicar que la trama ha llegado correctamente como para indicar que se está en condiciones de recibir la siguiente, es decir, el protocolo incorpora también la función de control de flujo. Este tipo de protocolos donde el emisor espera una confirmación o acuse de recibo para cada dato enviado se denominan protocolos ARQ (Automatic Repeat reQuest).

Cuando la trama recibida es errónea no se produce ACK. Lo mismo sucede cuando la trama enviada se pierde por completo. En este caso el emisor, pasado un tiempo máximo de espera, reenvía la trama. Una optimización que se puede incorporar en el protocolo es el uso de acuse de recibo negativo o NAK(Negative Acknowledgement) cuando se recibe una trama errónea; de esta forma el emisor puede reenviar la trama sin esperar a agotar el tiempo de espera, con lo que se consigue una mayor utilización de la línea.

Supongamos que una de las veces lo que se pierde no es la trama enviada sino el mensaje de ACK; pasado el tiempo de espera el emisor concluirá erróneamente que la trama se ha perdido y la reenviará, llegando ésta duplicada al receptor; el receptor no tiene ningún mecanismo para detectar que la trama es un duplicado, por lo que pasará el duplicado al nivel de red, lo cual no sería deseable en un protocolo de enlace. Una forma de que el receptor distinga los duplicados es numerar las tramas, por ejemplo, con un campo de un bit podemos numerar las tramas en base2 (0, 1, 0, 1, ...) que es suficiente para detectar los duplicados.

Aunque la transmisión de datos ocurra únicamente en un sentido, este protocolo requiere un canal dúplex para funcionar; como la comunicación no ocurre simultáneamente un canal semi-dúplex sería suficiente.

Los protocolos de parada y espera son sencillos de implementar, pero son poco eficientes. Veamos un ejemplo.

Supongamos que usamos una línea de 64Kb/s para enviar tramas de 640 bits de un ordenador A a otro B que se encuentra a una distancia de 2.000 Km. A tarda 10ms en emitir cada trama (640/64000) o, dicho de otro modo, transmite 64 bits cada milisegundo. Por otro lado los bits tardan 10ms en llegar de A a B (la velocidad de las ondas electromagnéticas en materiales es aproximadamente 200.000Km/s); justo cuando llega a B el primer bit de la primera trama A termina de emitirla (en ese momento la trama está 'en el cable'); diez milisegundos más tarde B recibe la trama en su totalidad, verifica el CRC y devuelve el ACK; Suponiendo que el tiempo que tarda B en verificar el CRC y generar el ACK es despreciable la secuencia de acontecimientos es la siguiente:

Instante(ms)	Suceso en 'A'	Suceso en 'B'
0ms	Emite primer bit de trama 1	Espera
10ms	Emite último bit de trama 1;espera	Recibe primer bit de trama 1
20ms	Espera	Recibe último bit de trama 1; envía ACK
30ms	Recibe ACK; emite primer bit de trama 2	Espera

A partir de aquí el ciclo se repite. De cada 30ms se están transmitiendo 10ms y esperando 20ms, es decir, se está usando la línea con una eficiencia de 10/30=0,33=33%.

La eficiencia obtenida depende de tres parámetros: la velocidad de la línea (v), el tamaño de la trama (s) y el tiempo de ida y vuelta, también llamado 'Round Trip Time'; en el caso normal de que el tiempo de ida y el de vuelta son iguales definimos como el tiempo de ida T, por lo que el tiempo de ida y vuelta es de 2T.

Podemos derivar una expresión que nos permita calcular la eficiencia a partir de estos valores:

$$\text{Eficiencia} = \frac{s/v}{(s/v) + 2T}$$

Que aplicada al ejemplo anterior resulta:

Eficiencia = (640/64000) / (640/64000 + 2*0,01)=0,01 / (0,01 + 0,02) = 0,01 / 0,03

Si en vez de una línea de 64Kb/s hubiera sido una de 2.048Mb/s la eficiencia habría sido del 1,5%. Es evidente que los protocolos de parada y espera tienen una baja eficiencia en algunos casos. El caso extremo de ineficiencia se da cuando se usan enlaces vía satélite, en los que el valor de 2T puede llegar a ser

de medio segundo. Para aprovechar mejor los enlaces con valores elevados del tiempo de ida y vuelta hacen falta protocolos que permitan crear un 'pipeline', o dicho de otro modo tener varias tramas 'en ruta' por el canal de transmisión.

Al tener varias tramas simultáneamente pendientes de confirmación necesitamos un mecanismo que nos permita referirnos a cada una de ellas de manera no ambigua, ya que al recibir los ACK debemos saber a que trama se refieren. Para ello usamos un número de secuencia; sin embargo como el número de secuencia va a aparecer en todas las tramas y los mensajes ACK nos interesa que sea lo menor posible; por ejemplo con un contador de 3 bits podemos numerar las tramas módulo 8 (0, 1, 2, ...,7) con lo que es posible enviar hasta siete tramas (0....6) antes de recibir el primer ACK; a partir de ese punto podemos enviar una trama por cada ACK recibido. Esto es lo que se denomina protocolo de *ventana deslizante*.

Podemos imaginar el funcionamiento del protocolo de ventana deslizante antes descrito como un círculo dividido en ocho sectores de 45º cada uno, numerados del 0 al 7; sobre el círculo hay una ventana giratoria que permite ver los sectores correspondientes a las tramas pendientes de confirmación; la ventana puede abrirse como máximo 315º, es decir, permite ver hasta siete sectores correspondientes a las tramas enviadas pendientes de confirmación. Cuando se recibe un ACK se envía otra trama y la ventana gira un sector.

El protocolo de parada y espera se puede considerar como un protocolo de ventana deslizante en el que se usa un bit para el número de secuencia; en este caso el círculo estaría formado por dos sectores de 180º cada uno, y la ventana tendría una apertura de 180º.

Suponiendo un retardo nulo en el envío de los bits y en el proceso de las tramas en los respectivos sistemas, así como una longitud nula de las tramas ACK, el tamaño de ventana mínimo necesario W para poder llenar el canal de comunicación puede calcularse con la fórmula:

$$W = 2T * v / s+1$$

Debiendo redondearse el valor al entero siguiente por encima. En la fórmula anterior 2T es el tiempo (en segundos) que tarda una trama en hacer el viaje de ida y de vuelta *(Round Trip Time),* v es la velocidad del canal de transmisión y *s* el tamaño de la trama a transmitir. Por ejemplo para el caso del ejemplo anterior donde 2T=0,02 v=64000 y t=640 obtenemos W=3, por tanto la ventana mínima es 3; con una línea E1(v=2048000) W=65.

La suposición de que los tiempos de proceso y la longitud de las tramas ACK son despreciables no es correcta, por lo que en la práctica se consigue una mejora en el rendimiento incluso para valores de W bastante superiores al cálculo en la fórmula anterior.

Cuando se usa un protocolo de ventana deslizante con ventana mayor que uno, el emisor no actúa de forma sincronizada con el receptor; cuando el receptor detecta una trama defectuosa puede haber varias posteriores ya en camino, que llegarán irremediablemente a él, aún cuando reporte el problema inmediatamente. Existen dos posibles estrategias en este caso:

- El receptor ignora las tramas recibidas a partir de la errónea (inclusive) y solicita al emisor retransmisión de todas las tramas a partir de la errónea. Esta técnica se denomina *retroceso n.*
- El receptor descarta la trama errónea y pide retransmisión de ésta, pero acepta las tramas posteriores que hayan llegado correctamente. Esto se conoce como *repetición selectiva.*

Siguiendo con la representación en círculo de los números de secuencia podemos imaginar el comportamiento de retroceso n como una ventana de tamaño uno en el lado del receptor, es decir, el receptor solo aceptará recibir las tramas en estricta secuencia e irá desplazando su ventana una posición cada vez. En el caso de repetición selectiva el receptor tiene la ventana abierta el mismo número de sectores que el emisor.

PING-PONG

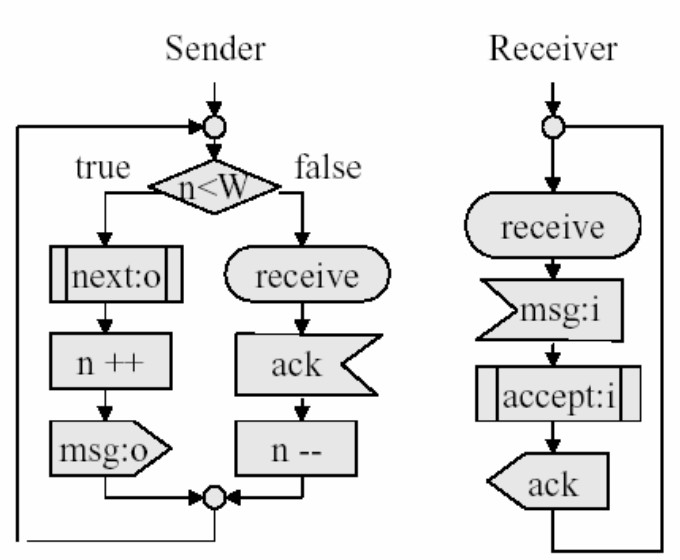
2t+a+p
t: tiempo propagación
p: tiempo producir
a: tiempo aceptar msg

4.2. PROTOCOLOS DE VENTANA

El mensaje de control *ack* del protocolo *Stop and Wait* puede verse como un mecanismo de crédito.
Así, al iniciar la comunicación, el receptor puede indicar el buffer disponible dando cierto crédito al emisor.
Suponiendo un canal sin pérdidas, el emisor:

- Por cada mensaje transmitido decrementa 1 crédito
- Por cada ack recibido incrementa 1 crédito

Así, suponiendo que el emisor dispone de un número inicial de créditos W:



Examinando el diagrama de flujo se puede observar la cantidad W-n que indica el número de créditos sin usar.
A nivel teórico:

a(t) - el número de créditos recibidos por el emisor (siempre creciente).
m(t) - el número de mensajes enviados (siempre creciente).
n(t) - el valor de n en el instante t

El número máximo de mensajes que el emisor tiene pendientes de recibir *ack* es:

W-n(t)+m(t)-a(t)

Que debe ser <=W
El valor de W se llama *"tamaño de la ventana".*

PÉRDIDAS DE MENSAJES: 2 problemas

Si se pierden suficientes *ack* el sistema se bloquea: Es necesario considerar el tiempo transcurrido desde la transmisión de un mensaje hasta la recepción del *ack*.
TIMEOUTS: Si se supera una cierta cota el tiempo de ida y vuelta, se retransmite.

El segundo problema que surge es hacer corresponder cada *ack* con su mensaje: aparición de los números de secuencia.

4.3. NUMEROS DE SECUENCIA Y RECONOCIMIENTO POSITIVO

Necesarios para resolver los siguientes problemas:

- Mensajes duplicados
- Mensajes fuera de secuencia

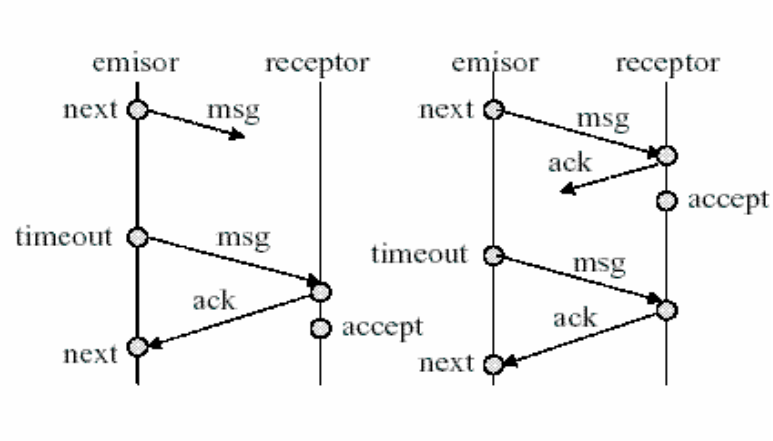
Para solucionarlo se utilizan los números de secuencia y el reconocimiento positivo. A cada mensaje enviado se le asignan números consecutivos para evitar repeticiones. A su vez, cada reconocimiento podría hacerse corresponder a un mensaje.

Ejemplo: Protocolo de bit alternado
Si al protocolo de ping-pong se le añade la posibilidad de pérdidas -uso de timeouts- y el uso de números de secuencia (1 bit) se convierte en el protocolo de bit alternado con timeouts.

Se usan 2 tipos de mensaje:

- msg { msg, dato, nº secuencia }
- ack { ack, nº secuencia }

Ahora si hay pérdidas de mensajes la recuperación es posible.



RECICLADO DE NÚMEROS DE SECUENCIA

Los números de secuencia tienen un rango limitado (número finito de bits en una cabecera) → Reciclaje.

¿Cuál será el número mínimo de números de secuencia?
¿Qué relación hay entre el tamaño de ventana W y los M números de secuencia disponibles?

Intuitivamente M tiene que ser suficientemente mayor que W para aparear cada mensaje con su *ack*.

RELACIÓN TAMAÑO VENTANA - Nº SECUENCIA

M: rango de números de secuencia.
W: tamaño de ventana.
H: último mensaje reconocido por el receptor.

Desde el punto de vista del receptor, se tiene que:

- Mensaje más viejo que puede ser retransmitido:

  (H-W+1)%M

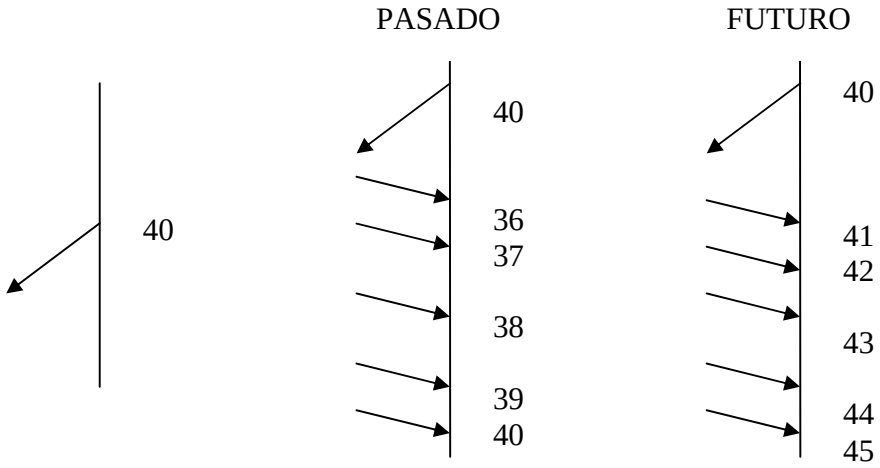
- Mensaje más nuevo (futuro) que puede ser transmitido:

  (H+W)%M

Estos mensajes deben ser distinguibles: M > 2W - 1

Ejemplo:

W = 5



H - W + 1 = 45 - 36 + 1 = 10
45 y 36 deben ser distinguibles (10 mensajes distinguibles)

4.4. RECONOCIMIENTO NEGATIVO

El concepto de reconocimiento negativo sirve para que una vez detectada una trama con errores, solicitar la retransmisión de la misma.

Este método es útil si la probabilidad de error es alta, ya que cuando el emisor recibe un *nack* ya no debe esperar el *timeout* para saber que su mensaje llegó mal.

Las 3 variantes principales de protocolos que usan reconocimientos son:

- *Stop and Wait* (Parada y espera)
- *Selective repeat* (Repetición Selectiva)
- *Go-back-N continuous* (Retroceso N)

PROTOCOLO DE RETROCESO N

En retroceso n el receptor procesa las tramas en estricta secuencia, por lo que sólo necesita reservar espacio en buffers para una trama. En cambio en repetición selectiva el receptor ha de disponer de espacio en el buffer para almacenar todas las tramas de la ventana, ya que en caso de pedir retransmisión tendrá que intercalar en su sitio la trama retransmitida antes de pasar las siguientes a la capa de red (la capa de red debe recibir los paquetes estrictamente en orden).

En cualquiera de los dos casos el emisor deberá almacenar en su buffer todas las tramas que se encuentren dentro de la ventana, ya que en cualquier momento el receptor puede solicitar la retransmisión de alguna de ellas.

Con un número de secuencia de n bits se puede tener como máximo una ventana de 2^{n-1} tramas, no de $2^{n.}$
Por ejemplo, con un número de secuencia de tres bits el emisor puede enviar como máximo cuatro tramas sin esperar contestación.

PROTOCOLO CON REPETICIÓN SELECTIVA

La repetición selectiva aprovecha las tramas correctas que llegan después de la errónea, y pide al emisor que retransmita sólo esta trama. Como los paquetes se han de transferir en orden a la capa de red cuando falla una trama el receptor ha de conservar en buffers todos los paquetes posteriores hasta conseguir correctamente la que falta; en la práctica esto requiere tener un buffer lo suficientemente grande para almacenar un número de tramas igual al tamaño de la ventana, ya que se podría perder la primera trama de la ventana y recibirse correctamente el resto, en cuyo caso habría de conservarlas hasta recibir correctamente la primera.

La posibilidad de una recepción no secuencial de tramas plantea algunos problemas nuevos. Por ejemplo, supongamos que con un número de secuencia de tres bits el emisor envía las tramas 0 a 6, las cuales son recibidas correctamente.
Entonces el receptor realiza las siguientes acciones:

1. Las transmite a la capa de red
2. Libera los buffers correspondientes
3. Avanza la ventana para poder recibir siete tramas más, cuyos números de secuencia podrán ser 7,0,1,2,3,4,5
4. Envía un ACK para las tramas 0 a 6 recibidas

Imaginemos ahora que al ACK no llega no al emisor. Éste supondrá que ninguna de las tramas ha llegado, por lo que las reenviará todas de nuevo (tramas 0 a 6). De estas las tramas 0 a 5 se encuentran dentro de la ventana del receptor y son por tanto aceptadas; la trama 6 está fuera de rango y es ignorada. En procesamiento secuencial el receptor no aceptaría estas tramas si no recibiera antes la trama 7 pendiente, pero con retransmisión selectiva las tramas fuera de orden se aceptan y se pide retransmisión de la trama 7; una vez recibida ésta se pasaría a la capa de red seguida de las tramas 0 a 5 antes recibidas, que serían

duplicados de las anteriores. Los duplicados no detectados serían pasados al nivel de red, con lo que el protocolo es erróneo.

La solución a este conflicto está en evitar que un mismo número de secuencia pueda aparecer en dos ventanas consecutivas. Por ejemplo con un número de secuencia de 4 bits (0-15) y tamaño de ventana 8 la ventana del receptor sería inicialmente 0-7, después 8-15, 0-7 y así sucesivamente. Al no coincidir ningún número de secuencia entre ventanas contiguas se puede efectuar el proceso no secuencial de tramas sin que ocurra el conflicto anterior. El valor máximo de la ventana para un protocolo de repetición selectiva en el caso general es (MAX_SEQ+1)/2.

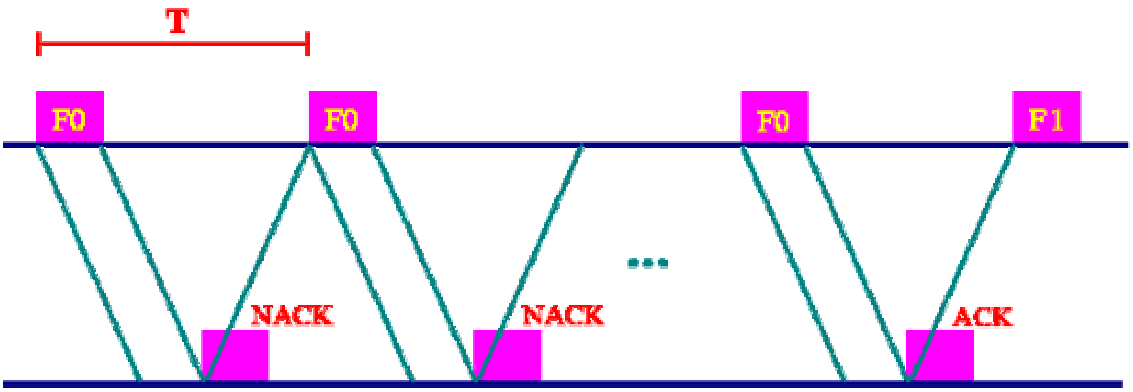
Como es lógico la técnica de repetición selectiva da lugar a protocolos más complejos que la de retroceso n, y requiere mayor espacio de buffers en el receptor. Sin embargo, cuando las líneas de transmisión tienen una tasa de errores elevada la repetición selectiva da un mayor rendimiento, ya que permite aprovechar todas las tramas correctamente transmitidas. La decisión de cual usar se debería tomar valorando en cada caso la importancia de estos factores: complejidad, espacio en buffers, tasa de errores y eficiencia.

Breve explicación del cálculo de la eficiencia:

Parada y espera:
Si llamamos T al tiempo que tarda una trama en llegar a su destino más el que tarda su ACK correspondiente en llegar a la fuente, y tenemos en cuenta que al ser una técnica ARQ un error supone la retransmisión de la trama las veces que sea necesario (digamos Nr), el tiempo total que una trama tarda en ser transmitida correctamente es:

$$T_{TOTAL} = Nt * T$$

Con **Nt = Nr +1** como el número de transmisiones realizadas, y donde hemos supuesto que tanto el tiempo de procesamiento como el de ACK son despreciables.



Como se observa en la figura: T = Ttx + 2 T prop, quedando: T_{TOTAL} = Nt [Ttx + 2 T prop]. El número de transmisiones (Nt) es función de la probabilidad de error del canal, cuanto mayor sea, mayor también el número de retransmisiones necesarias:

Si hay que retransmitir muchas veces, por ejemplo n=16, se tira el enlace.
1-Peb es la probabilidad de que no halla error en la trama (lo que antes hemos llamado Pneb).

Nt	Prob
1	1-Peb
2	Peb (1-Peb)
3	Peb^2 (1-Peb)
⋮	⋮
n	Peb^{n-1} (1-Peb)

Nt =1*(1-Peb)+2* Peb(1-Peb)+3*Peb^2(1-Peb)+...+n*Peb^{n-1}(1-Peb)= 1**/1-Peb**

$$U = \frac{Ttx}{T_{TOTAL}} = \frac{Ttx}{\frac{1}{1-Peb}\ (Ttx + 2Tprop)}$$

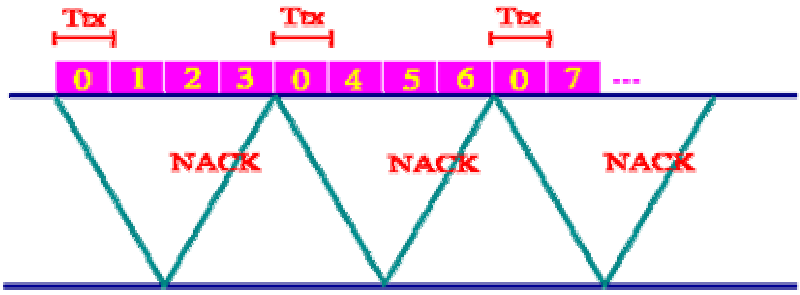
$$U = \frac{1-Peb}{1+2a}$$

Simplificando, la fórmula de la eficiencia cuando la transmisión no está libre de errores es la misma que para un canal ideal, pero dividida por el número total de transmisiones realizadas.

