Tema 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.

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

Tema 7: DISEÑO DE PROTOCOLOS

Ingeniería de Protocolos y Servicios

7.1. Introducción

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.

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

Tema 7: DISEÑO DE PROTOCOLOS

Ingeniería de Protocolos y Servicios

7.1. Introducción

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.

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

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.

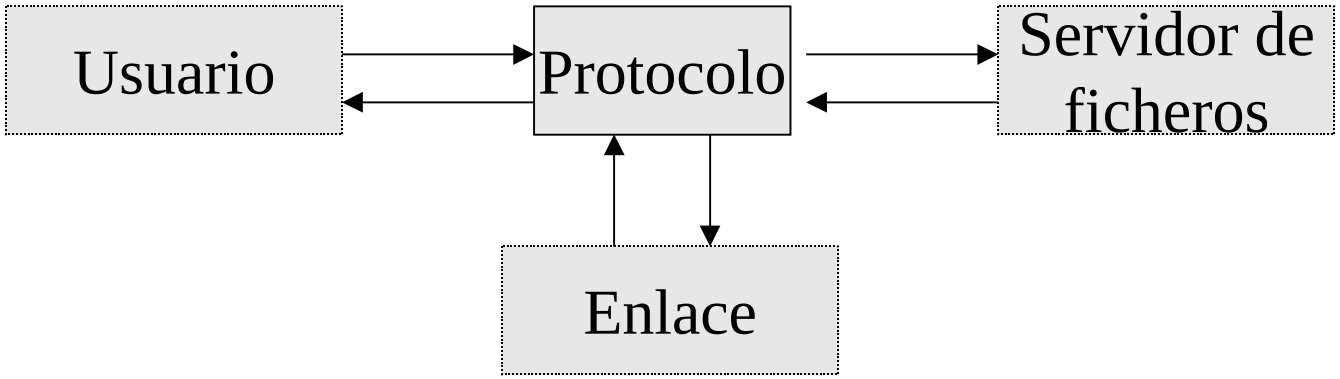
7.3. Suposiciones sobre el canal.

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).

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

7.4. Vocabulario del protocolo.

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



7.4. Vocabulario del protocolo.

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

7.4. Vocabulario del protocolo.

- 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).

francesc.burrull@upct.es

7.4. Vocabulario del protocolo.

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.

7.4. Vocabulario del protocolo.

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

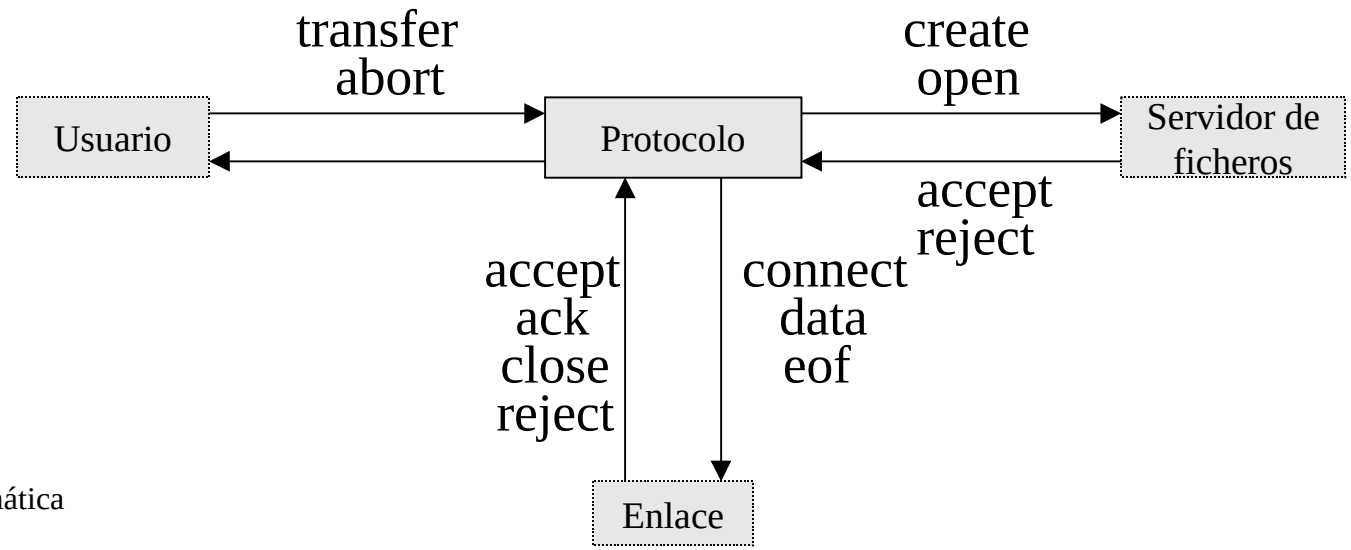
Para la implementación del control de flujo:

- ack

Tema 7: DISEÑO DE PROTOCOLOS

Ingeniería de Protocolos y Servicios

7.4. Vocabulario del protocolo.



DTIC
Área Ingeniería Telemática

francesc.burrull@upct.es

7.5. Formato de los mensajes.

Como mínimo:

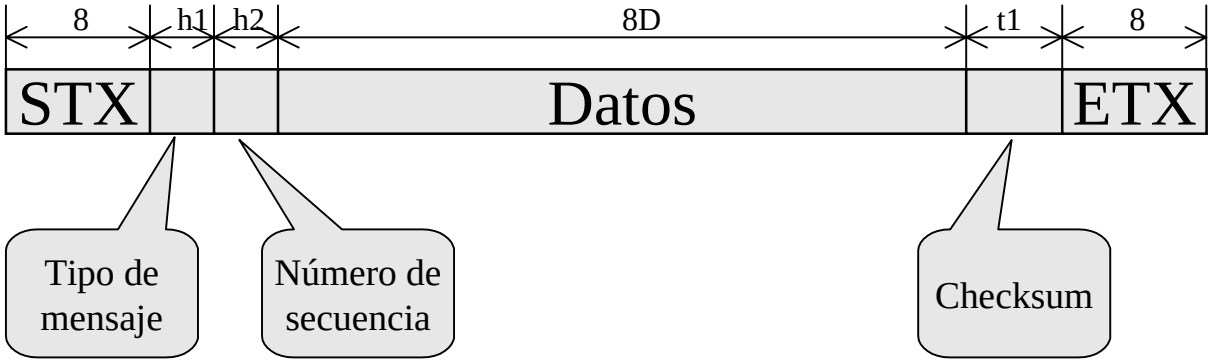
- Un campo de tipo de mensaje y otro opcional de datos.
- Número de secuencia para el control de flujo.
- Checksum para la detección de errores.
- Canal orientado a byte: El mensaje será una secuencia de bytes. ¿Tamaño secuencia? Lo mínimo posible.

7.5. Formato de los mensajes.

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*).

7.5. Formato de los mensajes.



7.5. Formato de los mensajes.

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.

7.5. Formato de los mensajes.

Campo "datos":

El tamaño del mensaje mínimo 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 y p_d=prob. error mensaje, p_a=prob. error ack.

7.5. Formato de los mensajes.

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 = $p_i = (1-p_r)p_r^{i-1}$

7.5. Formato de los mensajes.

Así,

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

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

francesc.burrull@upct.es

7.5. Formato de los mensajes.

Para hallar la máxima eficiencia E se puede derivar respecto d.

Substituyendo los valores conocidos:

a = STX + h1 + h2 + t1 + ETX = 8 + 4 + 2 + 16 + 8 = 38

t = a = 38

Y mediante aproximación de primer orden:

$p_d = (d + 38)/125 \cdot 10^3$ y $p_a = 38/ 125 \cdot 10^3 = 3{,}04 \cdot 10^{-4}$

7.5. Formato de los mensajes.

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

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

francesc.burrull@upct.es

7.5. Formato de los mensajes.

E presenta un máximo para d=3004 bits, y la eficiencia máxima para esta d es del 95,13%.

Con estos valores se tiene:

$p_d = (3004+38)/(125 \cdot 10^3) = 243{,}36 \cdot 10^{-4}$

7.5. Formato de los mensajes.

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.

7.5. Formato de los mensajes.

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.

7.5. Formato de los mensajes.

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, $P(error) < 10^{-8}$

7.5. Formato de los mensajes.

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.