Rechazo Selectivo con Envío Continuo:
Siguiendo el mismo razonamiento conque finalizaba el apartado anterior, para hallar la eficiencia del ARQ con rechazo selectivo basta con dividir la fórmula de la eficiencia entre Nt = 1/1- Peb

$$U = \begin{cases} 1-Peb & N>1+2a \\ \dfrac{N(1-Peb)}{1+2a} & N<1+2a \end{cases}$$

Para el caso de N>1+2a la solución es fácilmente demostrable observando el diagrama de tiempos



En este caso el tiempo total T_{TOTAL} empleado en transmitir la trama (la número 0 en el ejemplo) es el tiempo de transmisión tantas veces como haya sido necesario retransmitir.

$$U = \frac{Ttx}{T_{TOTAL}} = \frac{Ttx}{Nt\ Ttx}$$

$$\boxed{U = 1-Peb}$$

Rechazo Simple con Envío Continuo:

Los mismos razonamientos se pueden aplicar a este caso, sin embargo hay que poner especial atención en la aproximación de Nt.

Ahora cada error hace que sea necesaria la retransmisión de K tramas en lugar de sólo una:

$$Nt = S_{i=1}\ f(i)\ Peb^{i-1}\ (1-Peb)$$

Donde f(i) es el número total de tramas a transmitir si la trama original tiene que ser transmitida i veces.
f(i) se puede expresar como:

$$f(i)= 1+(i-1)K = (1-K) + Ki$$

Sustituyendo una ecuación en la otra queda que:

$$Nt = (1-Peb+KPeb)/1-Peb$$

por lo que sabemos ya de la técnica de rechazo simple **K** se puede aproximar a **(1+2a)** si N> 1+2a, o a **N** cuando N<1+2a:

$$U= \begin{cases} \dfrac{(1-Peb)}{(1+2aPeb)} & N>1+2a \\ \dfrac{N(1-Peb)}{(1+2a)(1-Peb+NPeb)} & N<1+2a \end{cases}$$

Como hicimos antes, para el caso de N>1+2a con Nr = Nt-1 = Peb/1-Peb, se demuestra que la eficiencia es:

$$U = \frac{Ttx}{T_{TOTAL}} = \frac{Ttx}{Nr^{*}(Ttx+2Tprop)+Ttx} = \frac{1}{Nt(1+2a)+1}$$

$$\boxed{U= \frac{1-Peb}{1+2aPeb}}$$

4.5. PREVENCIÓN DE LA CONGESTIÓN

La congestión se define como una excesiva cantidad de paquetes almacenados en los buffers de varios nodos en espera de ser transmitidos. La congestión es indeseable porque aumenta los tiempos de viaje de los paquetes y retrasa la comunicación entre los usuarios.

Para entender el fenómeno de la congestión es necesario analizar el comportamiento de la subred de conmutación de paquete como una subred de colas. En cada nodo, asociado a cada canal habrá una cola de entrada o salida respectivamente. Si la velocidad de llegada de los paquetes al nodo excede la velocidad con la que pueden ser transmitidos, la cola asociada al canal de salida empieza a crecer y los paquetes irán experimentando un retardo creciente, que podría llegar a tender a infinito si la longitud de las colas lo permitiera.

Cuando se alcanza el punto de saturación y el nodo no puede absorber más paquetes, tiene dos posibilidades:

- Rechazar los paquetes que van llegando.

- Ejercer un control de flujo sobre sus vecinos, impidiendo el envío de nuevos paquetes.

Ambas estrategias conducirán a la saturación de los nodos vecinos, debido a que no podrán deshacerse de los paquetes que tenían para enviar. Así, la congestión en un punto de la subred se propaga rápidamente hacia las zonas vecinas. Por ello, será deseable establecer algún tipo de control para evitar estas situaciones. Ello disminuirá las prestaciones de la subred respecto al caso ideal en una tasa aproximadamente igual a la sobrecarga de control generada; pero evitará que se produzcan situaciones catastróficas que podrían conducir al bloqueo total de la subred.

Para un cierto enlace, ¿cómo se escoge W y M?
Es fácil determinar el límite superior del tamaño de la ventana: después de un punto, incrementar este tamaño no aumenta el desempeño del canal.
Límite superior W: saturación del canal.

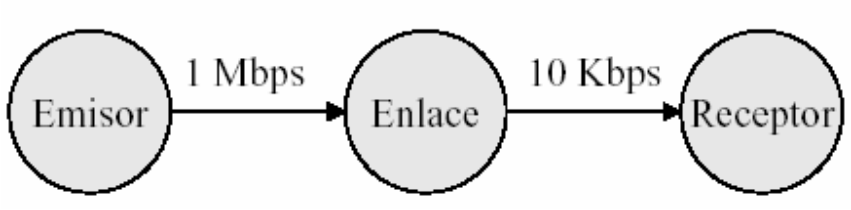
En un control de flujo estático, la pérdida de un mensaje ocasiona que el transmisor retransmita el mensaje.

Mayor tráfico → Pérdida Mensaje → Retransmisión → CONGESTIÓN

El transmisor debe reducir el tráfico que inyecta a la red cuando detecta que está perdiendo mensajes (reduciendo el tamaño de su ventana).

Ejemplo:
Tiempo Emisor-Receptor: 0.5 segundos.
Velocidad de transmisión: 1200bps.
El emisor satura el canal si transmite durante 1 segundo.
Si los mensajes son de 8 bits, la ventana máxima seria de 150 mensajes (mayor sería un desperdicio).

Considerando la red siguiente:



Se puede definir el protocolo de control de flujo como:
- Nodo a nodo
- Extremo a extremo

Nodo a nodo:
Calcular W por separado en ambos tramos (saturar ambos)
Aparece un problema mayor: Enlace recibe datos 100 veces más deprisa que los transmite: Desbordamiento.

Un esquema de control de flujo debería optimizar el uso de:
- Espacio de buffer de los nodos intermedios.
- Ancho de banda de los enlaces entre nodos.

Se hace necesario algún mecanismo de *feedback* entre nodos.

Extremo a extremo:
La capacidad del enlace es la del tramo más lento, y a partir de dicho tramo se calcula W.
Se desperdicia tiempo en los demás tramos.
Si la red evoluciona, se debería adaptar el tráfico ofrecido.
Una solución para reducir el tráfico ofrecido es reducir W.

CONTROL DE FLUJO DINÁMICO

Un método común consiste en que el transmisor disminuya el tamaño de su ventana cada vez que expire un temporizador.

Cuando dejan de expirar, el transmisor incrementa gradualmente el tamaño de su ventana hasta llegar al valor máximo.

Existen diferentes filosofías:
- Disminuir la ventana en uno por cada temporizador que expire, e incrementarla en uno por cada reconocimiento positivo recibido.
- Reducir la ventana a la mitad de su tamaño actual cada vez que un temporizador expire, e incrementarla en uno por cada N reconocimientos positivos recibidos. (Disminución exponencial, comienzo lento).

La *congestión* entonces, puede definirse como una condición en la red donde un aumento en la carga de trabajo ocasiona una degradación en el desempeño.

Capítulo 5: MODELOS DE VALIDACIÓN
Ingeniería de Protocolos y Servicios

5.1. INTRODUCCIÓN

Para especificar y verificar el conjunto de reglas de un protocolo se utiliza un modelo de validación, que es una visión parcial del protocolo, es decir, una visión que se centrará en obtener un juego de reglas consistente y completo.
Se pretende modelar un protocolo lo más simple posible para estudiar su estructura y verificar que sea consistente y completo.
Para describir un modelo de validación será necesario un lenguaje. Existen multitud de lenguajes para validación de protocolos.
Entre ellos hay 3 que son estándar internacional:

- Estelle
- Lotos
- SDL

El lenguaje SDL es un lenguaje orientado a la especificación y descripción del comportamiento de sistemas de telecomunicaciones. El área de aplicación de SDL es la especificación del comportamiento de sistemas que funcionan en tiempo real.
Por ejemplo:

a) Procesamiento de llamadas en sistemas de conmutación.
b) Mantenimiento y tratamiento de fallos en sistemas de telecomunicaciones.
c) Control de sistemas.
d) Funciones de operación y mantenimiento, gestión de redes.
e) Protocolos de comunicación de datos.

Dichas especificaciones son formales en el sentido que permiten análisis e interpretación sin ambigüedades.

- Especificación: descripción del comportamiento requerido del sistema (qué va a hacer el sistema).
- Descripción: descripción del comportamiento del sistema (comportamiento de la implementación).

SDL (Specification and Description Language)
- Fácil de aprender, usar e interpretar
- Extensible a futuros desarrollos
- Soporta múltiples metodologías de especificación y diseño de sistemas.
- Puede usarse para test de descripciones de sistemas, en combinación con MSC (*Message Sequence Chart*) ITUT Rec Z.120 y TTCN (*Tree and Tabular Combined Notation)*

5.2 TIPOS DE SINTAXIS (ITU-T Rec. Z.100 11/99)

SDL permite elegir entre dos formas sintácticas diferentes para la representación de sistemas:
- SDL/GR *(Graphical Representation)*
- SDL/PR *(Phrase Representation)*

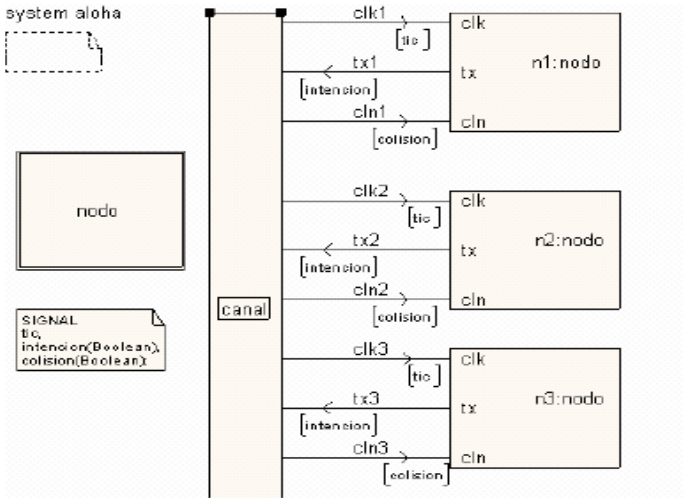
SDL/GR es un lenguaje gráfico que permite visualizar la estructura y flujos de control de un sistema y SDL/PR es un lenguaje de programación, mas apropiado para la utilización de herramientas automatizadas. Ambas sintaxis tienen el mismo modelo semántico.

La entidad principal en SDL es el sistema que se compone de bloques. Los bloques se conectan entre si y con el entorno por medio de canales. Los canales sirven como medio de transporte de las señales entre bloques y con el entorno. Un bloque puede contener varias especificaciones de procesos, y un proceso interactúa con otros procesos y con el entorno por medio de las señales. Un proceso es modelado como una máquina de estados finita extendida, lo cual le agrega el uso de variables, parámetros, acciones y temporizadores.

5.3. DEFINICIONES (ITU-T Rec. Z.100 11/99)

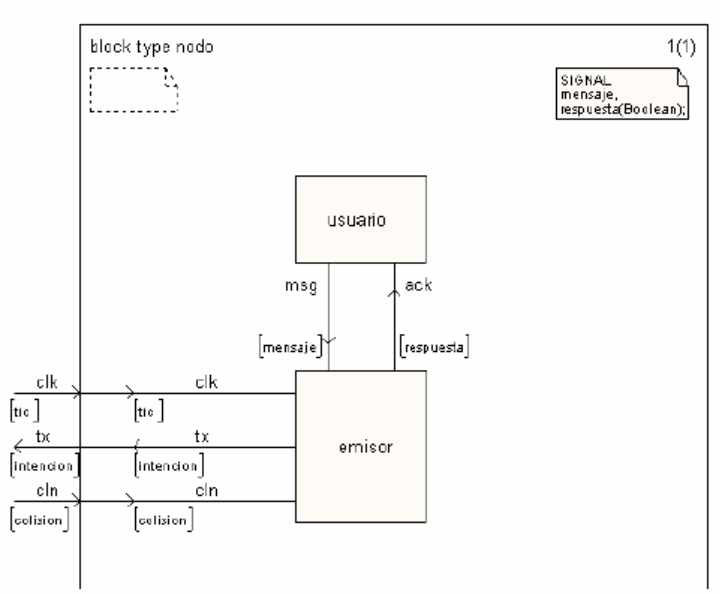
5.3.1 Sistema

Una definición de sistema en SDL es una especificación o descripción de un sistema. Un sistema está separado de su entorno por la frontera del sistema y contiene un conjunto de bloques. La comunicación entre el sistema y el entorno o entre los bloque dentro del sistema sólo puede efectuarse mediante señales. Dentro de un sistema, estas señales son transportadas por canales. Los canales conectan bloques entre sí o con la frontera del sistema. Debe haber por lo menos un bloque dentro del sistema.
También se dispone de un cuadro (SIGNAL) para la declaración de la lista de señales utilizadas.



5.3.2 Bloque

Una definición de bloque es un contenedor para una o más definiciones de proceso de un sistema. La definición de bloque tiene por finalidad agrupar procesos que realizan cierta función. Una definición de bloque proporciona una interfaz de comunicación estática por la cual sus procesos pueden comunicar con otros procesos. Además establece un ámbito para definiciones de proceso. Debe haber por lo menos un proceso dentro del bloque.



5.3.3 Proceso

Una instancia de un proceso es una máquina de estados finita extendida. En el modelo se da una transición de un estado a otro siempre que se recibe una señal válida de otro proceso o del entorno. Al recibir la señal, se pueden realizar acciones de manipulación de datos locales al proceso, o enviar señales a otros procesos o al entorno. Después de efectuada la transición la máquina se encontrará en espera en otro estado. Varias instancias del mismo tipo de proceso pueden existir al mismo tiempo y actuar asincrónicamente y en paralelo, entre sí y con otras instancias de un tipo de proceso diferente del sistema. Las señales recibidas por instancias de proceso se denominan señales de entrada, y las señales enviadas a instancias de proceso se denominan señales de salida. Las señales solo pueden ser consumidas por una instancia de proceso cuando ésta se encuentra en un estado. El conjunto de señales de entrada válidas es la unión del conjunto de señales en todos los canales que conducen al proceso y las señales del temporizador. Todos los procesos tienen acceso al tiempo absoluto (NOW) y pueden realizar mediciones de tiempo y temporizador.

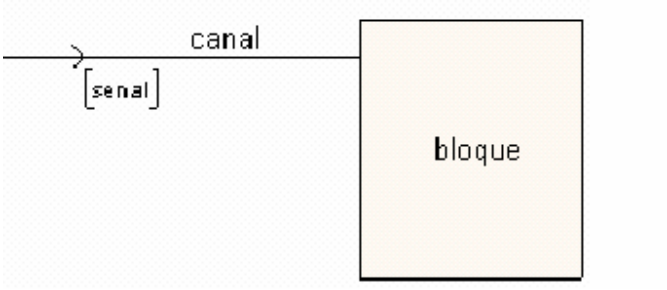
Cada proceso tiene asociado una única cola de señales de entrada, la cual no se comparte con otros. Además para cada estado hay un conjunto de señales de conservación. Cuando el proceso se encuentra en espera en un estado, la primera

señal de entrada cuyo identificador forme parte del conjunto de señales de conservación es extraída de la cola y consumida por el proceso.

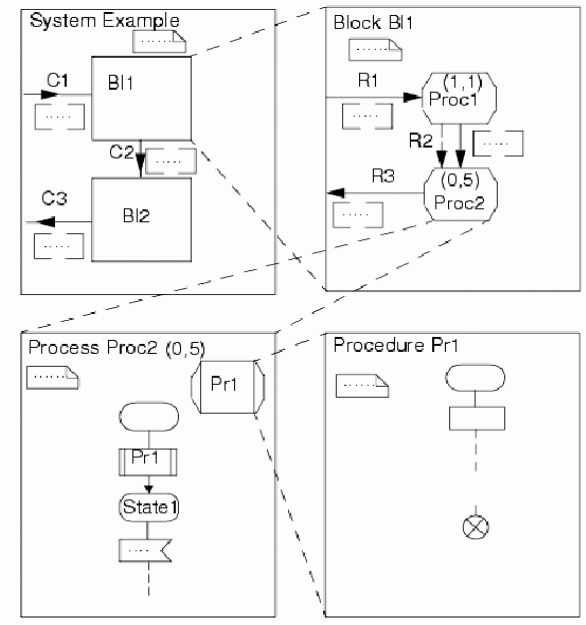
La manipulación de datos se hace por medio de variables locales a cada proceso. SDL permite la definición de cualquier tipo de datos que se necesite, incluidos tipos de datos compuestos (STRUCT).

5.3.4 Canal

Un canal representa una ruta unidireccional de transporte de señales entre dos bloques o entre un bloque y su entorno. Las señales transportadas por canales se entregan al punto extremo de destino. Las señales llegan al punto extremo de destino de un canal en el mismo orden en que fueron enviadas en el punto origen. Pueden existir varios canales entre los dos mismos puntos extremos. Canales diferentes pueden transportar señales del mismo tipo.



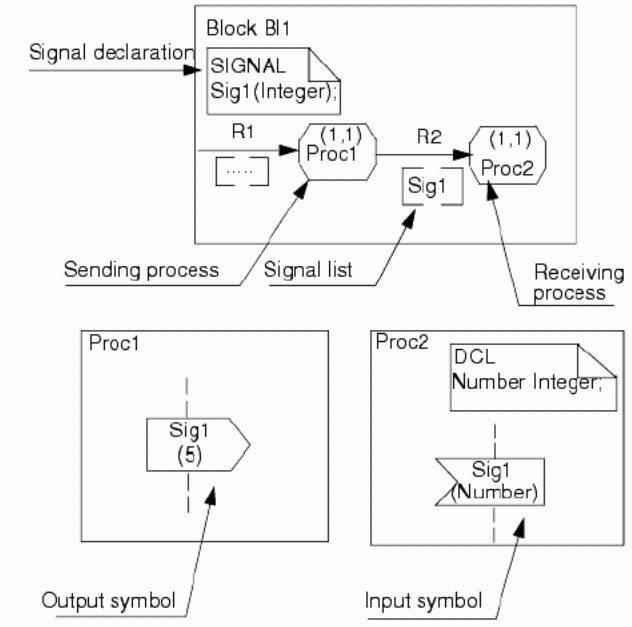
Para cada canal tiene que haber una lista de señales que transporta el canal. Por lo menos uno de los puntos extremos del canal tiene que ser un bloque. Si los dos puntos extremos son bloques, estos tienen que ser diferentes.
La siguiente figura muestra la arquitectura de un sistema SDL.



5.3.5 Señal

Una instancia de señal es un flujo de información entre procesos, siendo también una instanciación de un tipo de señal definido. Por lo menos uno de los puntos extremos de la ruta de señal tiene que ser un proceso. Si los dos puntos extremos son procesos, estos tienen que ser diferentes.

La siguiente figura muestra la comunicación entre dos procesos mediante el envío de señales.



5.3.6 Temporizadores

Todos los procesos pueden utilizar temporizadores y tienen acceso al tiempo absoluto (NOW), que es común a todos los procesos. Una instancia de temporizador es un objeto, en una instancia de proceso, que puede estar activo o inactivo. Cuando un temporizador inactivo es inicializado (SET), se le asocia un valor de tiempo. Si este temporizador no es reinicializado (RESET), o si no es inicializado de nuevo, antes de que el tiempo de sistema llegue a este valor de tiempo, se aplica a la cola de señales de entrada del proceso una señal con el mismo nombre que el temporizador. La misma acción se efectúa si el temporizador es inicializado con un valor de tiempo menor que NOW. Un temporizador está activo desde el momento de la inicialización hasta el momento del consumo de la señal de temporizador. Cuando un temporizador inactivo es reinicializado, sigue estando inactivo. Cuando un temporizador activo es reinicializado, la asociación con el valor de tiempo se pierde, si hay una señal de temporización correspondiente retenida en la cola de entrada, se suprime y el temporizador pasa a inactivo. La inicialización de un temporizador activo equivale a reinicializarlo e inicializarlo inmediatamente después.

Sentencias de definición de temporizadores.
TIMER <nombre> Definición de temporizador.
SET <tiempo>, <nombre> Inicialización de temporizador
RESET <nombre> Reinicialización de temporizador

5.4 SÍMBOLOS EN SDL

INCLUDE: En el sistema se pueden incluir librerías de SDL.

TEXT: Para declaración de variables (DCL) y constantes.

ARRANQUE PROCESO: Símbolo inicial con el que comienza el proceso.

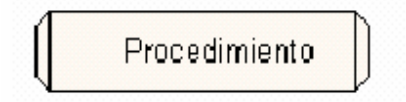
PARADA PROCESO: La parada causa la detención inmediata de la instancia de proceso que la emite. Esto significa que las señales retenidas en la cola de entrada se descartan y que las variables y temporizadores creados para el proceso y la cola dejaran de existir. Se pondrá para indicar donde termina un proceso (si termina).

ESTADO: Un estado representa una condición particular en la cual una instancia de proceso puede consumir una instancia de señal, lo que causa una transición.
Si no hay instancias de señal retenidas, el proceso espera en el estado hasta que se reciba una instancia de señal. Cada símbolo de estado debe tener un identificador del estado.

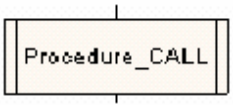
CONSERVACION: El símbolo de conservación está asociado a los estados del proceso. Cada símbolo de conservación debe especificar el tipo de señal que conserva o el símbolo * para conservar todas las señales. Las señales conservadas se retienen en la cola de señales en el orden de su llegada. El efecto de la conservación es válido solamente para el estado al cual está asociada la conservación. En el estado siguiente, las instancias de señal que han sido conservadas se tratan como instancias de señal normales.

ENTRADA: Una entrada permite el consumo de la instancia de señal de entrada especificada. El consumo de la instancia de la señal de entrada pone a la disposición del proceso la información transportada por la señal. A las variables asociadas con la entrada se asignan valores transportados por la señal consumida. Si no hay variable asociada con la entrada para una señal, se descarta el valor.

PROCEDIMIENTO: conjunto de instrucciones que son llamadas desde un proceso o procedimiento. Último objeto en el que se puede descomponer un proceso.



LLAMADA A UN PROCEDIMIENTO: Permite la ejecución de un procedimiento previamente declarado en un proceso o procedimiento.



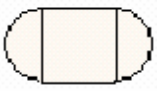
SALIDA: Los valores transportados por la instancia de señal son los valores de los parámetros efectivos en la salida. Si no hay ningún parámetro en la salida en la definición de señal, la señal transporta el valor indefinido.

TAREA: Una tarea puede contener varia sentencias de asignación o especificación de procesos descritos en algún leguaje de programación.