7.6. Reglas de procedimiento.

ABSTRACCIÓN

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

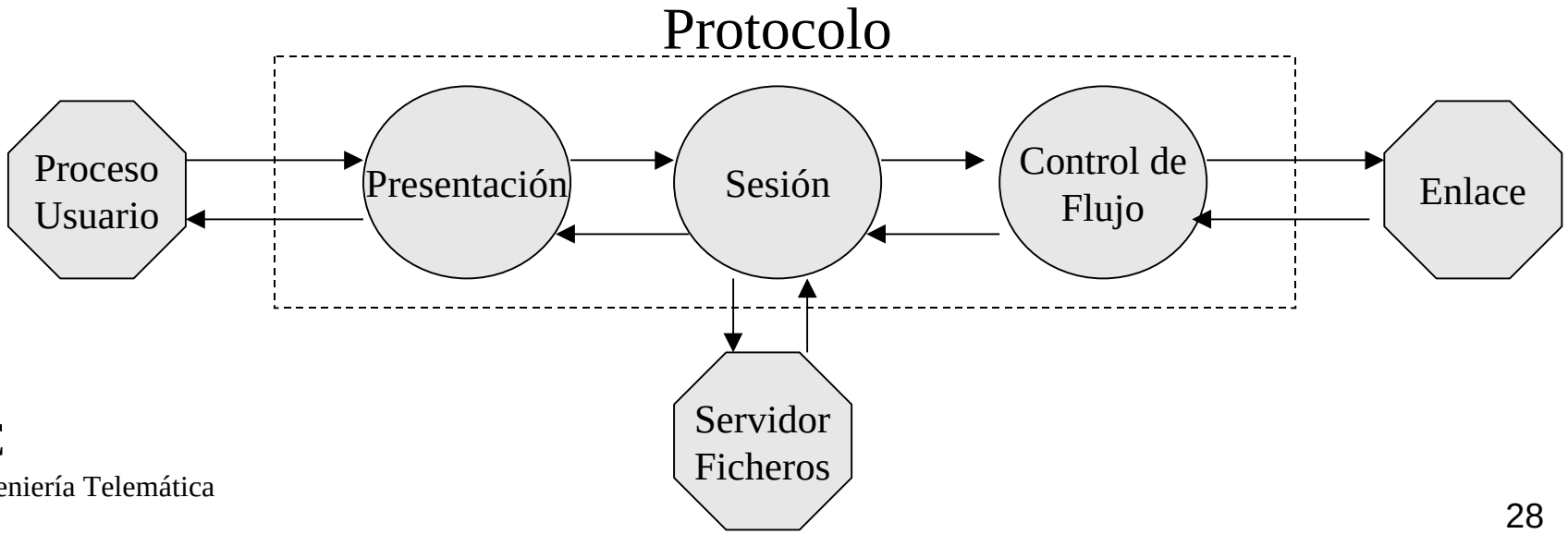
CAPAS

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

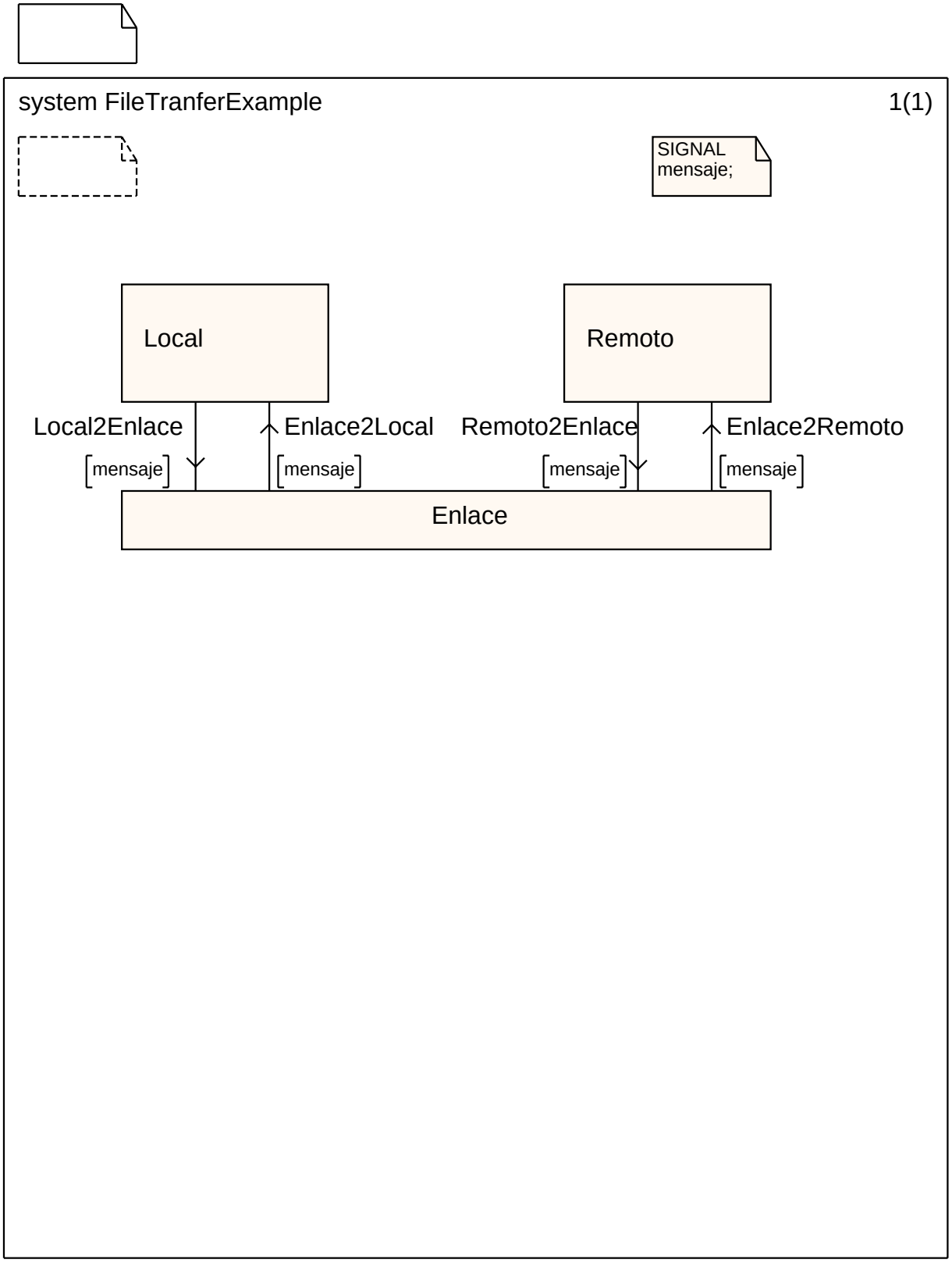
Tema 7: DISEÑO DE PROTOCOLOS

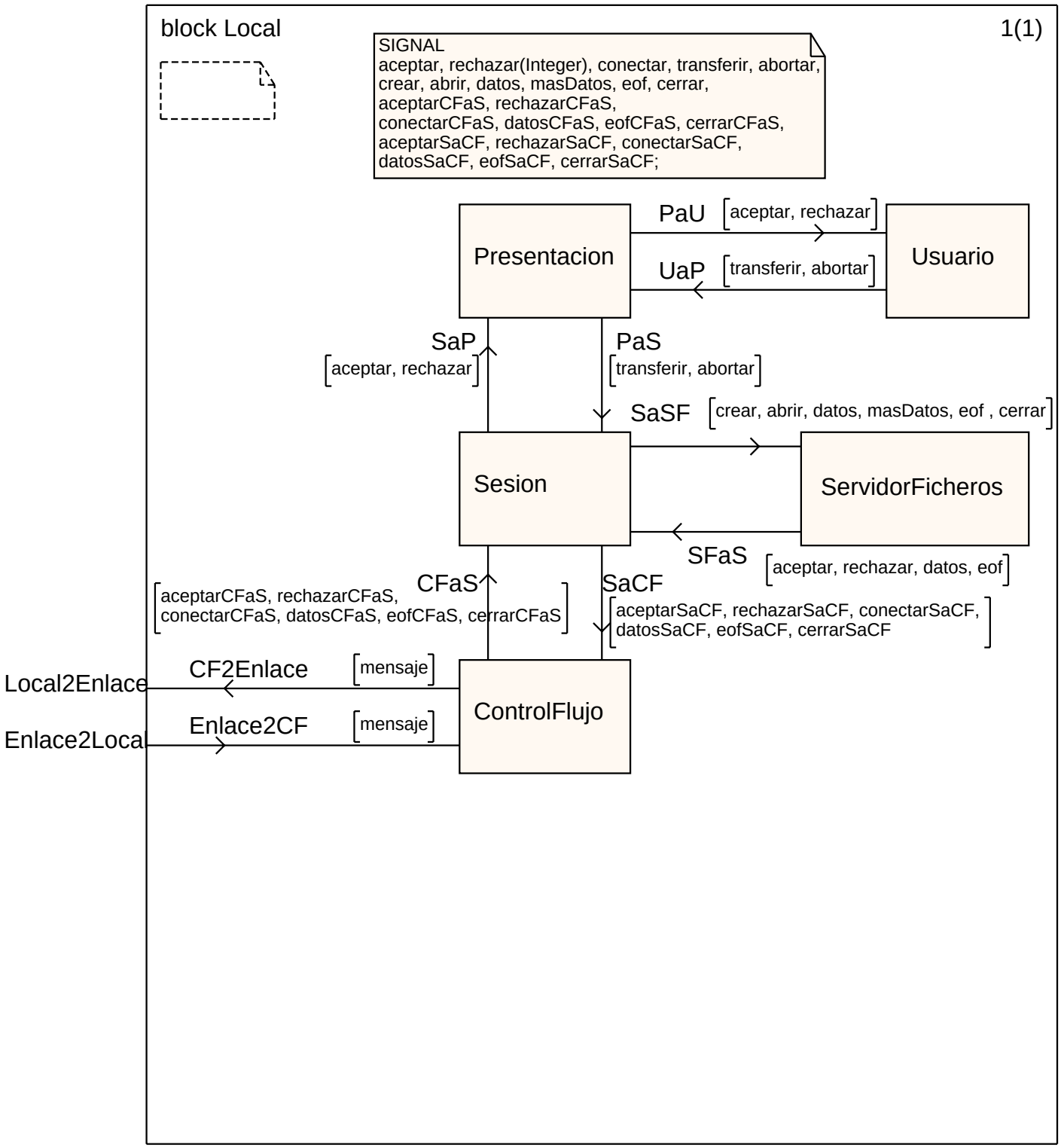
Ingeniería de Protocolos y Servicios

7.6. Reglas de procedimiento.
Jerarquía



DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es


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
PaU
[aceptar, rechazar]
UaP
[transferir, abortar]
Usuario
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]
Enlace2Local
Enlace2CF
[mensaje]
ControlFlujo

7.6. Reglas de procedimiento.

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

7.6. Reglas de procedimiento.

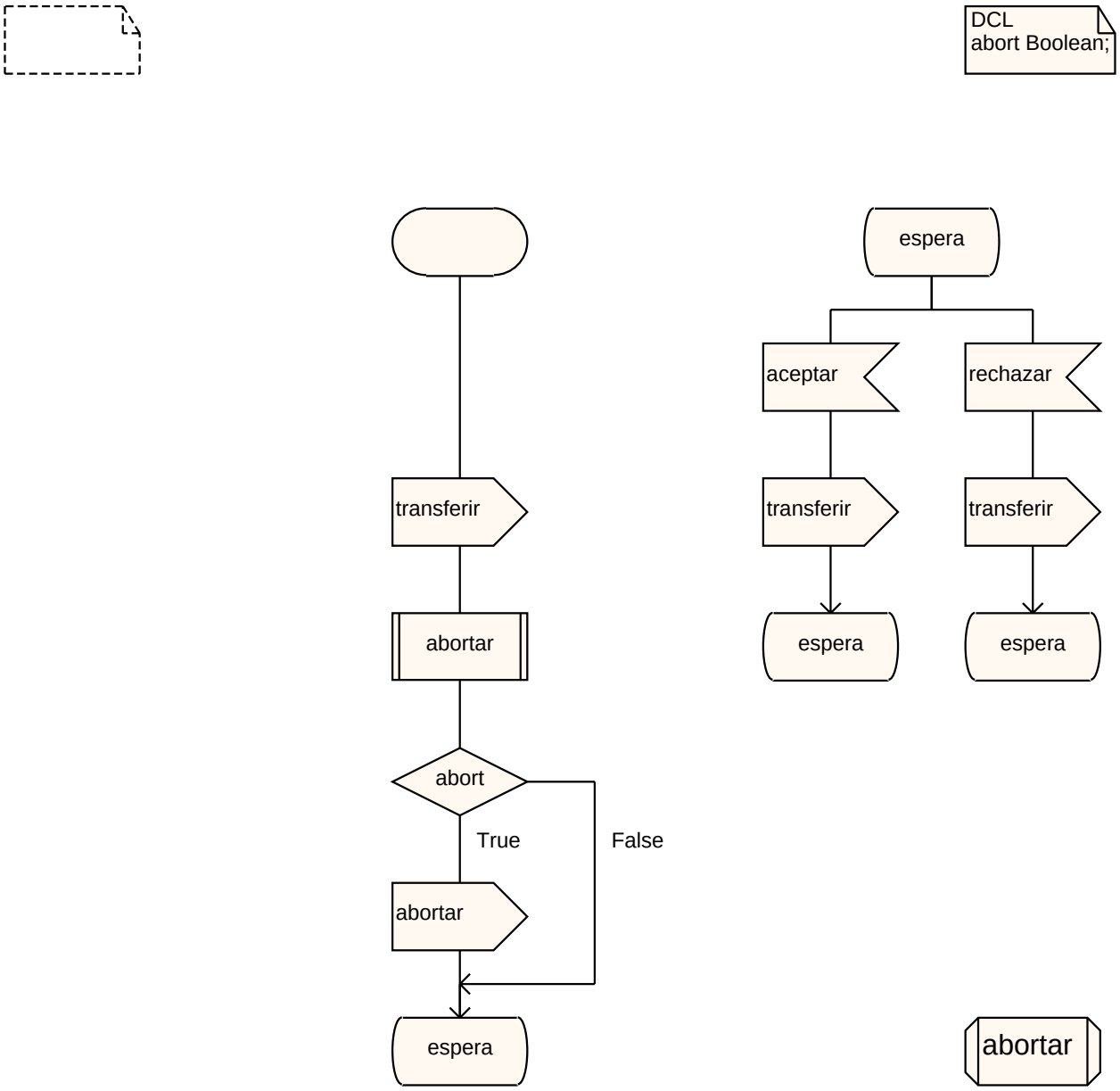
Proceso Usuario.