DECISION: Una decisión transfiere el control al trayecto cuya condición contiene el valor dado por la interpretación de la pregunta. Se define un conjunto de respuestas posibles a la pregunta, cada una de las cuales especifica el conjunto de acciones a interpretar para esa elección de trayecto.
Si Condicion = true hacer una cosa; si Condicion = false hacer otra cosa.

CONECTOR: Un conector representa la continuación de un trayecto desde otro conector correspondiente con el mismo número de conecto en la misma área de gráfico de proceso.

INICIO PROCEDIMIENTO: Símbolo inicial con el que comienza el procedimiento.



FIN PROCEDIMIENTO: Símbolo final con el que termina el procedimiento.

5.5 EJEMPLO: DEMONGAME

Este ejemplo se usa en los ejemplos de la recomendación ITU-T Rec Z.100 11/99. Consiste en un juego sencillo, donde el usuario puede generar las señales Newgame, Endgame, Probe y Result. Las 2 primeras señales son para iniciar (ignorada si ya se juega) y terminar (ignorada si no se juega) un juego.

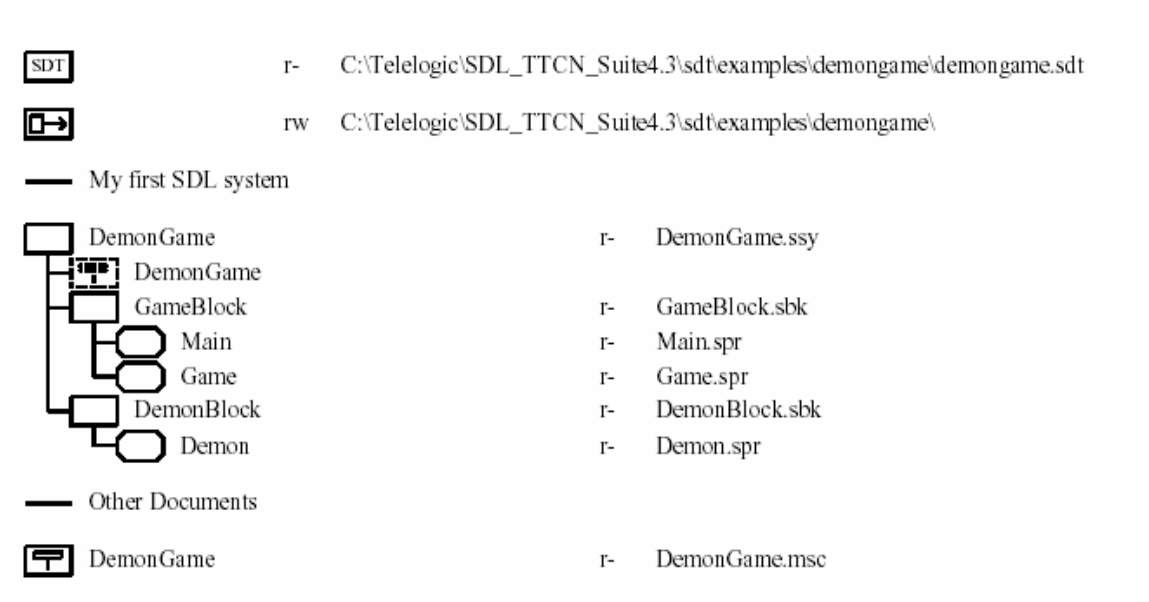
Las reglas del juego son sencillas:

1. Un demon, representado por el proceso Demon, cambia el estado del sistema (winning, losing) de vez en cuando.
2. El usuario debe adivinar cuando el estado es winning.
3. Si el usuario prueba (señal Probe) cuando el estado es winning, gana un punto.
4. Si el usuario prueba cuando el estado es losing, pierde un punto.
5. Para ver el marcador el usuario produce la señal Result, contestada por la señal Score, que contiene un parámetro de tipo entero con el resultado actual.

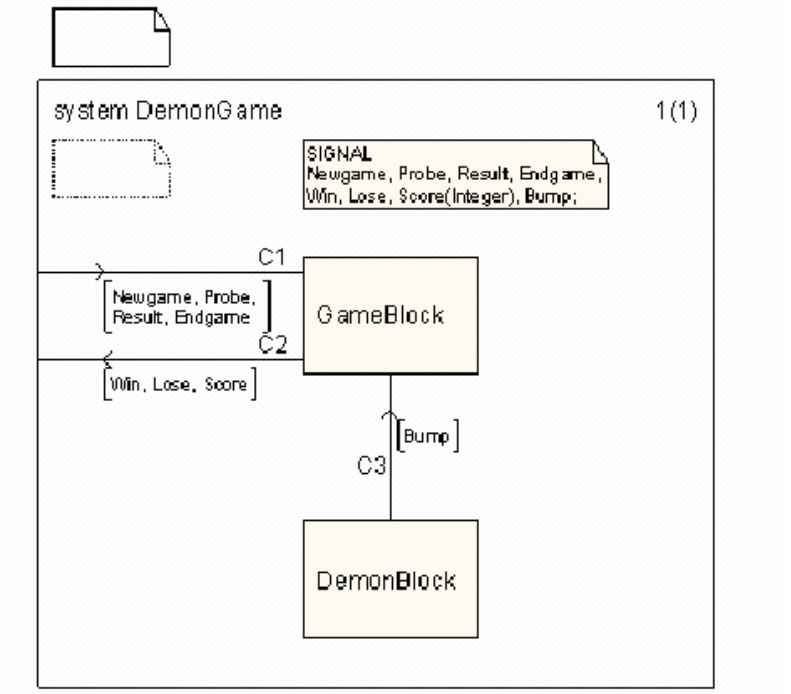
La estructura del sistema es la siguiente:

El sistema DemonGame, compuesto por los bloques

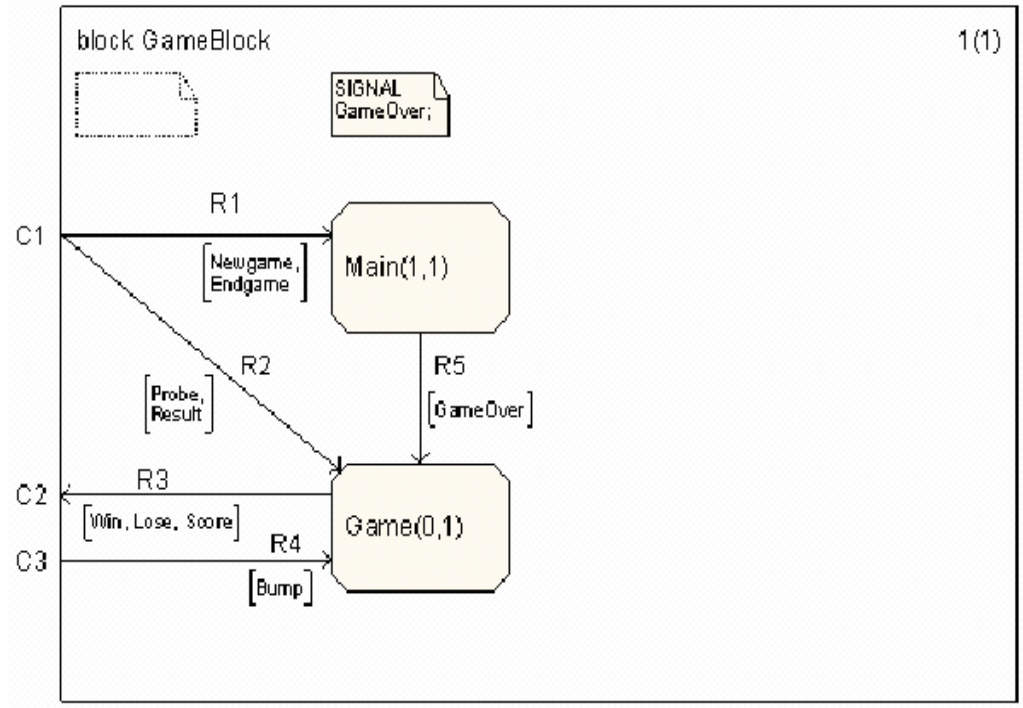
- i.A. Game Block, que está compuesto de los procesos
  - i.A.1. Main
  - i.A.2. Game
- i.B. Demon Block, compuesto por el proceso
  - i.B.1. Demon



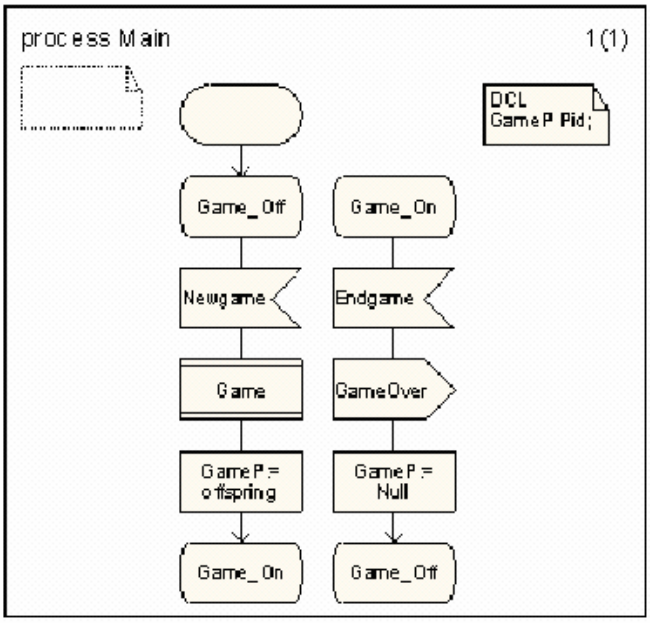
i.- SISTEMA



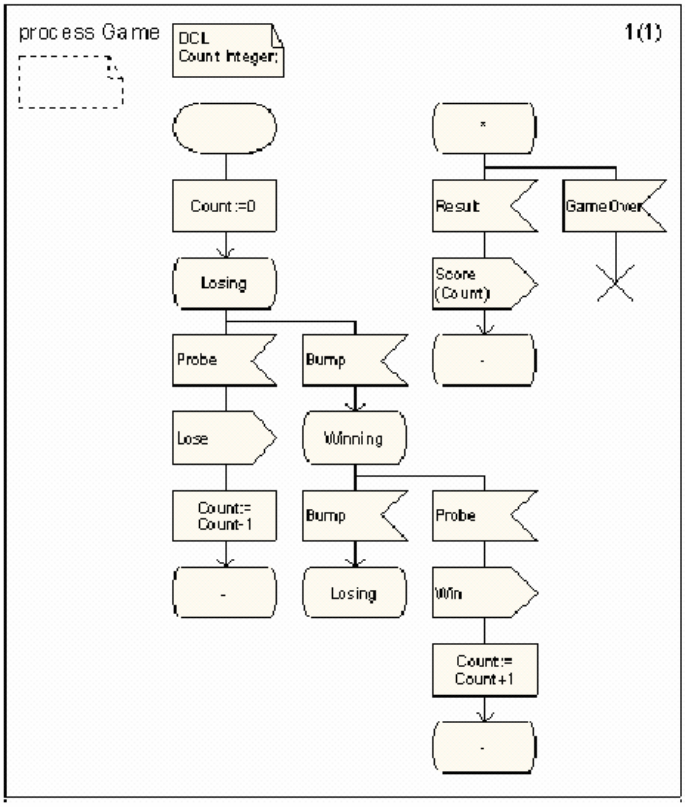
i.A.- Bloque GameBlock



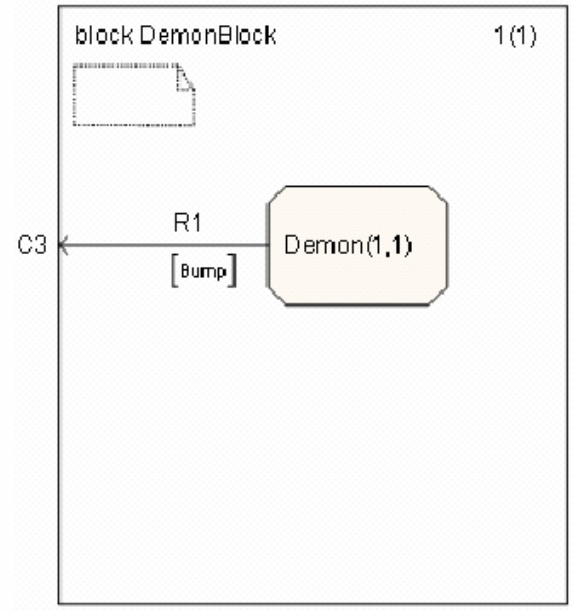
i.A.1.- Proceso Main



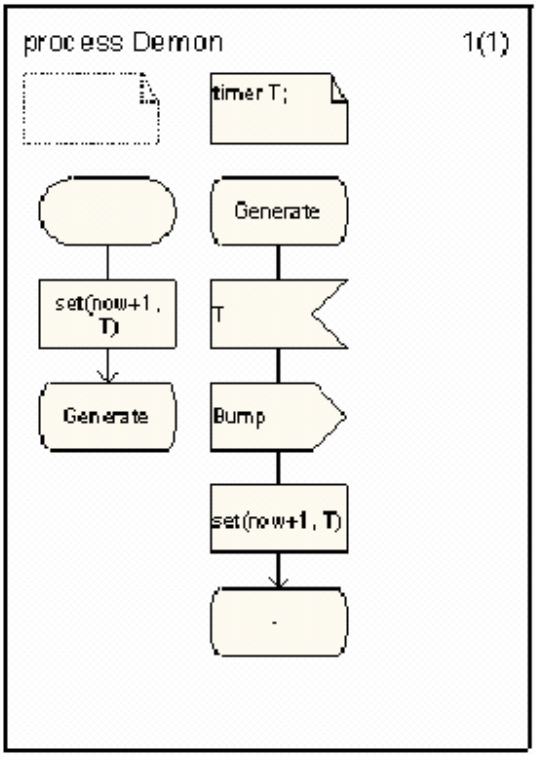
i.A.2.- Proceso Game



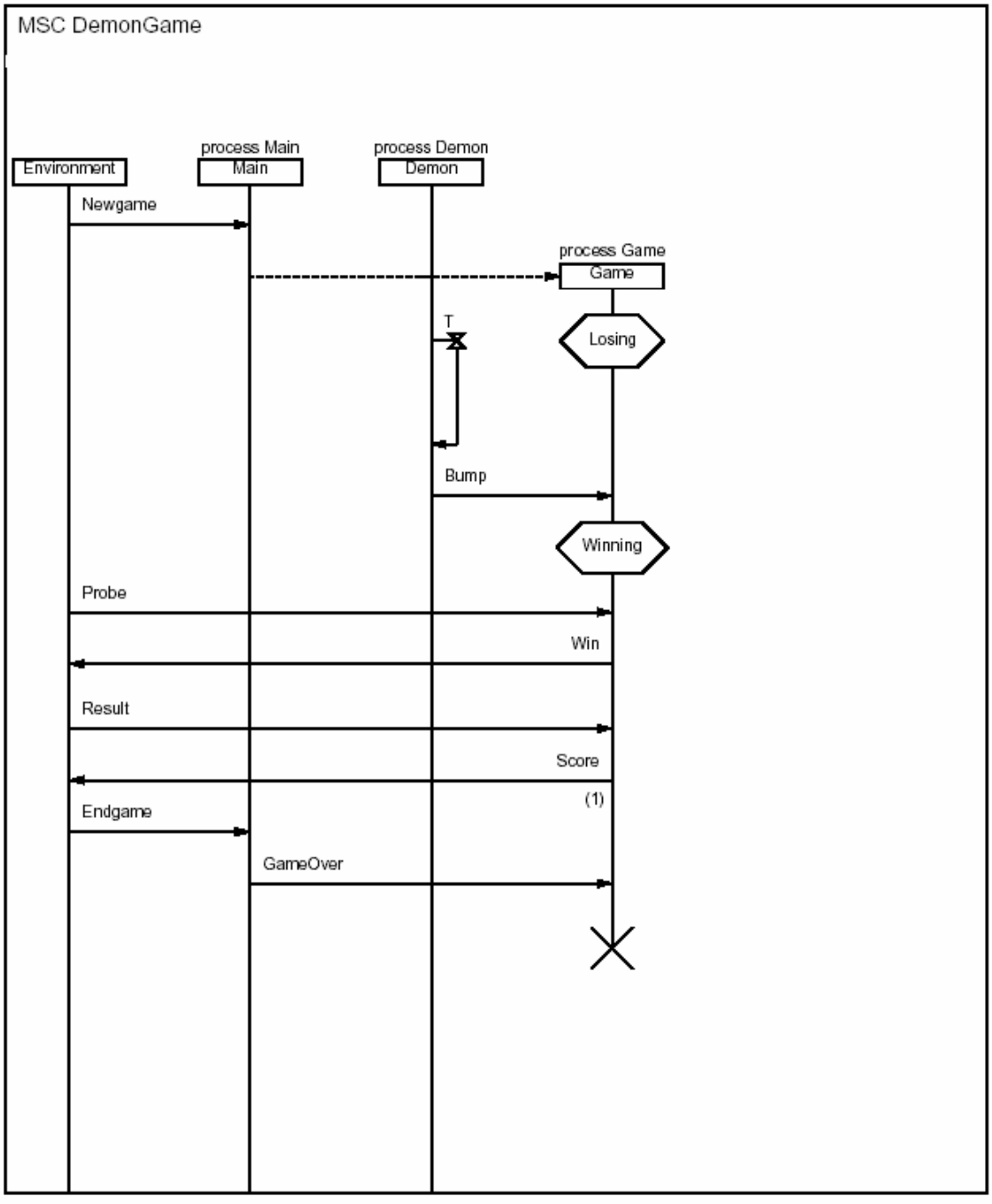
i.B.- Bloque DemonBlock



i.B.1.- Proceso Demon



Este diagrama de secuencia de mensajes (MSC, “Message Sequence Chart”) muestra un ejemplo de una partida de Demongame. En esta partida el usuario arranca el juego, prueba suerte una vez, gana (porque la consulta del resultado *“result”* es 1 *“score”*), y finaliza el juego.



Capítulo 6: REQUISITOS DE CORRECCIÓN
Ingeniería de Protocolos y Servicios

6.1. INTRODUCCIÓN

El lenguaje SDL nos permite desarrollar modelos de validación. Una vez diseñado un modelo, es necesario especificar de manera clara y precisa lo que entendemos por un diseño correcto.
Un diseño puede ser considerado correcto sólo para un conjunto específico de criterios de corrección.
Necesitamos de un formalismo para especificar los criterios de corrección.
Cuanto más expresiva sea nuestra notación, menos útil será en la práctica.
En este tema se estudiarán distintos criterios de corrección.

6.2. RAZONAMIENTOS SOBRE COMPORTAMIENTO

Formalizaremos los criterios de corrección como afirmaciones sobre el comportamiento de un modelo de validación.
Un comportamiento es:

- Inevitable o
- Imposible

Para afirmar que un comportamiento es inevitable, por ejemplo, podemos estipular que todos los comportamientos posibles que nos alejan de él, son imposibles.

• El *comportamiento* de un modelo de validación se define como el conjunto de todas las posibles secuencias de ejecución.
• Una *secuencia de ejecución* es un conjunto finito de estados ordenados.
• Un *estado* queda definido por el conjunto de los valores de las variables locales y globales, los puntos de control de flujo de los procesos en ejecución y por el contenido de todos los mensajes de comunicación.

Ahora bien, una colección de estados cualquiera no tiene porque ser una secuencia de ejecución válida.

Un conjunto finito de estados ordenados es considerado *válido* para un modelo M dado si satisface los siguientes dos criterios:

• El primer estado de la secuencia (p. ej. 1) es el estado inicial de M, con todas las variables inicializadas a 0, los canales vacíos, y un solo proceso activo y en su estado inicial.
• Si M se encuentra en el estado i, existe al menos una instrucción ejecutable que puede llevar M del estado i al estado i+1.

Dentro de las secuencias posibles hay 2 tipos que son de especial interés:

• Terminación: Una secuencia es de terminación si no se repite ningún estado en ella y el modelo M no tiene ninguna instrucción ejecutable cuando se le sitúa en el último estado de la secuencia.

• Cíclica: Una secuencia es cíclica si todos sus estados menos el último son distintos, y el último estado de la secuencia es igual a uno de los anteriores.
Las secuencias cíclicas definen ejecuciones potencialmente infinitas.

Todas las secuencias de terminación y cíclicas que pueden ser generadas al ejecutar un modelo M definen juntas el *"comportamiento del sistema"* para ese modelo M.
La unión de todos los estados incluidos en el comportamiento del sistema define el *"conjunto de estados alcanzables"* del modelo M.

6.3. ASERCIONES

Los criterios de corrección pueden expresarse como condiciones booleanas que deben ser satisfechas cuando un proceso alcanza un determinado estado. Las aserciones se aplican a un estado determinado.

6.4. INVARIANTES DEL SISTEMA

Son una generalización de las aserciones. Una invariante del sistema es una condición booleana que siendo verdadera en el estado inicial del sistema, permanece verdadera en todos los estados alcanzables del sistema, sin importar la secuencia de estados seguida.

6.5. BLOQUEOS

El bloqueo es un estado que alcanza el sistema, a partir del cual cualquier ejecución es imposible.
Una secuencia de bloqueo es un caso particular de secuencia de terminación (normalmente no deseada).

6.6. CICLOS INDESEADOS

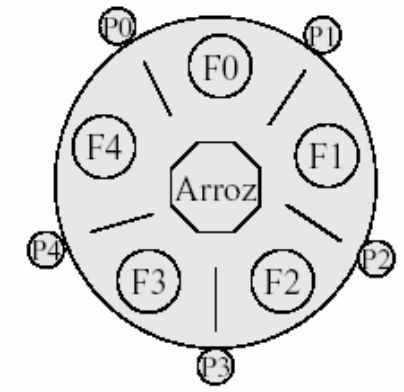
Un ciclo indeseado o *livelock* es un ciclo de ejecución del programa donde a pesar de que se van ejecutando instrucciones, el programa concurrente no avanza en la tarea que supuestamente debe realizar.
Un ciclo indeseado es un caso particular de secuencia cíclica (normalmente no deseada).

6.7. PROCLAMACIONES TEMPORALES

Una proclamación temporal especifica un cierto orden temporal en las propiedades de los estados.
P → Q : Significa cada estado donde la propiedad P es verdadera irá seguido por un estado donde la propiedad Q es verdadera.

6.8. EJEMPLO CLÁSICO: EL PROBLEMA DE LOS 5 FILÓSOFOS

El problema de los 5 filósofos es un problema clásico de la literatura de sistemas concurrentes.
Consiste en 5 filósofos chinos sentados alrededor de una mesa circular, intercalando entre cada 2 filósofos un palillo para comer. En el centro de la mesa hay una fuente de arroz. Un filósofo realiza 2 tareas de manera cíclica: Pensar y comer.



Con distintas soluciones del problema se puede ver:
• Bloqueo: Ningún filósofo -proceso- logra comer - avanzar- (S1).
• *Livelock*: Secuencia donde no se avanza (S2).
• Inanición: Un filósofo -proceso- puede quedarse sin comer si sus vecinos así lo deciden (S2).
• Paso de mensajes: Los filósofos -procesos- pasan a comer enviando palillos -mensajes- (S4, S5) .
• Sincronización de condiciones (S*): Un filósofo no puede comer hasta tener ambos palillos.
• Exclusión Mutua (S*): Cada palillo solo puede ser utilizado por un filósofo a la vez.
• Variables Compartidas (S*): Un palillo puede verse como una variable que va a ser compartida por 2 filósofos.
• Espera activa (S2): Un filósofo puede quedarse a la espera de un palillo continuamente preguntando.

Solución 1 (S1), memoria compartida. Con bloqueo.
Cada filósofo se representa con un proceso.
Cada palillo con una variable booleana {libre, ocupado}.
Las reglas de procedimiento son coger el palillo de la derecha y luego el de la izquierda para comer. Liberarlos para pensar.

Solución 2 (S2), memoria compartida. Con *livelock*.
Cada filósofo se representa con un proceso.
Cada palillo con una variable booleana {libre, ocupado}.
Las reglas de procedimiento son coger el palillo de la derecha y luego el de la izquierda para comer. Si se ha cogido el palillo derecho y el izquierdo esta ocupado, liberar el derecho y reintentar. Liberarlos para pensar.

Solución 2 (S2), memoria compartida. Con bloqueo.
Esta solución puede llevar a inanición a un filósofo, si sus vecinos comen de manera alternada, y nunca pensando de manera concurrente.

Solución 3 (S3), memoria compartida. Sin bloqueo.
Cada filósofo se representa con un proceso.
Cada palillo con una variable booleana {libre, ocupado}.
Las reglas de procedimiento son coger el palillo de la derecha y luego el de la izquierda para comer, excepto el filósofo 0, que es zurdo. Liberarlos para pensar.

Solución 4 (S4), paso de mensajes. Sin bloqueo. Solución centralizada.
Cada filósofo se representa con un proceso. Hay un proceso que distribuye los palillos (camarero).
Las reglas de procedimiento son mandar un mensaje al camarero para pasar a comer. Éste responderá cuando los palillos del filósofo en cuestión queden libres. De manera equivalente, mandar un mensaje para liberarlos y pensar.

Capítulo 7: DISEÑO DE PROTOCOLOS
Ingeniería de Protocolos y Servicios

7.1. INTRODUCCIÓN

Se pretende aplicar lo visto sobre estructuración y especificación a un ejemplo de diseño concreto: Diseño de un protocolo para transferir ficheros entre 2 máquinas asíncronas.

Se especificarán los cinco elementos esenciales del protocolo, construyendo un modelo para especificar las reglas. Finalmente (herramientas automáticas) se comprobará si se satisfacen los requisitos de diseño.

Se implementará un prototipo de alto nivel (modelo), y una vez validado se obtendrá un diseño a partir de él.
Para el proceso de diseño se supondrán 2 cosas:
• El diseño de un protocolo es iterativo: La primera vez no será correcto, seguramente la segunda tampoco…
• El diseñador suele estar convencido de que el diseño no presenta errores: Una simple inspección manual hará aparecer errores esperados. Una automática inesperados.

PROTOCOLO DE TRANSMISIÓN DE FICHEROS

El protocolo a desarrollar se puede clasificar como “punto a punto” (1 emisor, 1 receptor). Además dará un servicio “extremo a extremo” entre 2 usuarios de 2 máquinas.
La definición del problema se hará especificando el servicio, las suposiciones, el vocabulario y el formato.

7.2. ESPECIFICACIÓN DEL SERVICIO

El protocolo debe implementar un servicio extremo a extremo de transferencia de ficheros fiable. Incluye el establecimiento y liberación de la comunicación, la recuperación de errores de transmisión, una estrategia de control de flujo para no desbordar al receptor. Debe ser capaz de transmitir ficheros ASCII, uno detrás de otro, con probabilidad de error menor que 1 cada 10^8 bits.
Debe ser capaz de recuperar mensajes perdidos.
El usuario puede abortar una transmisión.

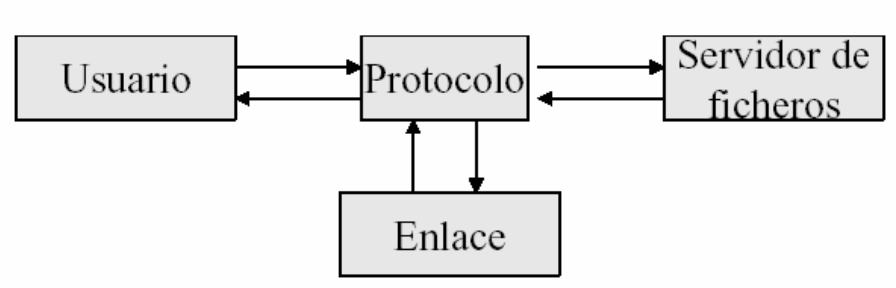
7.3. SUPOSICIONES SOBRE EL CANAL

El protocolo transferirá en modo full-dúplex sobre línea telefónica dedicada. La transferencia será entre Cartagena y Barcelona (700Km). Con la velocidad de propagación en cable de 30000Km/s el tiempo mínimo aproximado es 0,0233

segundos. Se usan módems a 1200bps, transmitiendo caracteres del código ASCII. Se suponen ráfagas de errores de longitud media 10 milisegundos -12bits- (Mandelbrot) e intervalo medio libre de errores de 100 segundos -125000bits- (Poisson).

7.4. VOCABULARIO DEL PROTOCOLO

Si se toma al protocolo como una caja negra se tiene:



Sin entrar en detalles del protocolo, los mensajes mínimos necesarios serán:
- Desde usuario:
    - transfer(file_descriptor): Inicia una transferencia
    - abort: Interrumpe una transferencia en curso

- Hacia Servidor de ficheros:
    - open(file_descriptor): Abre un fichero

- Hacia el sistema remoto:
    - connect(size): Petición de conexión

- De protocolo remoto a servidor de ficheros remoto:
    - create(size): Petición de creación de fichero

- Respuesta:
    - accept
    - reject(status)

Para transferir un fichero deben suceder 4 eventos:
1.- Establecimiento conexión con servidor de ficheros local.
2.- Establecimiento conexión con sistema remoto.
3.- Transferencia de datos.
4.- Desconexión ordenada.

Para la fase 3 se necesita un mensaje para obtener datos del servidor de ficheros y transmitirlos.

- data(cnt, ptr): lee cnt datos de ptr y los transmite.

Para indicar fin de fichero, el emisor usará:

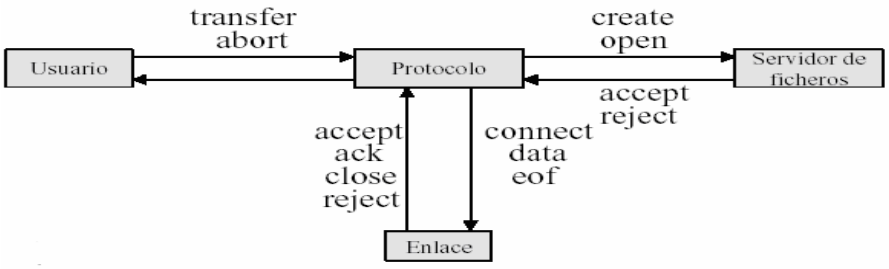
- eof

El protocolo remoto confirmará con:

- close

Para la implementación del control de flujo:

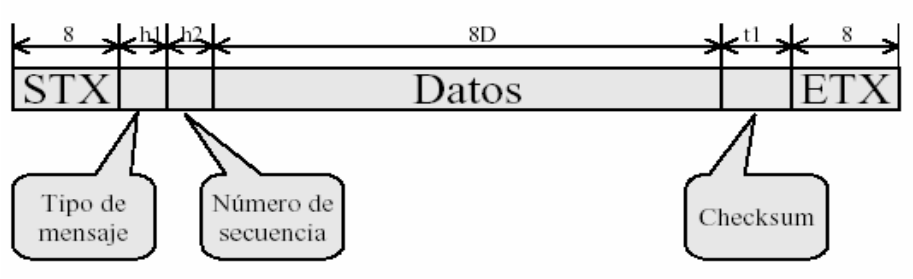
- ack



7.5. FORMATO DE LOS MENSAJES

Como mínimo:
- Un campo de tipo de mensaje y otro opcional de datos.
- Número de secuencia para el control de flujo.
- Checksum para la detección de errores.
- Canal orientado a byte: El mensaje será una secuencia de bytes. ¿Tamaño secuencia? Lo mínimo posible.

De alto nivel a bajo nivel:
- Mensaje como serie de bytes que indican tipo y valor de los parámetros.
- Añadir número de secuencia para el control de flujo.
- Añadir checksum para el control de errores.
- Añadir STX y ETX (y *byte stuffing*).



Campo “tipo de mensaje”:
11 posibles tipos de mensajes diferentes: $2^3 < 11 <= 2^4$ con h1=4 es suficiente.

Campo “número de secuencia”:
Retardo propagación 0,023 seg.
A 1200bps son 28 bits.
Ida y vuelta 56 bits.
(M=2W) Falta saber el tamaño del mensaje para saber cuantos “caben” en el canal.
Si el mensaje es mayor que 56 bits con h2>=2 no se pierde tiempo (saturación del canal).

Campo “datos”:
El tamaño del mensaje mínimo (datos+overhead) debería ser 56 bits.
El tamaño máximo debería ser 125.000 bits. (EFI).
El óptimo se encuentra entre:

- mensaje corto: overhead elevado.
- mensaje largo: probabilidad de error aumenta.

Para obtenerlo, supóngase
t= overhead
d= datos
a= ack
pd= probabilidad de error en el mensaje,
pa= probabilidad de error en el ack.

La eficiencia, si R es el numero esperado de transmisiones por mensaje, es:

$$E=d/R(d+t+a)$$

(Si R=1 -no hay errores- la eficiencia máxima es con $d=\infty$)

Para calcular R:
prob. Retransmisión = $p_r = 1-(1-p_a)(1-p_d)$
prob. Transmision OK en intento i = $p_i = (1-p_r)p_r^{\,i-1}$

Así:

$$R = \sum_{i=1}^{\infty} i \cdot p^i = \sum_{i=1}^{\infty} i(1-p_r)\,{p_r}^{i-1} = (1-p_r)\sum_{i=1}^{\infty} i{p_r}^{i-1} = \frac{1}{1-p_r}$$

$$\sum_{i=1}^{\infty} i{p_r}^{i-1} = \sum_{i=0}^{\infty} i{p_r}^{i-1} = \sum_{i=0}^{\infty} \frac{d({p_r}^i)}{dp_r} = \frac{d(\sum_{i=0}^{\infty} {p_r}^i)}{dp_r} = \frac{d(\frac{1}{1-p_r})}{dp_r} = \frac{1}{(1-p_r)^2}$$

Para hallar la máxima eficiencia E se puede derivar respecto d.
Sustituyendo los valores conocidos:
a = STX + h1 + h2 + t1 + ETX = 8 + 4 + 2 + 16 + 8 = 38
t = a = 38
Y mediante aproximación de primer orden:
$p_d = (d + 38)/125 \cdot 103$ y $p_a = 38/ 125 \cdot 103 = 3{,}04 \cdot 10^{-4}$

$$R = \frac{1}{1-p_r} = \frac{1}{(1-p_d)(1-p_a)}$$

$$E = \frac{d(1-p_d)(1-p_a)}{d+t+a} = \frac{d(1-\frac{d+38}{125\cdot 10^3})(1-\frac{38}{125\cdot 10^3})}{d+76}$$

E presenta un máximo para d=3004 bits, y la eficiencia máxima para esta *d* es del 95,13%.
Con estos valores se tiene:
$p_d = (3004+38)/(125 \cdot 103) = 243{,}36 \cdot 10^{-4}$

Campo “checksum”:
Si la longitud de los mensajes es bastante inferior a 125000 bits la mayor parte de ellos no presentarán errores.
Las ráfagas, sin embargo, deben detectarse. Como son de duración 10ms.
-12 bits- se puede usar un CRC con polinomio generador de grado mayor de 12.

Se pretende obtener una tasa de error residual inferior a 10^{-8}.
Se propone utilizar el CRC-CCITT 16, que detecta todos los errores simples y dobles, todos los impares, todas las ráfagas de hasta 16 bits, 99,997% de las ráfagas de 17 bits, y 99,998% de las ráfagas mayores de 17 bits.

Las ráfagas ocurrían cada 100 segundos de media, y de duración media 10ms. Esto es una tasa de error medio a largo plazo de 10^{-4}.
De 2 a 3 ráfagas de longitud mayor que 16 no se detectarán de entre 10^5.

Si todas fueran así, 2 o 3 ráfagas no detectadas cada $100 \cdot 10^5$ seg. (aprox. 10^{-6} largo plazo).
Con la distribución de Mandelbrot anterior:
$Pr(rafaga>=17) = 0.009 \cdot e^{-0.009 \cdot 17} = 0.007$
Así, $P(error) < 10^{-8}$

Efectos del redondeo:
Tomando h1=8, h2=8 y t1=16 se obtiene una E óptima (94,5% -antes 95,13%) para el valor d=3370 (422 bytes -antes 376-).

```
struct {
unsigned char type;
unsigned char seqno;
unsigned char data[422];
unsigned char checksum[2];
} message;
```

7.6. REGLAS DE PROCEDIMIENTO

Para la descripción de las reglas de procedimiento, que deben ser consistentes y completas, se utilizará un modelo de validación.
Dicho modelo servirá para generar una implementación del protocolo.

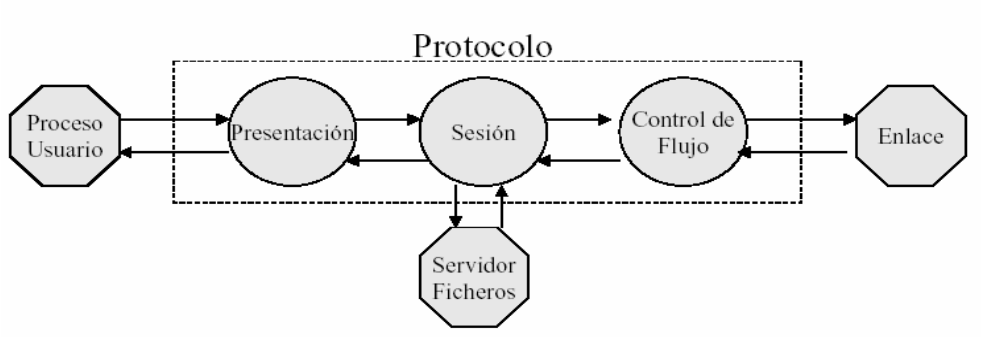
ABSTRACCIÓN

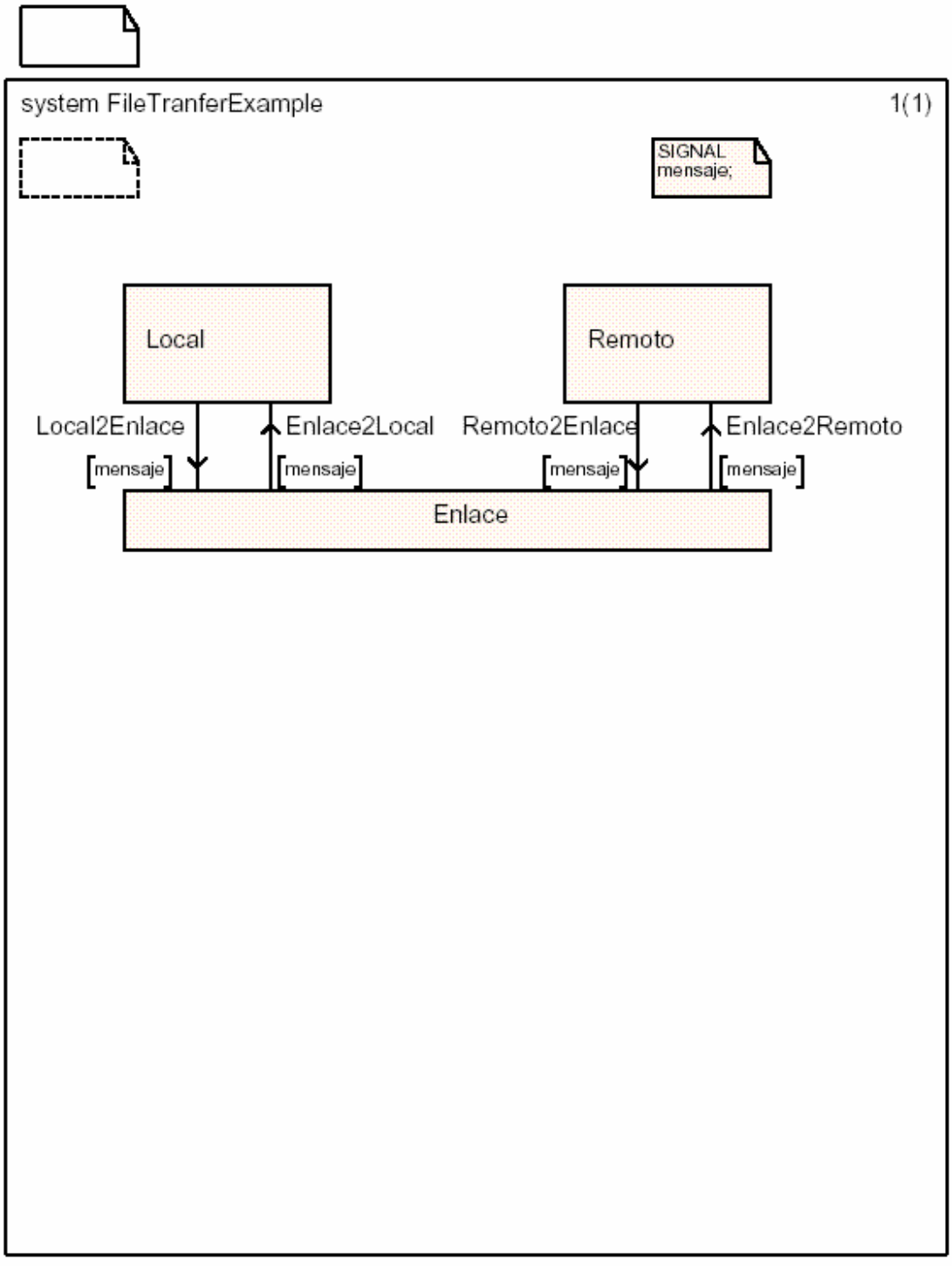
En el modelo, para centrarse en la validación, se omiten detalles innecesarios (por ejemplo, el contenido de los ficheros transferidos).

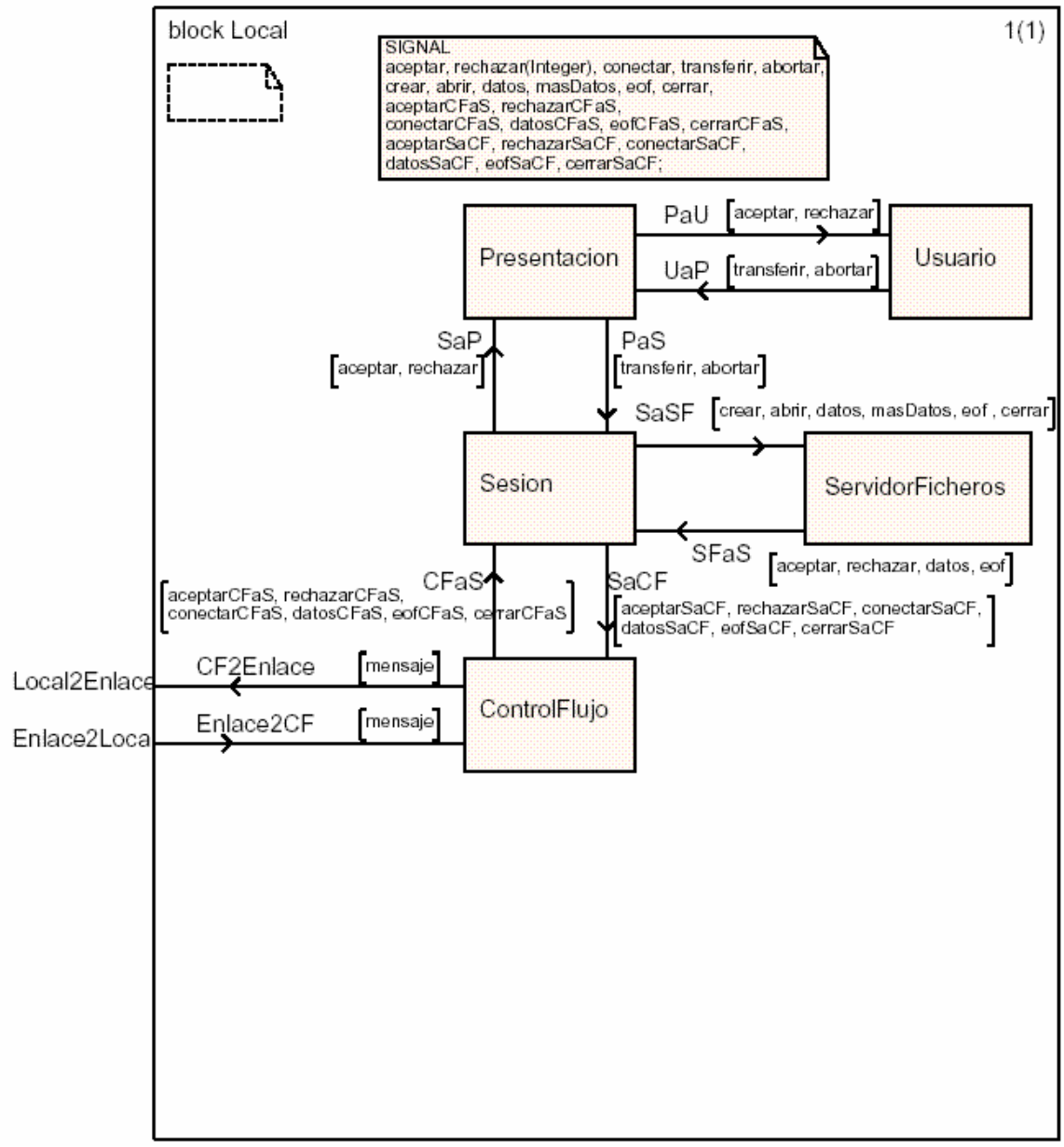
CAPAS

Para estructurar el diseño, éste se dividirá en 3 capas jerárquicas: Presentación, Sesión y Control de flujo.