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



7.6. Reglas de procedimiento.

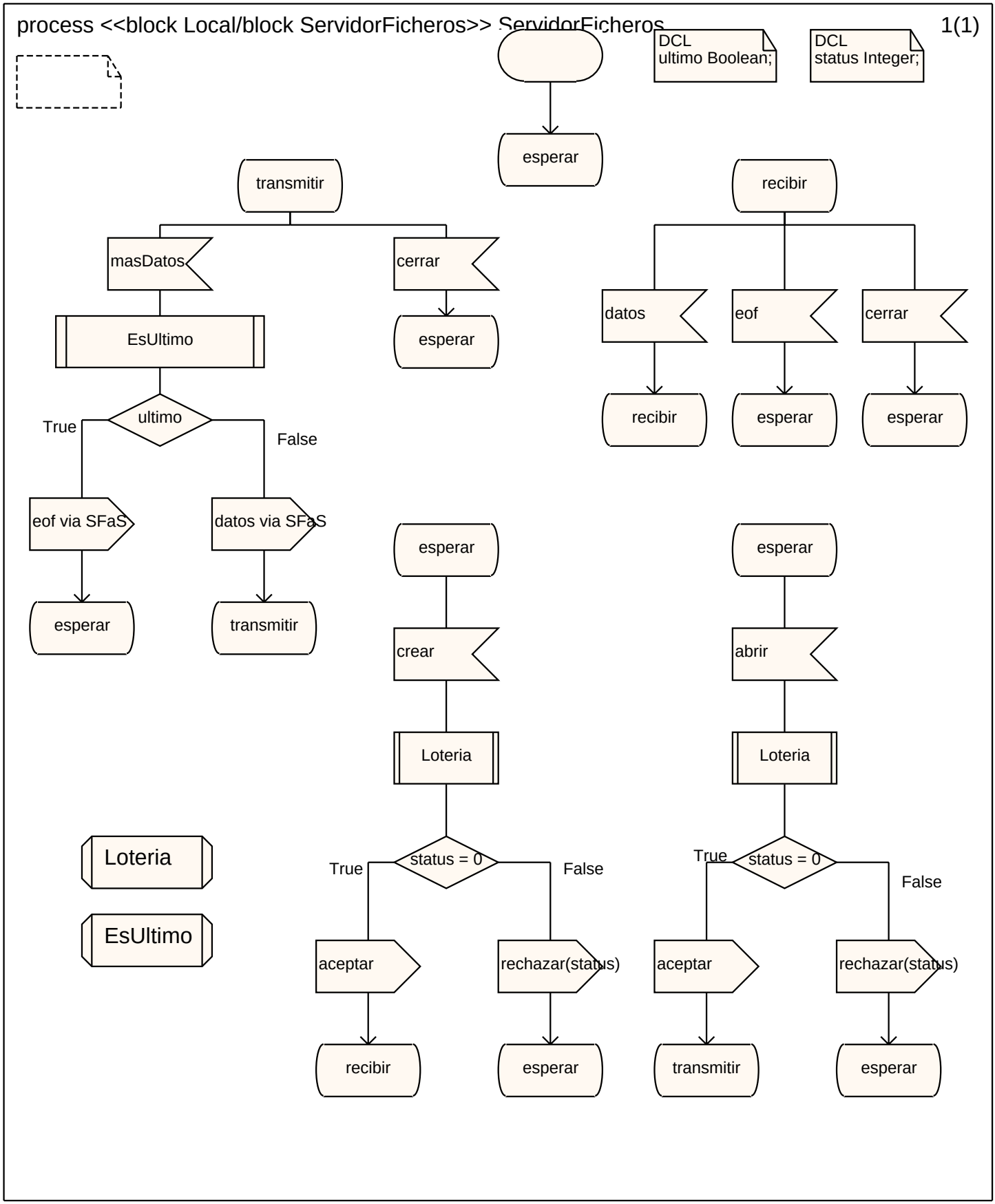
Proceso Servidor de Ficheros

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).

7.6. Reglas de procedimiento.

Proceso Servidor de Ficheros

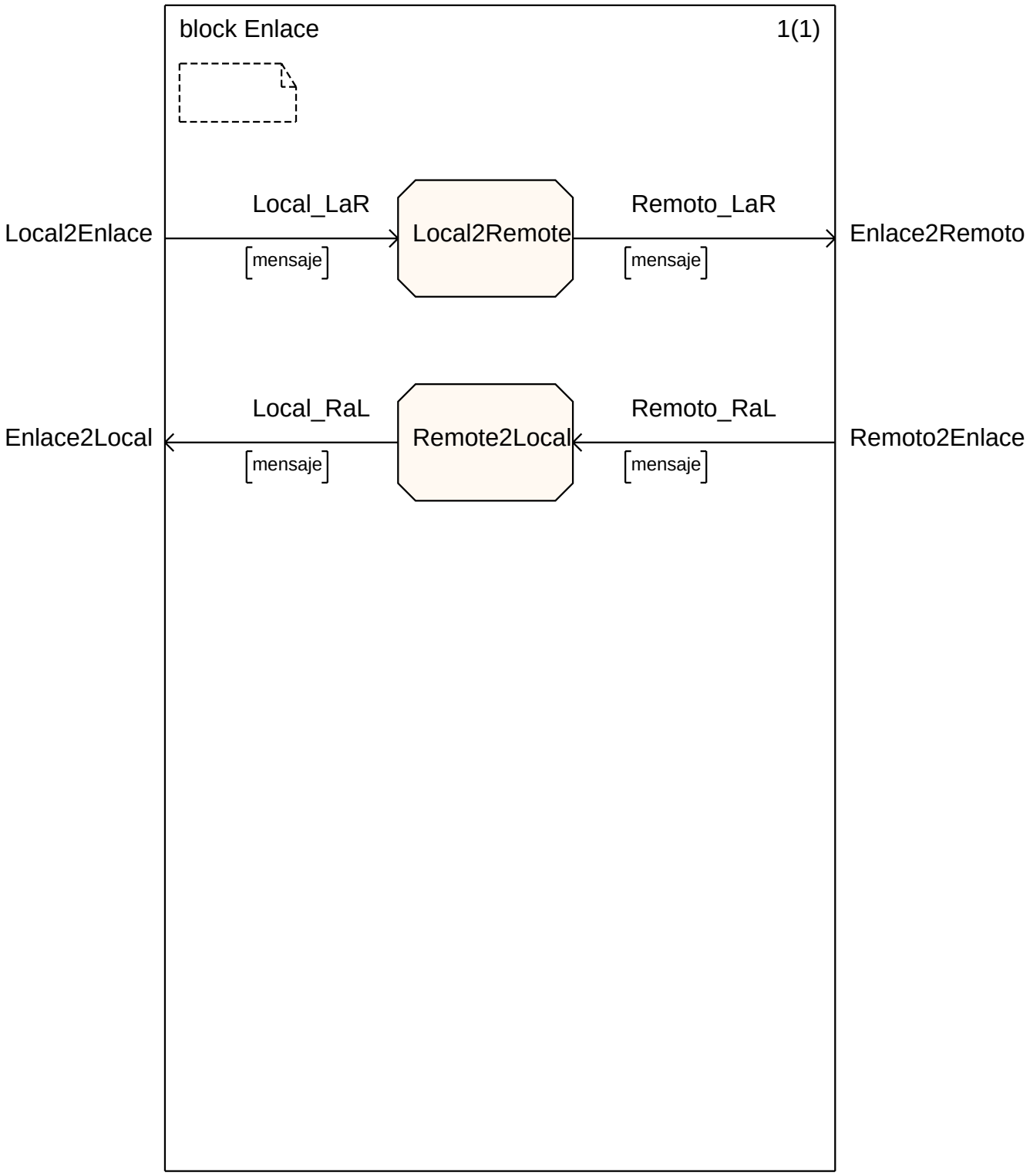
Una petición de leer un fichero empieza con el mensaje *open*. El servidor de ficheros responderá con *reject* o *accept*. En el segundo caso se transferirán cero o más mensajes *data*, volviendo al estado inicial al recibir un *eof* o un *close* (transmisión abortada).


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

7.6. Reglas de procedimiento.

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.


block Enlace
1(1)
Local_LaR
Local2Enlace
mensaje
Local2Remote
Remoto_LaR
Enlace2Remoto
mensaje
Local_RaL
Enlace2Local
mensaje
Remote2Local
Remoto_RaL
Remoto2Enlace
mensaje

7.6. Reglas de procedimiento.

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.

7.6. Reglas de procedimiento.

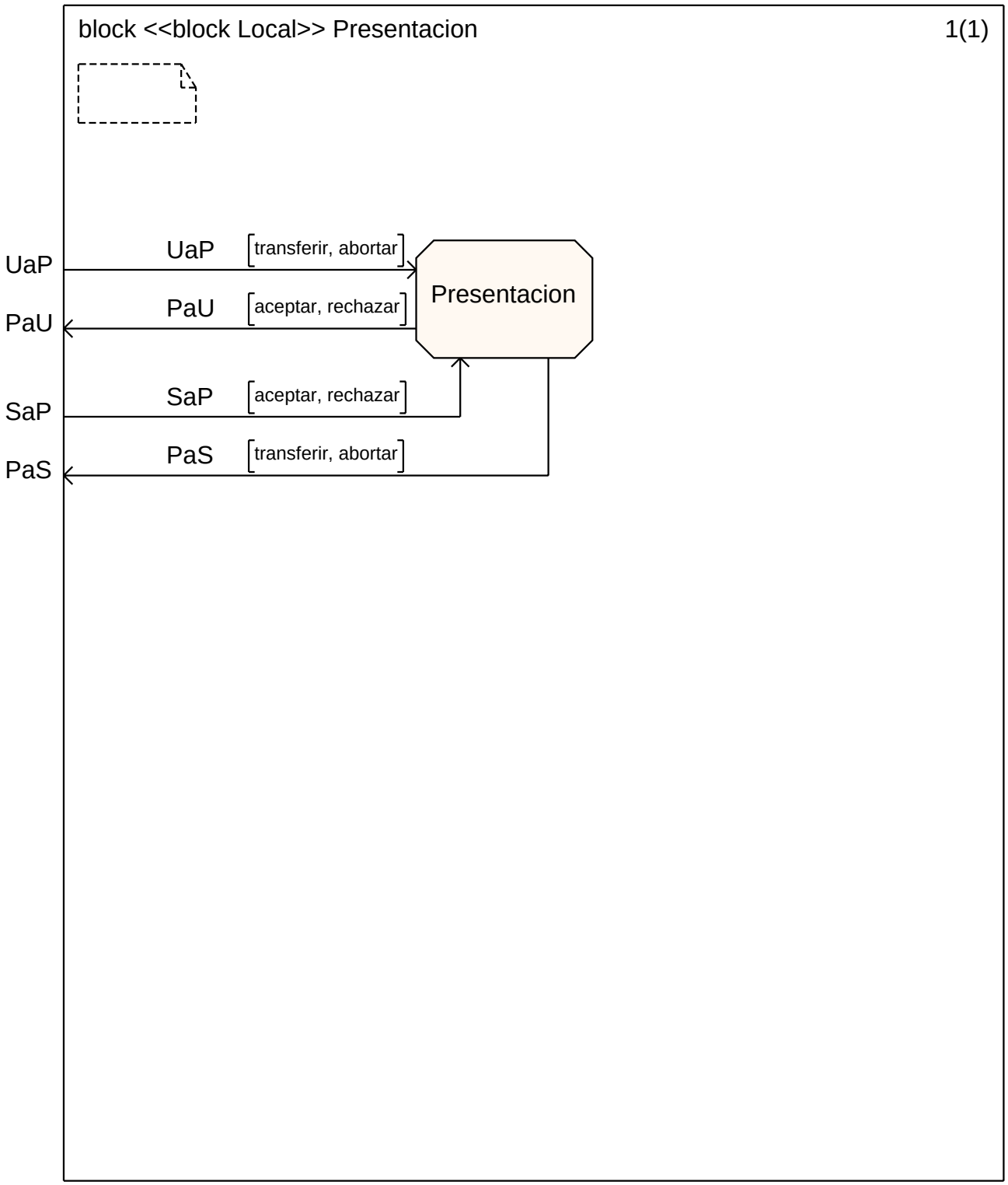