Jerarquía




system FileTranferExample
1(1)
SIGNAL
mensaje;
Local
Remoto
Local2Enlace
Enlace2Local
Remoto2Enlace
Enlace2Remoto
mensaje
mensaje
mensaje
mensaje
Enlace

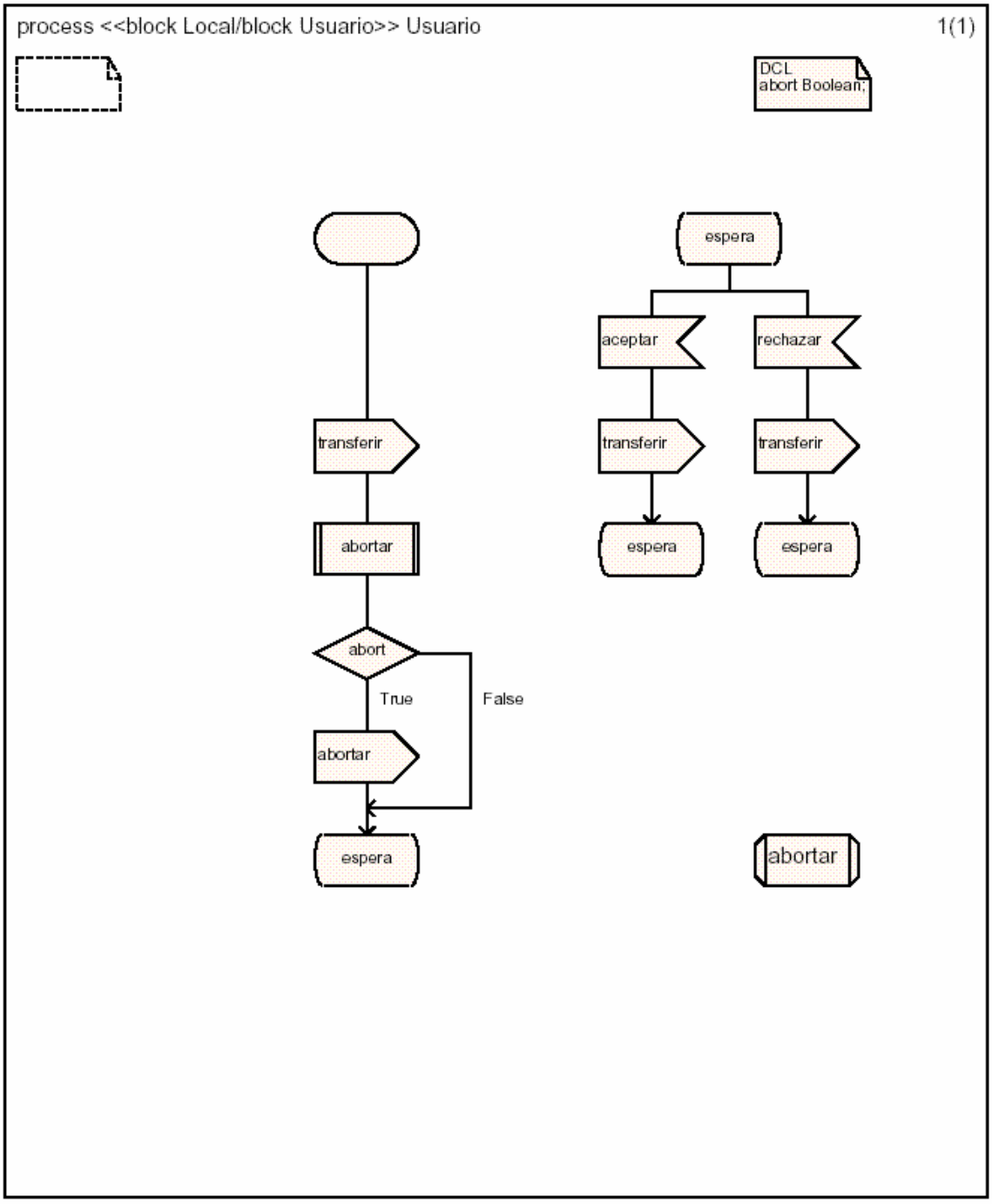

block Local
1(1)
SIGNAL
aceptar, rechazar(Integer), conectar, transferir, abortar,
crear, abrir, datos, masDatos, eof, cerrar,
aceptarCFaS, rechazarCFaS,
conectarCFaS, datosCFaS, eofCFaS, cerrarCFaS,
aceptarSaCF, rechazarSaCF, conectarSaCF,
datosSaCF, eofSaCF, cerrarSaCF;
Presentacion
Usuario
PaU
aceptar, rechazar
UaP
transferir, abortar
SaP
aceptar, rechazar
PaS
transferir, abortar
SaSF
crear, abrir, datos, masDatos, eof , cerrar
Sesion
ServidorFicheros
SFaS
aceptar, rechazar, datos, eof
CFaS
aceptarCFaS, rechazarCFaS,
conectarCFaS, datosCFaS, eofCFaS, cerrarCFaS
SaCF
aceptarSaCF, rechazarSaCF, conectarSaCF,
datosSaCF, eofSaCF, cerrarSaCF
Local2Enlace
CF2Enlace
mensaje
Enlace2Loca
Enlace2CF
mensaje
ControlFlujo

El modelo de validación a construir debe hacer explícitas todas las suposiciones relevantes sobre el entorno.
El entorno del protocolo consta de:

- Proceso usuario
- Servidor de ficheros
- Enlace

Proceso Usuario

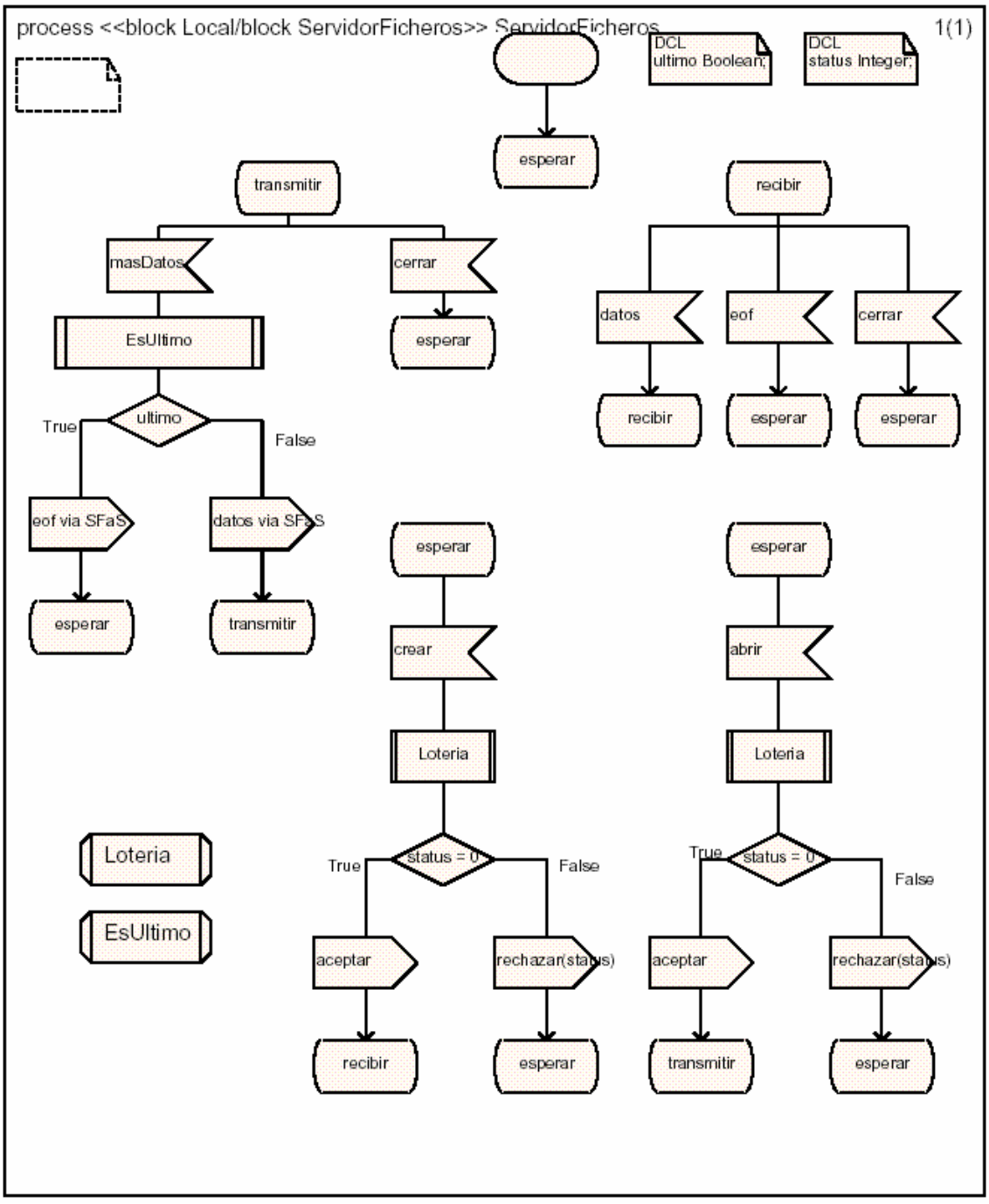
El usuario (uno en cada extremo) puede solicitar una transferencia en cualquier momento.
Después de la solicitud de transferencia, en cualquier momento puede decidir abortar la transferencia.
Se asume que el usuario espera una respuesta de la capa inferior indicando si la transferencia ha sido satisfactoria o no.


process <<block Local/block Usuario>> Usuario
1(1)
DCL
abort Boolean;
transferir
abortar
abort
True
False
abortar
espera
espera
aceptar
rechazar
transferir
transferir
espera
espera
abortar

Proceso Servidor de Ficheros

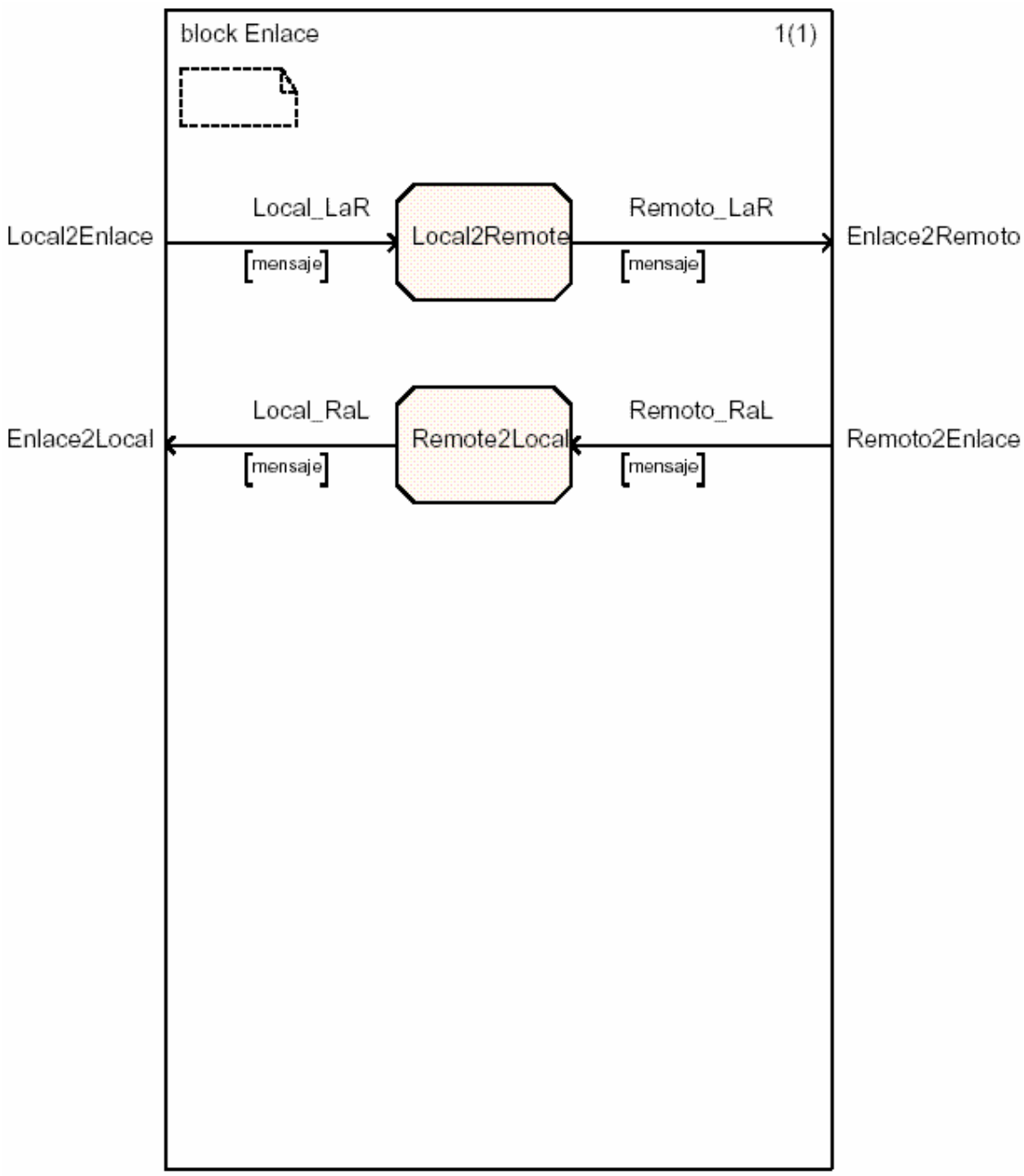
Una petición de crear un fichero empieza con el mensaje *create*. El servidor de ficheros responderá con *reject* o *accept*. En el segundo caso seguirán cero o más mensajes *data*, volviendo al estado inicial al recibir un *eof* o un *close* (transmisión abortada).

Una petición de leer un fichero empieza con el mensaje *open*. El servidor de ficheros responderá con *reject* o *accept*. En el segundo caso se transferirán cero o más mensajes *data*, volviendo al estado inicial al recibir un *eof* o un *close* (transmisión abortada).


process <<block Local/block ServidorFicheros>> ServidorFicheros
1(1)
DCL
ultimo Boolean;
DCL
status Integer;
esperar
transmitir
recibir
masDatos
cerrar
EsUltimo
esperar
datos
eof
cerrar
recibir
esperar
esperar
ultimo
True
False
eof via SFaS
datos via SFaS
esperar
transmitir
esperar
crear
Loteria
status = 0
True
False
aceptar
rechazar(status)
recibir
esperar
esperar
abrir
Loteria
status = 0
True
False
aceptar
rechazar(status)
transmitir
esperar
Loteria
EsUltimo

Proceso Enlace

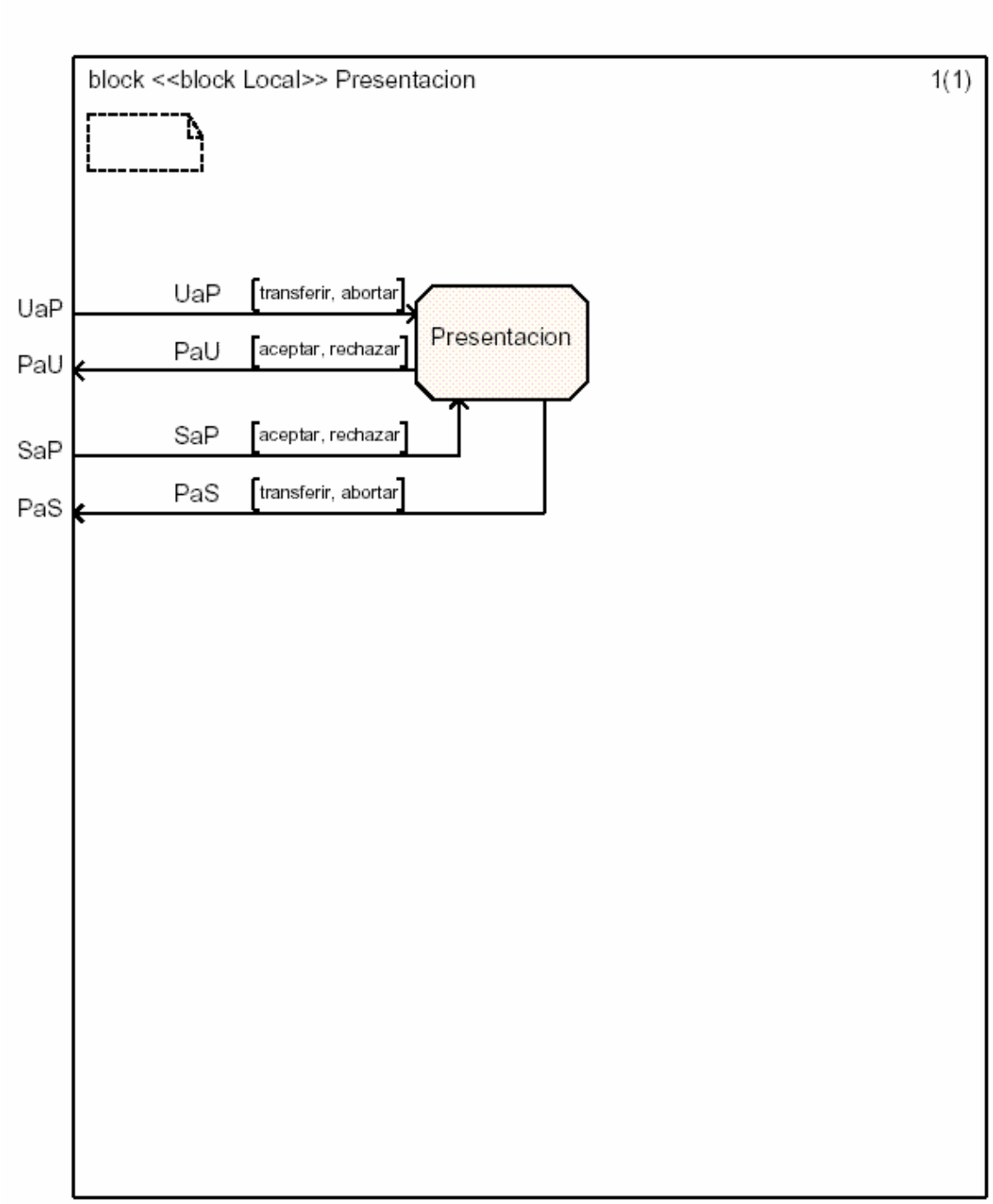
El enlace dispone de un protocolo detector de errores.
Ahora bien, lo que interesa en el modelo es el comportamiento hacia fuera del enlace, que significa que a veces los mensajes atravesarán el enlace y otras veces no.

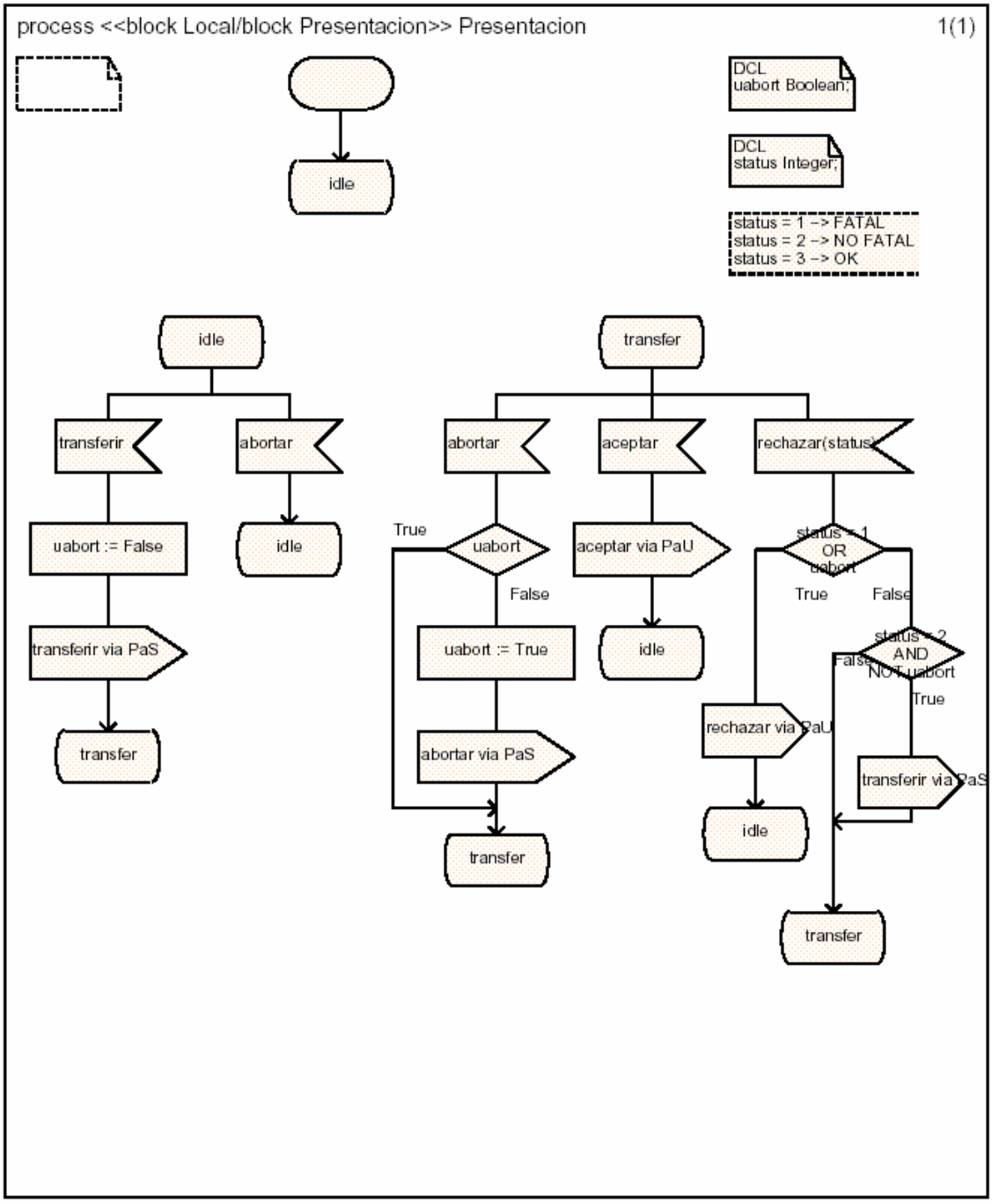


7.6.1. Capa de Presentación

Ofrece el interfaz con el usuario, encargándose de los detalles en la transmisión de ficheros cuando se den errores no fatales.
Se pueden dar si:
• El sistema local está ocupado sirviendo una transferencia entrante.
• Se da una colisión entre peticiones entrante y saliente.

Se dan errores fatales si:
• El sistema de ficheros local rechaza una transmisión porque el fichero no existe.
• El sistema de ficheros remoto rechaza la transmisión entrante porque no dispone de suficiente espacio.
• El usuario aborta la transmisión.


block <<block Local>> Presentacion
1(1)
UaP
PaU
SaP
PaS
UaP
transferir, abortar
PaU
aceptar, rechazar
SaP
aceptar, rechazar
PaS
transferir, abortar
Presentacion

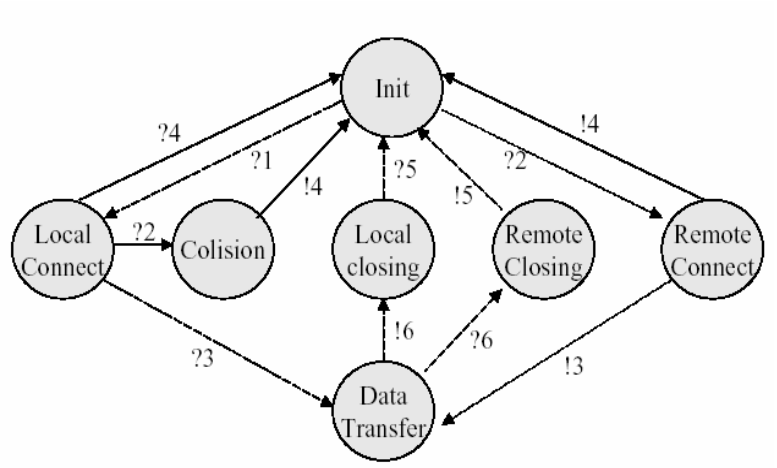

process <<block Local/block Presentacion>> Presentacion
1(1)
idle
DCL
uabort Boolean;
DCL
status Integer;
status = 1 -> FATAL
status = 2 -> NO FATAL
status = 3 -> OK
idle
transferir
abortar
uabort := False
idle
transferir via PaS
transfer
transfer
abortar
aceptar
rechazar(status)
True
uabort
False
uabort := True
abortar via PaS
transfer
aceptar via PaU
idle
status = 1
OR
uabort
True
False
rechazar via PaU
idle
status = 2
AND
NOT uabort
True
transferir via PaS
transfer

7.6.2. Capa de Sesión

Se encargará de las siguientes fases del protocolo:

- Inicialización de los servidores de ficheros.
- Establecimiento de la conexión
- Transferencia de datos
- Finalización de la transmisión

Un diagrama parcial de transiciones de estado muestra el modelo a programar (?=mensaje entrante, !=saliente, 1=*transfer*, 2=*connect*, 3=*accept*, 4=*reject*, 5=*close*, 6=*eof*).
Las flechas a trazos significan una transferencia saliente satisfactoria.
Las flechas a puntos indican una transferencia entrante satisfactoria.



ESTABLECIMIENTO DE COMUNICACIÓN

Al inicio solo deben aceptarse los mensajes *transfer* (local) y *connect* (remoto).
Si se recibe de la capa superior el mensaje *transfer* la capa de sesión debe intentar abrir un fichero local para lectura, debe intentar establecer una comunicación con el sistema remoto, y debe intentar crear un fichero remoto para escritura.

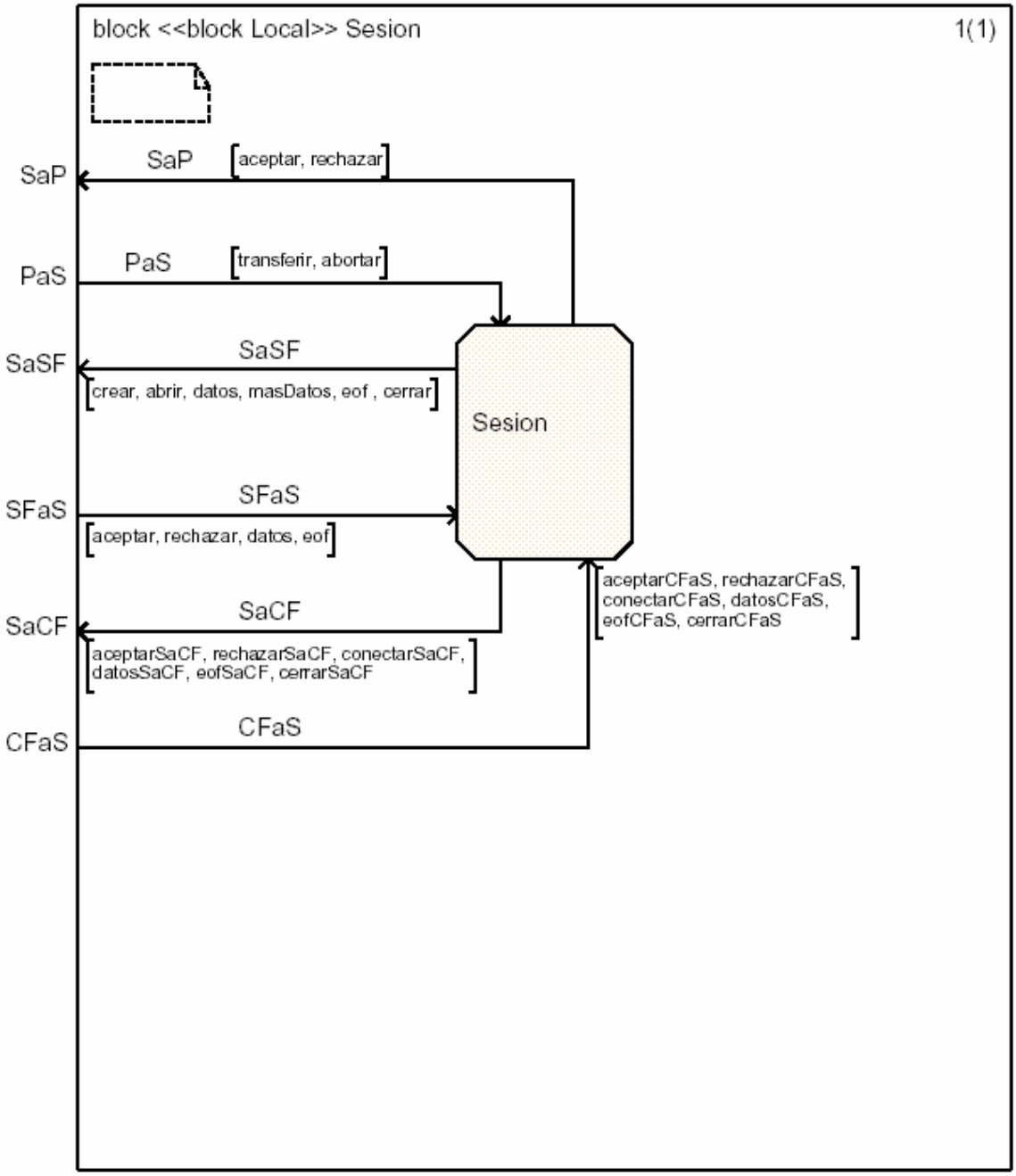
Si se recibe de la capa inferior el mensaje *connect* la capa de sesión sólo debe intentar abrir un fichero local para escritura.

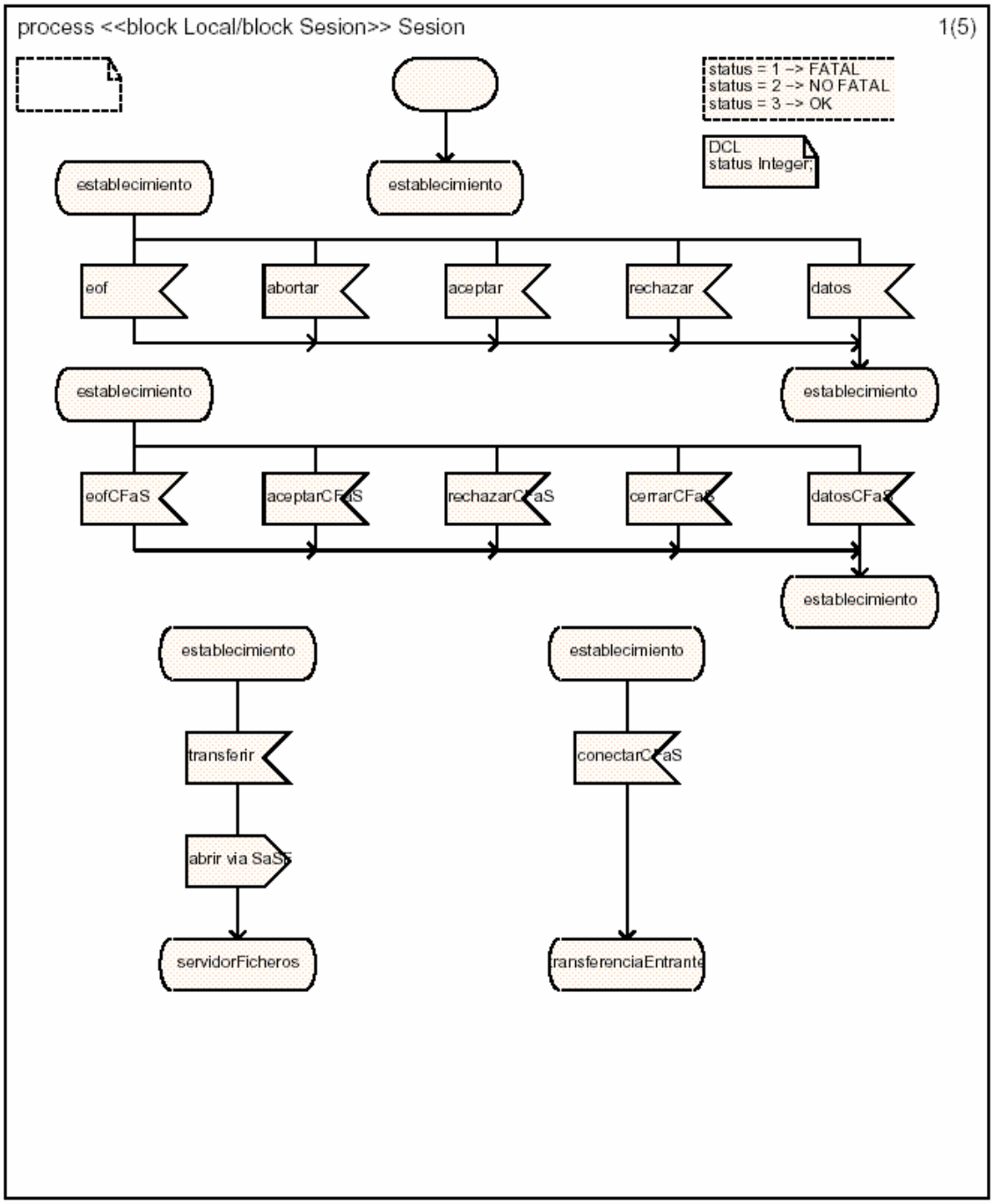
TRANSFERENCIA DE DATOS

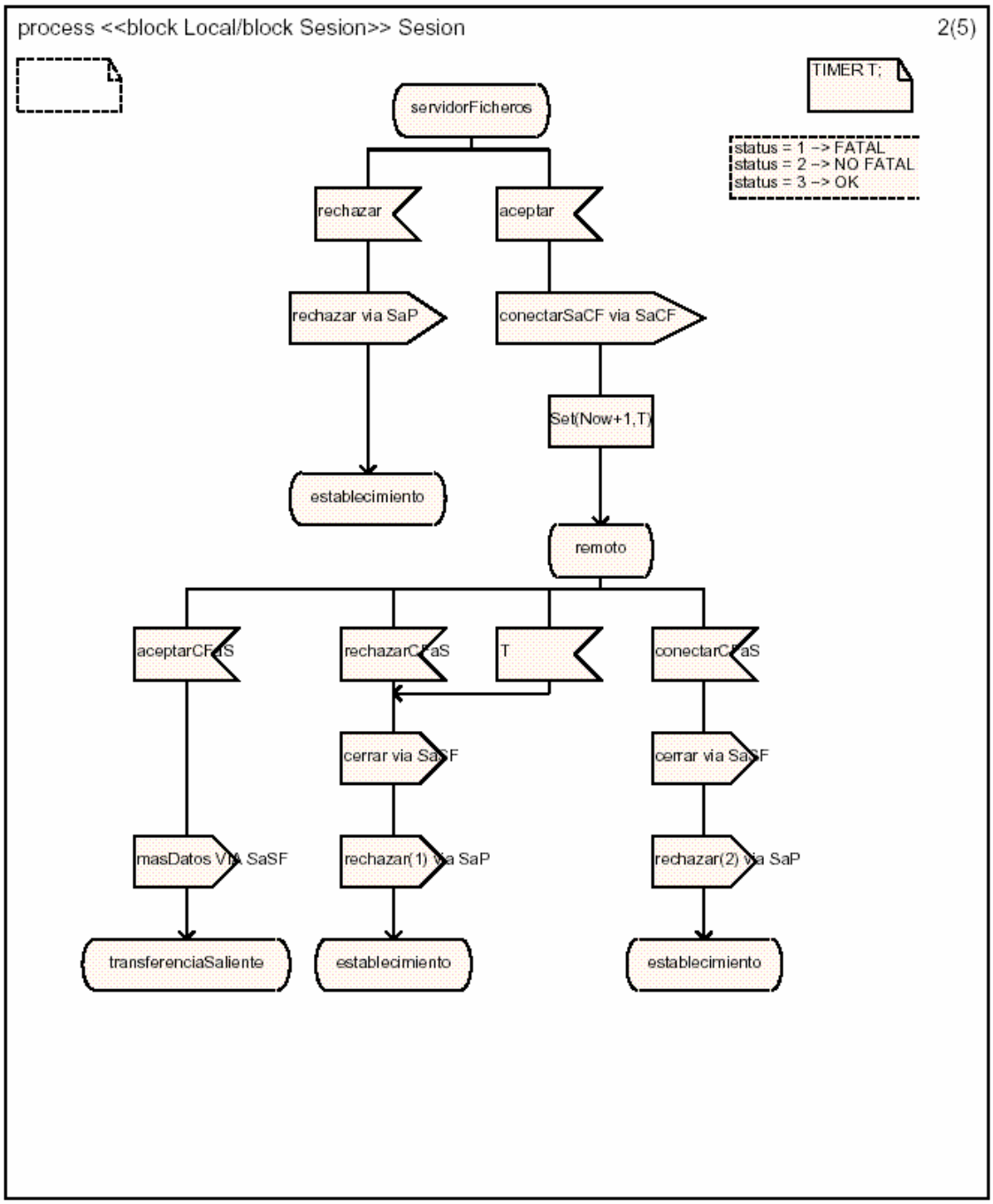
Para transferencias salientes se debe leer del servidor de ficheros y transferir a la capa inferior.
También hay que tener en cuenta la interrupción que pudiera causar un *abort*.
Para transferencias entrantes se recibe de la capa inferior y se escribe en el servidor de ficheros. Si el usuario remoto aborta se recibe *close*. Hay que contemplar el *timeout*.

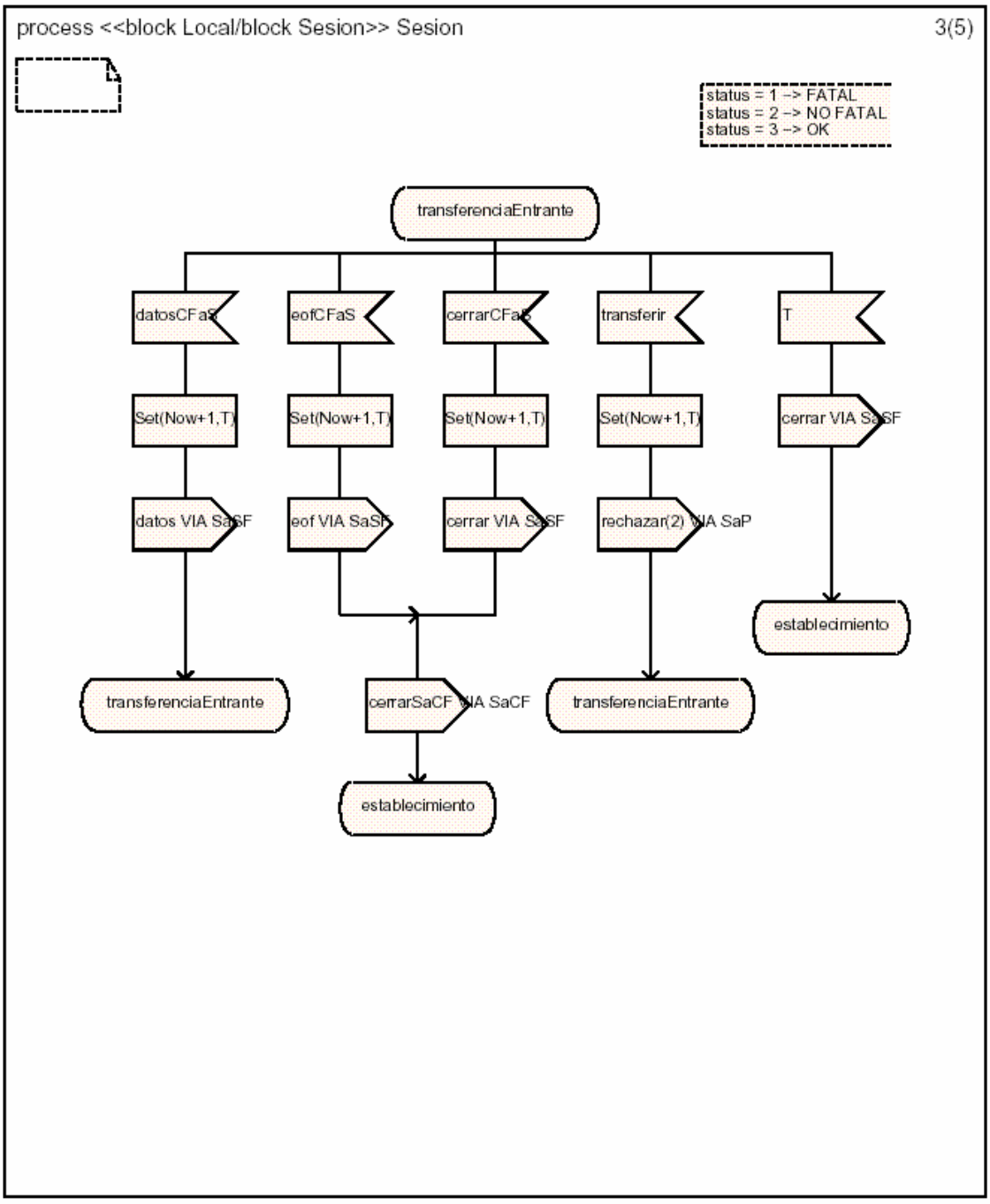
FINALIZACIÓN DE LA COMUNICACIÓN

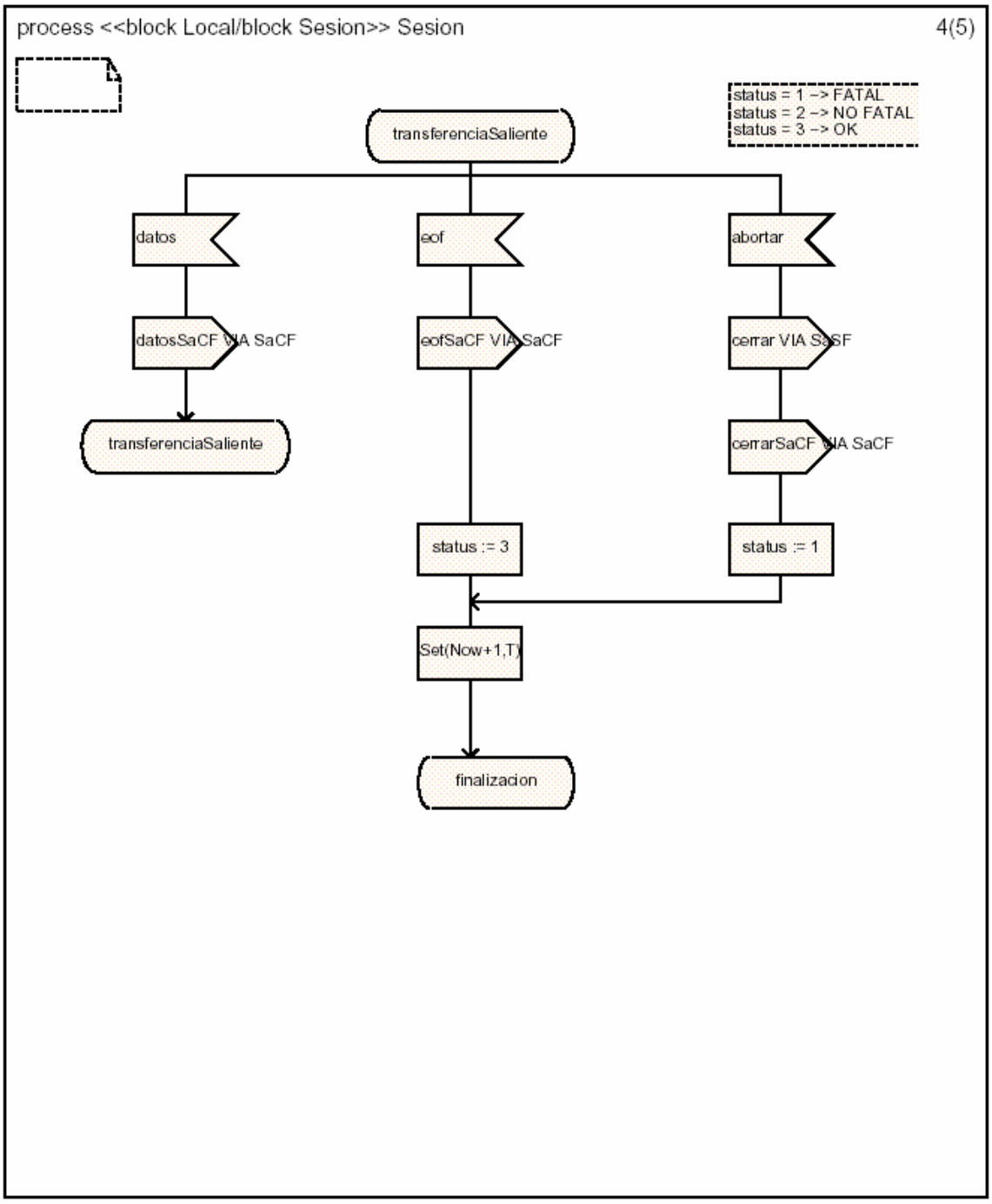
En una transmisión saliente se devolverá a la capa superior *accept* o *reject*, en función del estado final de la transmisión.
En una transmisión entrante se responderá a la capa inferior con *close* para confirmación.

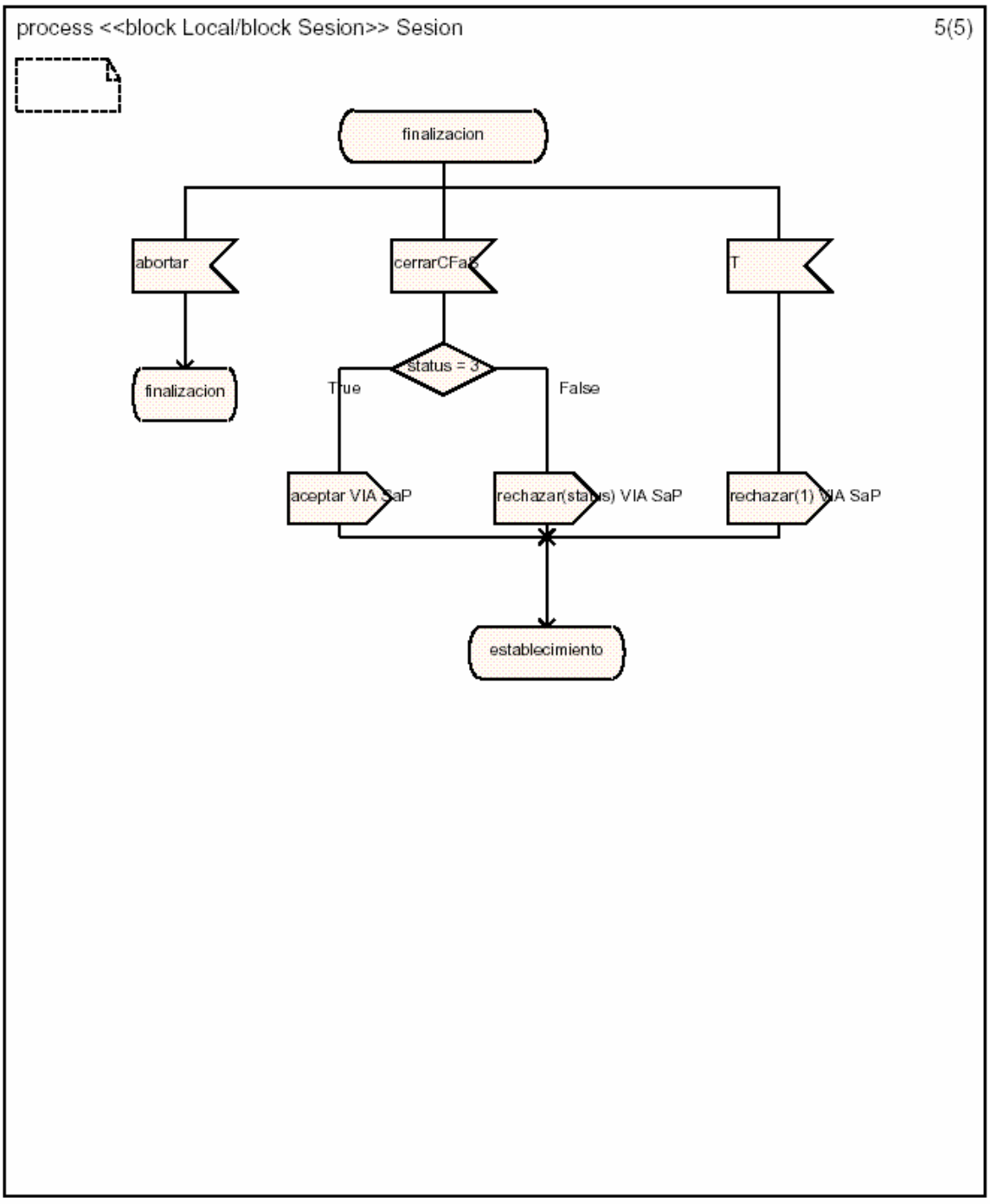



process <<block Local/block Sesion>> Sesion
1(5)
status = 1 -> FATAL
status = 2 -> NO FATAL
status = 3 -> OK
DCL
status Integer;
establecimiento
establecimiento
eof
abortar
aceptar
rechazar
datos
establecimiento
establecimiento
eofCFaS
aceptarCFaS
rechazarCFaS
cerrarCFaS
datosCFaS
establecimiento
establecimiento
establecimiento
transferir
abrir via SaSs
servidorFicheros
conectarCFaS
transferenciaEntrante


process <<block Local/block Sesion>> Sesion
2(5)
TIMER T;
status = 1 -> FATAL
status = 2 -> NO FATAL
status = 3 -> OK
servidorFicheros
rechazar
aceptar
rechazar via SaP
conectarSaCF via SaCF
Set(Now+1,T)
establecimiento
remoto
aceptarCFaS
rechazarCFaS
T
conectarCFaS
cerrar via SaSF
cerrar via SaSF
masDatos VIA SaSF
rechazar(1) via SaP
rechazar(2) via SaP
transferenciaSaliente
establecimiento
establecimiento


process <<block Local/block Sesion>> Sesion
3(5)
status = 1 -> FATAL
status = 2 -> NO FATAL
status = 3 -> OK
transferenciaEntrante
datosCFaS
eofCFaS
cerrarCFaS
transferir
T
Set(Now+1,T)
Set(Now+1,T)
Set(Now+1,T)
Set(Now+1,T)
cerrar VIA SaSF
datos VIA SaSF
eof VIA SaSF
cerrar VIA SaSF
rechazar(2) VIA SaP
establecimiento
transferenciaEntrante
cerrarSaCF VIA SaCF
transferenciaEntrante
establecimiento


process <<block Local/block Sesion>> Sesion
4(5)
status = 1 -> FATAL
status = 2 -> NO FATAL
status = 3 -> OK
transferenciaSaliente
datos
eof
abortar
datosSaCF VIA SaCF
eofSaCF VIA SaCF
cerrar VIA SaSF
transferenciaSaliente
cerrarSaCF VIA SaCF
status := 3
status := 1
Set(Now+1,T)
finalizacion

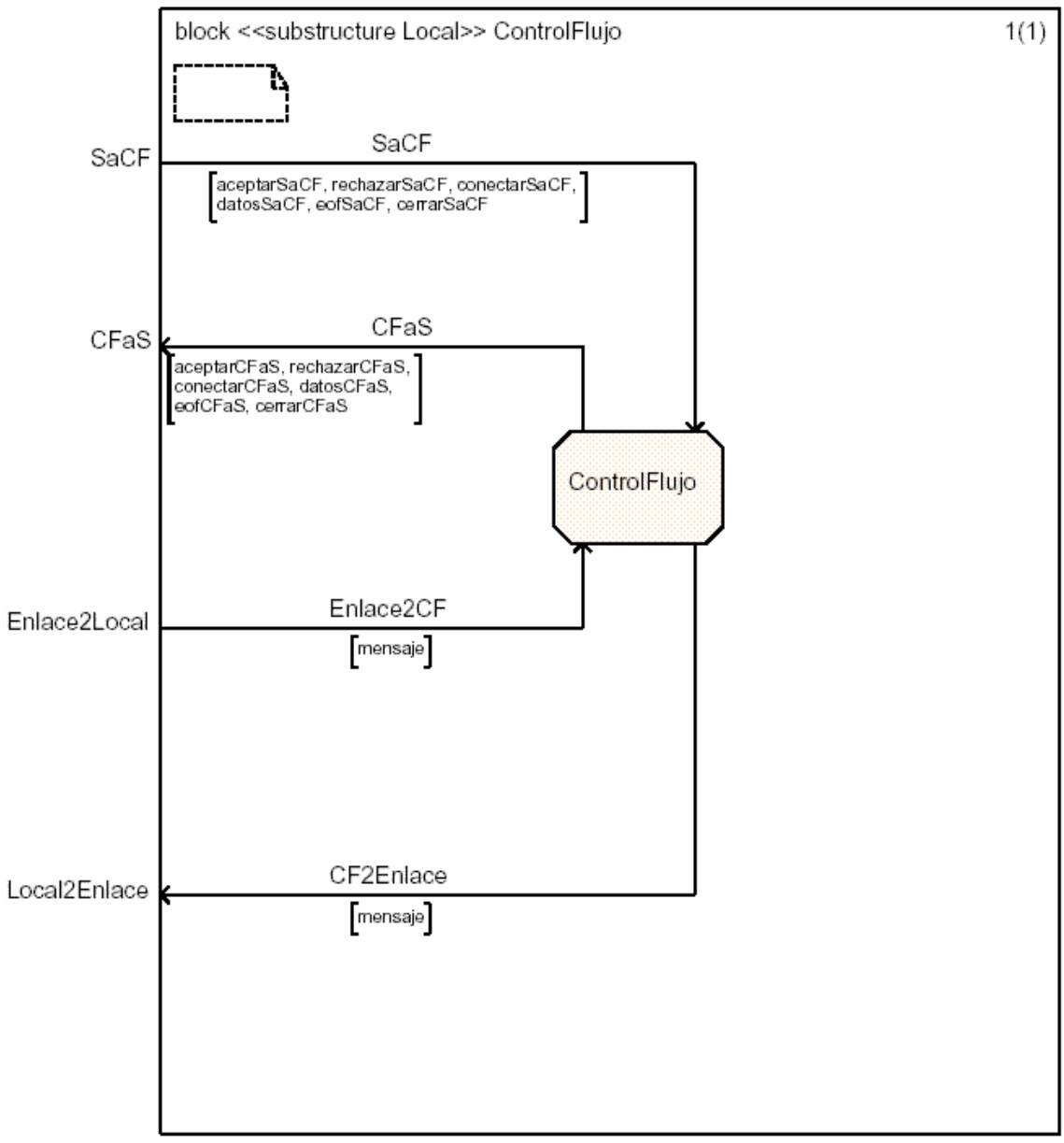

process <<block Local/block Sesion>> Sesion
5(5)
finalizacion
abortar
cerrarCFaS
T
finalizacion
status = 3
True
False
aceptar VIA SaP
rechazar(status) VIA SaP
rechazar(1) VIA SaP
establecimiento

7.6.3. Capa de Control de Flujo

Para el modelado del protocolo de ventana deslizante se debe tener en cuenta los números de secuencia y el tamaño de la ventana.
Los mensajes salientes se almacenan en un bufer por si hay que repetirlos. El proceso repite los pasos de añadir números de secuencia a los mensajes, mandarlos al enlace y recibir el correspondiente ack.

Los mensajes entrantes se desencapsulan y se mandan a la capa superior.
Al recibir un ack se libera el espacio del bufer que ocupaba el correspondiente mensaje. Si era el más viejo se desliza la ventana.
En recepción se mantiene un array de los mensajes que han llegado para enviarlos a la capa superior bien ordenados.

En recepción también se comprueba que un mensaje recibido no lo haya sido antes. Si lo ha sido, hay que comprobar que ha sido reconocido.
Una vez está en la capa superior y el espacio de bufer liberado ya puede reconocerse.


block <<substructure Local>> ControlFlujo
1(1)
SaCF
SaCF
aceptarSaCF, rechazarSaCF, conectarSaCF,
datosSaCF, eofSaCF, cerrarSaCF
CFaS
CFaS
aceptarCFaS, rechazarCFaS,
conectarCFaS, datosCFaS,
eofCFaS, cerrarCFaS
ControlFlujo
Enlace2Local
Enlace2CF
mensaje
Local2Enlace
CF2Enlace
mensaje

DETALLES DE LA IMPLEMENTACIÓN

En el diseño del modelo se han utilizado los detalles relevantes para la validación. Una vez el modelo esté validado, se procederá a obtener una implementación de él. Finalmente hay que comprobar que el diseño realizado se ajusta al modelo (test de conformidad).

Capítulo 8: MAQUINAS DE ESTADO FINITO
Ingeniería de Protocolos y Servicios

8.1. INTRODUCCIÓN

En un nivel bajo de abstracción, un protocolo puede verse como una máquina de estado finito, que es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.
El estado de un protocolo define qué acciones se permiten a cada proceso, qué eventos esperan, y cómo responden a dichos eventos.
El modelo de máquina de estado finito comunicante es muy importante en tres áreas de diseño de protocolos: validación formal, síntesis de protocolos y test de conformidad.

8.2. DESCRIPCIÓN INFORMAL

Normalmente una máquina de estado finito se especifica en forma de tabla de transiciones, por ejemplo:

Condición		Efecto	
Estado Actual	IN	OUT	Siguiente Estado
Q0	-	1	Q2
Q1	-	0	Q0
Q2	0	0	Q3
Q2	1	0	Q1
Q3	0	0	Q0
Q3	1	0	Q1

La tabla especifica el conjunto de transiciones válidas entre estados.
Las 2 primeras columnas son las condiciones que deben ser satisfechas para que la transición pueda llevarse a cabo, que son el estado actual en que la máquina debe estar y una condición del entorno (por ejemplo, una señal de entrada).
Las 2 últimas columnas definen los efectos de la transición, especificando cómo cambiaría el entorno (por ejemplo, señal de salida) y el nuevo estado que alcanzaría la máquina.
El entorno de la máquina consiste en 2 conjuntos de señales finitos y disjuntos: señales de entrada y señales de salida.

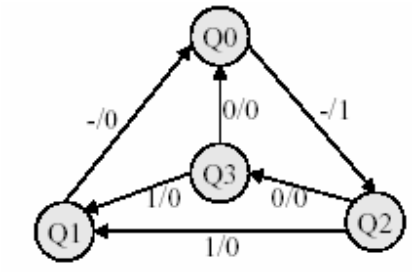
En cada estado hay cero o más reglas que son ejecutables.
Si no hay reglas ejecutables la máquina se dice que está en un estado de terminación.

Si hay una regla ejecutable, la máquina realiza una transición -determinista- hacia el siguiente estado.

Por ejemplo:

Estado Actual	IN	OUT	Siguiente Estado
Q1	-	0	Q0
Q1	0	0	Q3

Una máquina de estado finito también puede representarse gráficamente mediante un diagrama de transición de estados (ejemplo determinista)



Los círculos representan los estados y las flechas las transiciones. Las etiquetas de las flechas son del tipo *c/e*, donde *c* representa una condición de transición (p.ej. un valor de entrada) y *e* el correspondiente efecto (p.ej. un nuevo valor de salida).

MÁQUINAS DE TURING

Un AFD o autómata finito determinista es una máquina de estados que tiene acceso a una secuencia de símbolos de entrada (mediante una cabeza lectora).
Un AFD se encuentra en cada momento en un estado determinado y puede transitar a otro estado. Para ello se realizan los siguientes pasos:

- Se lee la cinta y se avanza la cabeza lectora.
- En función del símbolo leído y del estado actual el autómata transita a otro estado.

Un AFD para el procesamiento cuando no le quedan más símbolos en la entrada.
La parada puede ocurrir en un estado marcado como “final” o uno que no está marcado como final → “salida binaria”.

El *determinismo* del autómata proviene del hecho que la imagen de la función de transición sea S. Es decir, en cada momento (para cada símbolo de entrada y cada estado) solo tiene definido una posible transición.

El nombre de *finito* proviene del hecho que el autómata sólo tiene un conjunto finito de estados distintos para recordar lo procesado (no tiene ningún sistema de almacenamiento de información adicional).

Un AFND o autómata finito no determinístico es aquel que presenta cero, una o más transiciones por el mismo carácter del alfabeto y se clasifican con y sin transiciones, dependiendo de la existencia o no de una o más transiciones sin que el autómata lea el próximo carácter de la cadena que está analizando.
Un autómata finito no determinístico también puede o no tener más de un nodo inicial, por tanto, si hay dos reglas o más que son ejecutables se selecciona de manera no determinista una opción.

Los grafos sencillos que no correspondan a AFD:

- Carecen de transiciones para determinados estados y símbolos del alfabeto.
- Algunas transiciones no están etiquetadas.
- No tienen estado inicial (un AFD puede carecer de estados finales).

Ejemplo

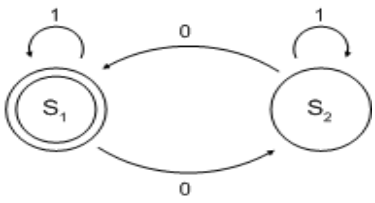
- $\Sigma = \{0,1\}$
- $S = \{S_1, S_2\}$
- $T = \{(S_1,0,\{S_2\});(S_1,1,\{S_1\});(S_2,0,\{S_1\});(S_2,1,\{S_2\})\}$
- $s = S_1$
- $A = \{S_1\}$

Como hemos visto, existen distintas formas de representar un autómata finito a través de su definición formal que resultan más cómodas. Entre estas notaciones, las más usuales son:

- *Tablas de Transiciones*: la tabla de transición para el AF del ejemplo es

	0	**1**
***S*₁**	S_2	S_1
***S*₂**	S_1	S_2

- *Diagramas de Transiciones*: el diagrama de transición para el AF del ejemplo es



Si el entorno ya no se considera finito se obtiene el modelo de máquina de Turing, que es un autómata que se mueve sobre una secuencia lineal de datos. En cada instante la máquina puede leer un solo dato de la secuencia (generalmente un carácter) y realiza ciertas acciones en base a una tabla que tiene en cuenta su "estado" actual (interno) y el último dato leído. Entre las acciones está la posibilidad de escribir nuevos datos de la secuencia; recorrer la secuencia en ambos sentidos y cambiar de "estado" dentro de un conjunto finito de estados posibles.

Existen distintas "variedades" de una máquina de Turing, pero la más simple puede ser descrita diciendo que es cualquier dispositivo que cumple las siguientes condiciones:

- Tiene una cinta sobre la que puede desplazarse a izquierda y derecha de un cabezal de lectura/escritura. La cinta contiene una serie de celdas, y en cada una de ellas puede escribirse un símbolo de un conjunto finito; este conjunto de símbolos se denomina alfabeto de la máquina. En principio todas las celdas que no se hayan escrito antes contienen un carácter especial nulo o vacío (que se representa por 0 o #). La cinta puede contener tantas celdas a derecha e izquierda del cabezal como sean necesarias para el funcionamiento de la máquina.

- El cabezal puede moverse a derecha (R) a izquierda (L) de su posición actual, así como leer el contenido de una celda o escribir en ella cualquier carácter de su alfabeto.

- Existe un registro de estado que almacena el estado de la máquina. El número de estados posibles es finito, y no se exige ningún estado especial con el que sea iniciada la máquina.

- Existe una tabla de acción, que contiene las instrucciones de lo que hará el autómata. Estas instrucciones representan en cierta forma el "programa" de la máquina. La ejecución de cada instrucción de la tabla de acción incluye cuatro pasos:

  - Leer un carácter en la posición actual.
  - Escribir un nuevo símbolo en esta posición. El símbolo a escribir es alguno del alfabeto de la máquina, y depende del carácter leído y del estado actual.
  - Desplazar el cabezal una celda a derecha o izquierda (R/L); en algunos modelos el desplazamiento puede ser nulo (detener H).
  - Decidir cual será el nuevo estado en función del carácter que se acaba de leer y del estado actual. Si la tabla de acción no contiene ninguna correspondencia con el estado actual y el símbolo leído, entonces la máquina detiene su funcionamiento.

En los modelos didácticos computarizados la tabla suele definirse mediante una matriz de cinco columnas que contiene:

Estado/Carácter-leído/Carácter-a-escribir/
Movimiento/Nuevo-estado

En el recuadro se incluye una muestra de una de estas tablas. Representa el comportamiento de una máquina de Turing capaz de sumar 1 a cualquier número unario. El alfabeto solo tiene dos símbolos: Vacío (0) y valor (1). La máquina puede adoptar tres estados diferentes numerados del 0 al 2 (es costumbre señalar el estado inicial con 0). El movimiento H ("Halt") significa no desplazar el cabezal. En este caso la máquina se detiene (o entra en un bucle sin fin).

S	R	W	M	N
0	0	0	R	0
0	1	1	R	1
1	0	1	R	2
1	1	1	R	1
2	0	0	H	2
2	1	0	H	2

S = Estado actual

R = Carácter leído

W = Carácter escrito

M = Dirección del movimiento

N = Nuevo estado

También es posible representar la tabla de acción mediante un grafo. Los diferentes estados internos se representan por círculos. Los cambios de estado con flechas a las que se añade una leyenda. Generalmente se utiliza una flecha para señalar el estado inicial.

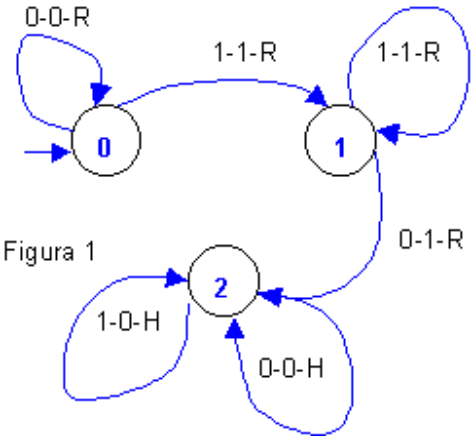


Figura 1

Es imposible predecir el comportamiento de la simple inspección de su tabla de acción, ya que el comportamiento depende también de la entrada recibida.
El hecho que el número de estados posibles y su alfabeto sea finitos, califica a estos autómatas como máquinas de estados finitos FSM ("Finite State Machine").

Es significativo que la cinta puede extenderse indefinidamente a derecha e izquierda. Es también destacable que la máquina da a la cinta tres utilizaciones distintas:

- Como elemento de almacenamiento de los datos de entrada.
- Como elemento de salida.
- Como almacenamiento de información intermedia durante el proceso.

Aunque tanto el alfabeto utilizado como el número de estados son finitos, lo que confiere su potencia a la máquina de Turing es su almacenamiento ilimitado.

Ejemplo: *Busy Beaver* (castor atareado)

Condición		Efecto	
Estado Actual	IN	OUT / MOVE	Siguiente Estado
Q0	0	1/L	Q1
Q0	1	1/R	Q2
Q1	0	1/R	Q0
Q1	1	1/L	-
Q2	0	1/R	Q1
Q2	1	1/L	Q3
Q3	-	-	-

Esta máquina tiene 2 señales de salida: Una sobre-escribe el cuadrado de la cinta y la otra reposiciona la cinta. El estado Q3 es un estado de terminación.

MÁQUINAS DE ESTADO FINITO COMUNICANTES

Resultan de interconectar las salidas y entradas de una máquina de estado. Dichas máquinas ejecutan un algoritmo en 2 pasos:
• Primero, se inspecciona el valor de entrada y se selecciona una regla ejecutable.
• Segundo, la máquina cambia de estado y actualiza las salidas en concordancia con las reglas de transición.

Dicho algoritmo se ejecuta para siempre. Un ejemplo serían las transiciones Q0 Q2 Q1 para siempre de la tabla inicial.
Se pueden construir sistemas elaborados utilizando la interacción entre máquinas de estado finito comunicantes, conectando las salidas de una con las entradas de otra.

ACOPLAMIENTO ASÍNCRONO

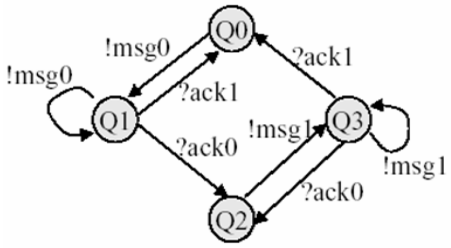
Las máquinas están acopladas mediante colas de mensajes FIFO.
Si una cola de entrada está vacía la transición será no ejecutable.
Si una cola de salida está llena la transición que provocaría un mensaje hacia dicha cola es no ejecutable.

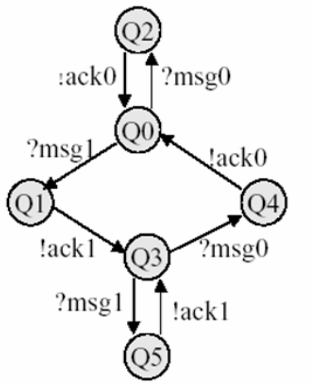
Se restringe el modelo a un evento de sincronismo por regla de transición: una regla especifica una entrada o una salida (no ambas).

Ejemplo: Protocolo de bit alternado con *timeouts*.

Estado	IN	OUT	Siguiente
Q0	-	msg0	Q1
Q1	ack1	-	Q0
Q1	ack0	-	Q2
Q2	-	msg1	Q3
Q3	ack0	-	Q2
Q3	ack1	-	Q0
Q1	-	msg0	-
Q3	-	msg1	-

Estado	IN	OUT	Siguiente
Q0	msg1	-	Q1
Q0	msg0	-	Q2
Q1	-	ack1	Q3
Q2	-	ack0	Q0
Q3	msg0	-	Q4
Q3	msg1	-	Q5
Q4	-	ack0	Q0
Q5	-	ack1	Q3





ACOPLAMIENTO SÍNCRONO

Las condiciones de transición son “selecciones” que la máquina puede hacer para comunicarse.
La máquina, de nuevo, puede seleccionar o bien una salida o bien una entrada.
Para hacer un movimiento una señal debe ser seleccionada por 2 máquinas simultáneamente, como entrada por una y salida por la otra.

Ejemplo: Semáforos.

Estado	IN	OUT	Siguiente
Q0	P	-	Q1
Q1	-	V	Q0

Usuario

Estado	IN	OUT	Siguiente
Q0	-	P	Q1
Q1	V	-	Q0

Servidor

Nótese que si se crean 1 servidor y 2 usuarios, éstos nunca podrán de manera simultánea estar en Q1

8.3. DESCRIPCIÓN FORMAL

Un autómata finito (AF) puede ser descrito como una 5-tupla (S, Σ, T, s, A) donde:

- Σ es un alfabeto;
- S un conjunto de estados;
- T es la función de transición: T: $S \times (\Sigma \cup \{\in\}) \rightarrow P(S)$;
- $s \in S$ es el estado inicial;
- $A \subseteq S$ es un conjunto de estados de aceptación o finales.

Una cola de mensajes es la terna (S, N, C) donde:
• S es un conjunto finito, llamado vocabulario.
• N es un entero que indica el número de posiciones de la cola.
• C es el contenido de la cola, un conjunto ordenado de elementos de S.
Los elementos de S y C se llaman mensajes.

Sea M el conjunto de todas las colas de mensajes, sea el superíndice 1<=m<=M el identificador de una cola y el subíndice 1<=n<=N la posición n-ésima de una cola.
C_n^m es el mensaje n de la cola m.
El vocabulario del sistema V será la unión de todos los vocabularios más el elemento nulo ε.
Para un conjunto de M colas (de 1 a M) se define V como:

$$V = \bigcup_{m=1}^{M} S^m \cup \varepsilon$$

Una máquina de estado finito comunicante es una tetra (Q, q0, M, T) donde:
• Q es un conjunto finito no vacío de estados,
• q0 es un elemento de Q, el estado inicial,
• M es un conjunto de colas de mensajes, y
• T es una relación de transición de estados.

T(q,a): *q* es el estado actual, *a* es una acción.
Se permiten 3 tipos de acciones:
-entradas,
-salidas
-acción nula 0

La relación de transición T define un conjunto de cero o más posibles estados sucesores del estado q del conjunto Q. Si no se modela el no-determinismo, habrá un estado sucesor. Si T(q,a) no está explícitamente definido se supone T(q,a)=∅. T(q,0) indica una transición espontánea.

8.4. EJECUCIÓN DE MÁQUINAS

• Poner todas las máquinas al estado inicial con las colas de mensajes vacías.
i, 1<=i<=P → $q^i = q_0^i$ y i, 1<=i<=M → $C^i = ø$
• Seleccionar una máquina arbitraria *i* y una regla de transición arbitraria Ti tal que Ti(q,a) ≠ ø, con a ejecutable, y ejecutarla.
• Si no quedan reglas ejecutables, el algoritmo termina.

La acción *a* puede ser una entrada, una salida o 0.
Sea 1<=*d(a)*<=M la cola destino de una acción *a*, y sea *m(a)* el mensaje enviado o recibido, *m(a)* ε $S^{d(a)}$. Sea N^i el número de posiciones en la cola de mensajes *i*. Para ver si *a* es ejecutable basta con cumplir una regla:
• *a* = ε
• *a* es entrada y *m(a)* = $C_1^{d(a)}$
• *a* es salida y $C^{d(a)} < N^{d(a)}$

8.5. MINIMIZACIÓN DE MÁQUINAS

Considérese la siguiente máquina:

Condición		Efecto	
Estado	IN	OUT	Siguiente
Q0	msg1	-	Q1
Q0	msg0	-	Q2
Q1	-	ack1	Q0
Q2	-	ack0	Q0

Esta máquina es equivalente a la del receptor del protocolo de bit alternado porque PUEDE generar la misma secuencia de símbolos de salida si se le ofrece la misma secuencia de símbolos de entrada.

Dos estados dentro de una máquina son equivalentes si la máquina puede iniciarse en cualquiera de estos 2 estados y puede generar el mismo conjunto de símbolos de salida para cualquier secuencia test de símbolos de entrada.

Para la minimización de máquinas se estudiará Q (conjunto de estados) y T (conjunto de transiciones), considerando M (colas de mensajes) como parte del entorno, no de la máquina.
Este planteamiento genera restricciones mayores al comportamiento de la máquina (impone a los estados restricciones de comportamiento mayores), generando además máquinas equivalentes con menos estados.

ALGORITMO DE MINIMIZACIÓN DE *FSM*

• Definir un array *E* de QxQ valores booleanos.
Inicialmente poner a *true* todos los elementos *E[i,j]* que cumplan:

. $\forall a, T(i,a) \neq \emptyset \leftrightarrow T(j,a) \neq \emptyset$

• De los elementos *E[i,j] true*, cambiar su valor a:
$\forall a, E[T(i,a),E(j,a)]$ Es decir, 2 estados no son equivalentes si sus sucesores no lo son.
• Repetir el punto anterior hasta que no se pueda incrementar el número de *false*.

Ejemplo: Algoritmo recepción bit alternado

	Q0	Q1	Q2	Q3	Q4	Q5
Q0	1	0	0	1	0	0
Q1	0	1	0	0	0	1
Q2	0	0	1	0	1	0
Q3	1	0	0	1	0	0
Q4	0	0	1	0	1	0
Q5	0	1	0	0	0	1

8.6. EL PROBLEMA DEL TEST DE CONFORMIDAD

A través de la equivalencia entre dos estados se puede determinar la equivalencia de dos máquinas: Hay que determinar que cada estado de una máquina tiene un estado equivalente en la otra, y viceversa.
Esto es muy importante cuando se aplica a 2 máquinas, siendo una un modelo y la otra una implementación.
Vistas como “cajas negras”, ambas máquinas deben responder igual a señales de entrada.

8.7. COMBINACIÓN DE MÁQUINAS

Se trata de obtener una tetra (Q, q0, M, T) para la máquina combinada de (Q1, q0 1, M1, T1) y (Q2, q0 2, M2, T2).
• Definir el producto de estados, Q = Q1 x Q2, con q0 1q0 2 el estado inicial de la nueva máquina.
• El conjunto de colas de mensajes M será M1¸ M2. El vocabulario V será la unión de vocabularios, y *a* la unión de acciones
q^1q^2 y a, →?$T(q^1q^2 , a) = T(q^1 , a)¸T(q^2 , a)$

8.8. MÁQUINAS DE ESTADO FINITO EXTENDIDAS

Mediante el modelo de máquina de estado finito se han obtenido máquinas que intercambian objetos abstractos utilizando el acceso a colas como único mecanismo de sincronismo.
Las máquinas de estado finito extendidas introducen 3 modificaciones: Uso de variables, colas de enteros y operadores aritmético-lógicos sobre variables.

Una variable puede definirse de manera parecida a una cola, con la diferencia que una variable tiene un único valor, dentro de un rango finito, en un instante dado. El valor de una variable será el último valor introducido.
Las variables disponen de nombres simbólicos y contienen, en este caso, enteros (en lugar de objetos abstractos).

La segunda modificación es el uso de colas para transferir valores enteros (en lugar de objetos abstractos).
La tercera modificación es la introducción de operadores aritméticos y lógicos para manipular el contenido de las variables.

Una variable puede simularse con una máquina de estado finito, siempre que su rango de valores sea finito.
El siguiente ejemplo es una máquina de estado finito que simula una variable con 3 posibles valores.

Estado	IN	OUT	Siguiente
Q0	S0	-	-
Q0	S1	-	Q1
Q0	S2	-	Q2
Q0	RV	-	R0
R0	-	0	Q0
Q1	S0	-	Q0
Q1	S1	-	-
Q1	S2	-	Q2
Q1	RV	-	R1
R1	-	1	Q1
Q2	S0	-	Q0
Q2	S1	-	Q1
Q2	S2	-	-
Q2	RV	-	R2
R2	-	2	Q2

La extensión de la máquina de estado finito mediante variables no es más que una conveniencia para el modelado (el precio es un incremento importante del número de estados).
Ahora las transiciones incluyen en las condiciones expresiones booleanas sobre el valor de las variables y en los efectos asignaciones a variables.

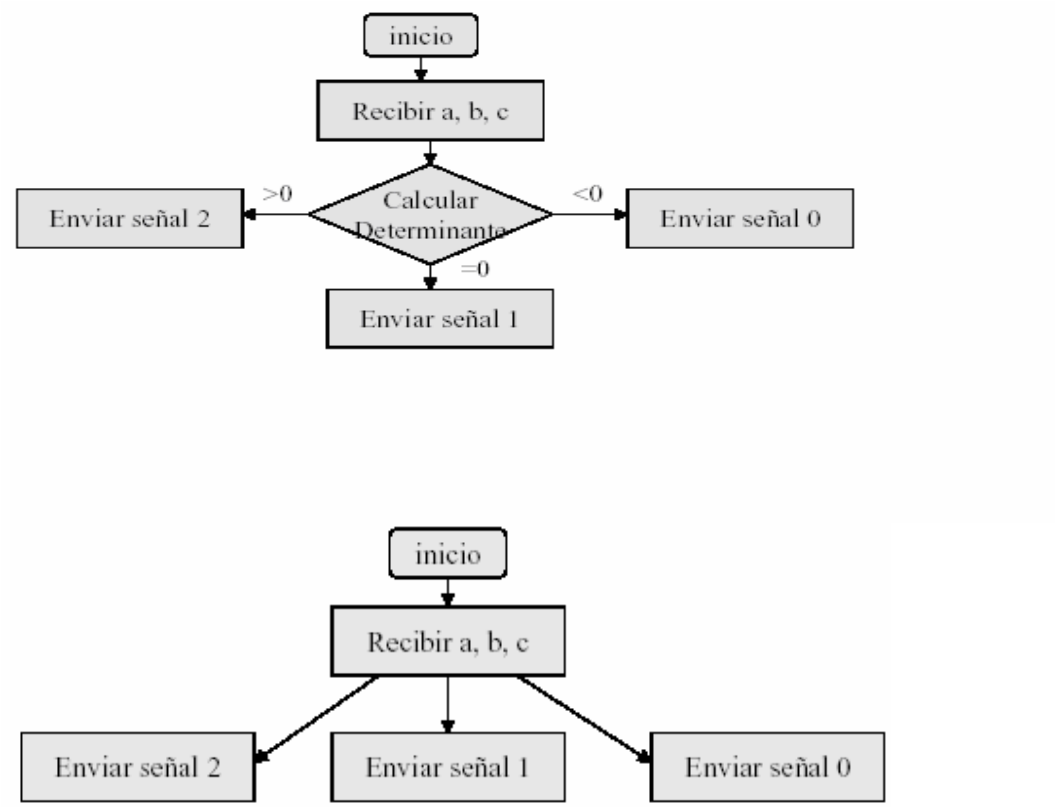
Se puede definir una máquina de estado finito extendida como la quíntupla (Q, q0, M, A, T) donde:
• A es un conjunto de nombres de variables, y
• Q, q0, M, T como antes.
Se han añadido 2 tipos de acciones extra (a entradas, salidas y acción nula 0): condiciones booleanas sobre elementos de A y asignaciones a elementos de A.

E/S EXTENDIDA
Se puede definir las acciones de E/S como un conjunto de valores finito y ordenado. Los valores serán expresiones sobre variables de A o simples constantes.

Ejemplo: obtención máximo común divisor (Euclides)

Estado	Acción	Siguiente
Q0	In?x,y	Q1
Q1	x>y	Q2
Q1	x<y	Q3
Q1	x=y	Q4
Q2	x:=x-y	Q1
Q3	y:=y-x	Q1
Q4	Out!x	Q5
Q5	-	-

8.9. GENERALIZACIÓN DE MÁQUINAS



La generalización de máquinas consiste en eliminar el comportamiento interno que no interesa.
La máquina generalizada puede hacer lo mismo o más que la original.
La máquina generalizada es más general sólo en la manera de generar salida.

La utilidad de la generalización de máquinas es el análisis de problemas complejos.
Supóngase un protocolo formado por los módulos A y B, y que se quiere validar el módulo A. Se puede simplificar el módulo B combinando, reduciendo, generalizando y minimizando máquinas. El nuevo módulo B es capaz de comportarse como el módulo B original, pero puede hacer más.

Si se puede demostrar que el módulo A es correcto en presencia del nuevo módulo B (más fácil) también será correcto en presencia del módulo B original, ya que su comportamiento es un subconjunto del nuevo comportamiento.
En la validación del módulo A de esta manera se han obtenido 2 beneficios: se ha realizado un test más duro (se validan condiciones más generales que en el protocolo original) y ha sido más fácil de realizar.

8.10. MODELOS RESTRINGIDOS

Para el análisis de protocolos se usan distintas variaciones del modelo de máquina de estado finito, tanto extensiones (mayor poder de modelado) como restricciones (mayor poder analítico) del modelo.
Se verá un caso modelo restringido: Redes de Petri.

REDES DE PETRI

Una red de Petri es una herramienta para el modelado de sistemas de información que son considerados no determinísticos, concurrentes, paralelos, asíncronos, distribuidos y/o estocásticos. Las redes de Petri han estado en desarrollo desde principios de los 60's, cuando Carl. A. Petri definió el lenguaje de dichas redes. En este trabajo fue presentado por primera vez una teoría general para sistemas discretos paralelos. El lenguaje es una generalización de la teoría de autómatas de tal forma, que el concepto de eventos que suceden de manera concurrente pueda ser expresado.

Una red de Petri es un grafo dirigido bipartito, con un estado inicial, llamado marcación inicial. Los componentes principales de una red de Petri son los sitios, transiciones y aristas dirigidas (*places, transitions and edges*). Gráficamente, los sitios (también conocidos como estados) son dibujados como círculos y las transiciones como barras o rectángulos. Las aristas del grafo son conocidas como *arcos*. Estos tienen un peso específico, el cual es indicado por un número entero positivo, y van de sitio a transición y viceversa. Por simplicidad, el peso de los arcos no se indica cuando éste es igual a 1. Un arco que esté etiquetado con *k* puede ser interpretado como *k* arcos paralelos.

El estado del sistema que la red esté modelando está representado con la asignación de enteros no-negativos a los sitios. Esta asignación es conocida como una *marcación*, la cual es representada gráficamente mediante unos pequeños círculos negros dentro de un sitio *p*, llamados *tokens* . Si el número de tokens es demasiado grande, los *k* tokens son representados con un número no-negativo dentro del correspondiente sitio.

Típicamente, los estados representan algún tipo de condición en el sistema, y una transición representa un evento. Un sitio de entrada (salida) a una transición representan las pre- (post-) condiciones.
Los tokens pueden tener muchas interpretaciones, como que una condición es verdadera. En otros casos, *k* tokens pueden representar *k* recursos, por ejemplo, el número de *clicks del mouse* realizados. Debido a que las redes de Petri pueden modelar muchos tipos de sistemas, lo que los sitios, transiciones y tokens representen varía enormemente.

Definición : Una red de Petri consta de cinco elementos, PN = (P,T,F,W,M_0) donde:

1. P=p_1,p_2,...,p_m es un conjunto finito de sitios.

2. T = t_1,t_2,...,t_n es un conjunto finito de transiciones.

3. $F \subseteq (PxT) \bigcup (TxP)$ es un conjunto de arcos.

4. W : F→ {1,2,3,...} es una función de peso.

5. M_0 : P→ {0,1,2,3,...} es la marcación inicial.

6. $P \bigcap T = \phi \quad P \bigcup T \neq \phi$

Es conveniente utilizar la siguiente notación:
Para una transición t, los sitios de entrada y los sitios de salida serán denotados como:

t={p|(p,t)∈ F} y t={p|(t,p)∈ F}

De manera formal, una marcación M es definida como M : P→{0,1,2,....}

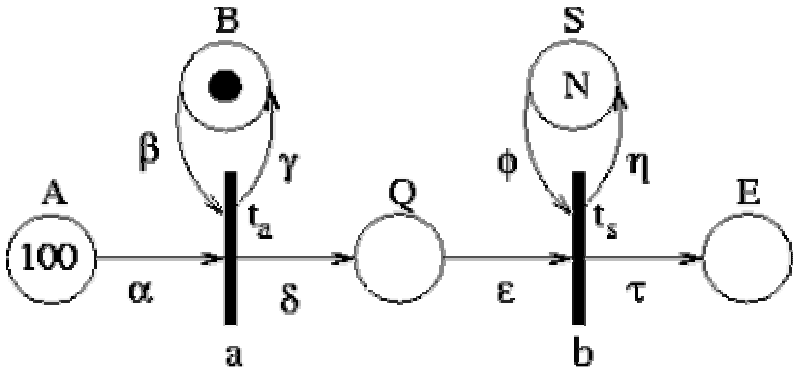
También es conveniente, en algunos casos, el denotar una marcación M de m sitios como un vector-m donde el i-ésimo componente es denotado como M(p_i), por ejemplo

M = <M(p_1)...,M(p_m)>

Una transición t está habilitada con una marcación M si cada sitio de entrada p está marcado con al menos W(p,t) tokens. Una transición puede o no ser disparada al habilitársele. Cuando más de una transición es habilitada, alguna de ellas es seleccionada de manera no-determinística dependiendo del modelo empleado. Conforme las transiciones son disparadas, el número total de tokens distribuidos a lo largo de la red puede variar, esto es, la conservación de los tokens no siempre sucede.

Ejemplo.
Se tiene una sola línea para atender a 100 clientes. Los tiempos de llegada de los clientes serán valores sucesivos de la variable aleatoria t_a, los tiempos de servicio están dados por la variable aleatoria t_s, y *N* es el número de servidores. Este modelo en su estado inicial tiene la cola vacía y todos los servidores en estado de espera. La red de Petri para este escenario se muestra en la figura.

Ejemplo de una red de Petri



Los estados están etiquetados con letras mayúsculas y las transiciones con minúsculas. Las etiquetas de los sitios también serán usados como las variables de cuyos valores son los tokens.

Las aristas tienen etiquetas que podrían representar las funciones de transición, las cuales especifican el número de tokens eliminados o agregados cuando una transición es activada.
El estado A inicialmente contiene la llegada de 100 clientes; el sitio B evita que los clientes entren más de una vez; el sitio Q es la fila que realizan los clientes cuando tienen que esperar a que se les atienda. El estado S es donde los servidores ociosos esperan la oportunidad para trabajar, y el sitio E cuenta el número de clientes que abandonan el sistema. El estado inicial implica que los sitios tengan los siguientes valores:

- A = 100
- B = 1
- Q = 0
- S = N
- E = 0

La transición a sirve para modelar a los clientes que entran al sistema y la transición b modela a los clientes cuando están siendo atendidos.

CONCLUSIONES

- El diseño de protocolos es tan viejo como la comunicación, y estos cada vez ofrecen mayor fiabilidad y funcionalidad, pero también mayor complejidad y tamaño.

- La definición de un protocolo para ser completa, debe incluir los cinco elementos básicos, y muchos fallos en protocolos, son suposiciones no definidas.

- El control de errores es un módulo funcional dentro de la jerarquía de protocolos, y en caso de ser implementado, debe ser transparente al resto del protocolo, y cualquier esquema utilizado debe adecuarse a las características de los errores en el canal de transmisión, velocidad de transmisión requerida y error residual deseado.

- Un esquema de control de flujo puede usarse para coordinar la velocidad de transmisión de mensajes entre procesos de un sistema distribuido, para evitar cuellos de botella y para la recuperación del sistema en caso de errores de transmisión.

- Un modelo por capas jerárquicas realizan funciones independientes, y cada capa añade funcionalidad.
  Hay que obtener un modelo de validación si se quiere hallar todos los errores (bloqueos, recepciones inesperadas, segmentos de código que no se ejecutan, ...)

- Hay tres criterios para evaluar lo adecuado de un modelo formal: poder analítico, poder de modelado y claridad descriptiva.
  El objetivo fundamental del modelado es el poder analítico (modelo más fácil de analizar el sistema).

BIBLIOGRAFIA

- www.linux-magazine.es/issue/01/Herramientasred.pdf
- http://neo.lcc.uma.es/evirtual/ cdd/tutorial/transporte/conges.html
- http://es.wikipedia.org/wiki/Máquinas_de_estado_finito
- http://docencia.udea.edu.co/SistemasDiscretos/ contenido/capitulo_12.html
- www-etsi2.ugr.es/alumnos/mlii/Maquina%20de%20Turing.htm
- www.fismat.umich.mx/~crivera/tesis/node25.html
- http://es.wikipedia.org/wiki/Red_de_Petri
- http://jungla.dit.upm.es/~trdt/apuntes/t7.html
- http://es.wikipedia.org/wiki/ARQ
- www.elprisma.com/apuntes/ apuntes.asp?page=12&categoria=602
- www.inf-cr.uclm.es/www/sreyes/Redes/Teoria/T3.pdf
- www.inf.utfsm.cl/~rmonge/uv/com/capitulo3x3.pdf
- www.dte.us.es/ing_inf/arc1/tema4-0506.pdf
- http://campus.uab.es/~2059673/doc_capa3.html
- www.it.uc3m.es/~jmoreno/telematica/ servidor/apuntes/tema6/tema06.htm
- http://jungla.dit.upm.es/~trdt/apuntes.html
- www.it.uc3m.es/~jmoreno/telematica/ servidor/apuntes/tema7/tema07.htm
- www.profesores.frc.utn.edu.ar/ sistemas/ingcura/Archivos_Redes/OSI.htm
- http://es.wikipedia.org/wiki/Capa_de_**enlace**_de_datos
- www.geocities.com/txmetsb/**enlace**-de-datos.html
- html.rincondelvago.com/ protocolos-de-comunicacion_1.html
- http://jungla.dit.upm.es/~trdt/examenes/feb97/feb97p2.html

- www.isa.cie.uva.es/proyectos/codec/teoria4.html
- http://lorca.umh.es/isa/es/asignaturas/ sii/Tema3%20Redes%20SII%2005-06.pdf
- http://telecom.fi-b.unam.mx/manuales/ commdigitales/practica10.doc
- "Design and Validation of Computer Protocolos", Gerard J. Holzmann
- Murray Hill, New Jersey 07974
- Página web de la asignatura de Ingeniería de Protocolos y Servicios (http:// labit501.upct.es/ips).
- Norma ISO 8879 – SGML (Standard Generalized Markup Language)