7.6.1. Capa de Presentación

Se dan errores fatales si:

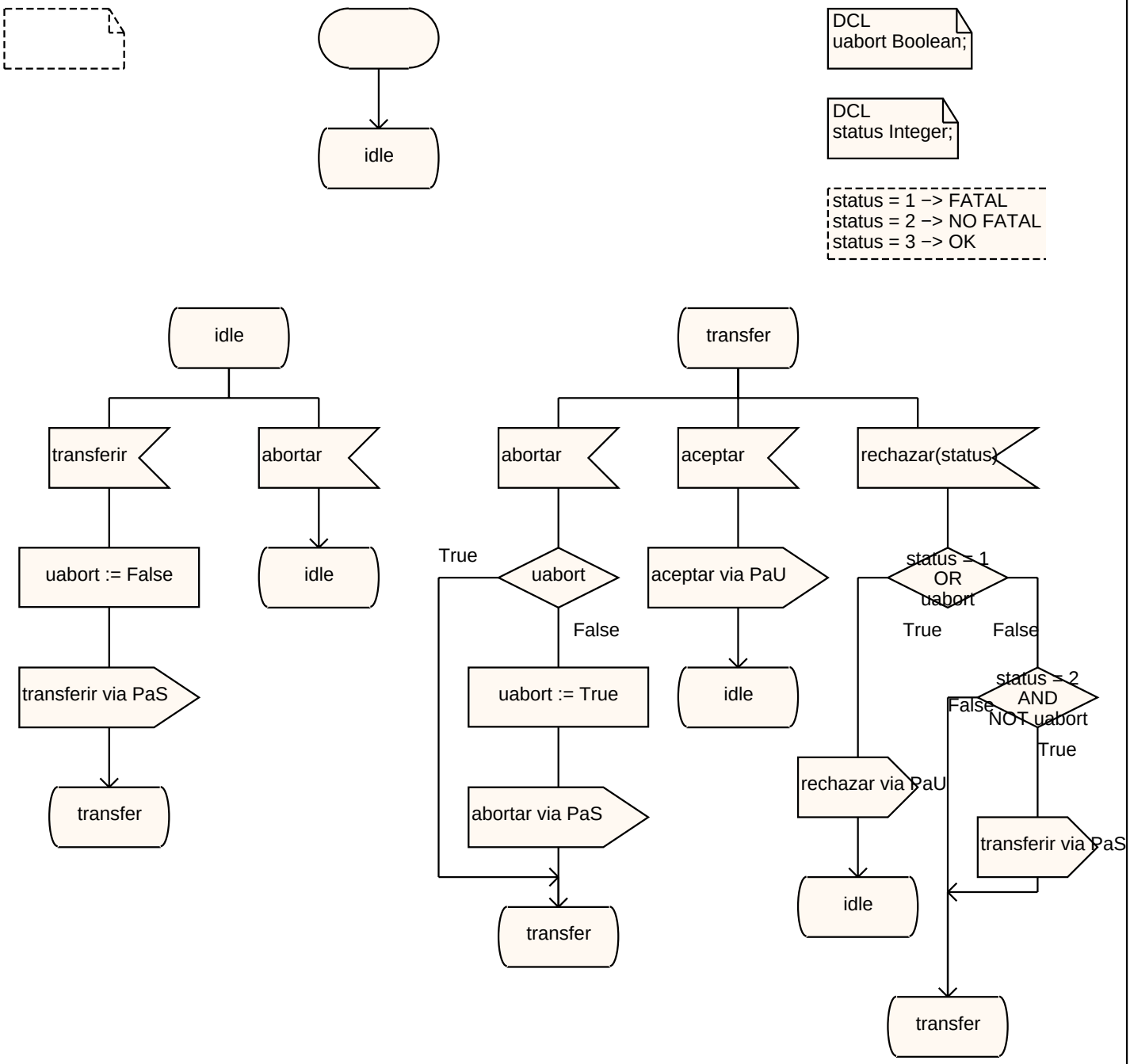
• El sistema de ficheros local rechaza una transmisión porque el fichero no existe.

• 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



7.6. Reglas de procedimiento.

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

7.6. Reglas de procedimiento.

7.6.2. Capa de Sesión

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.

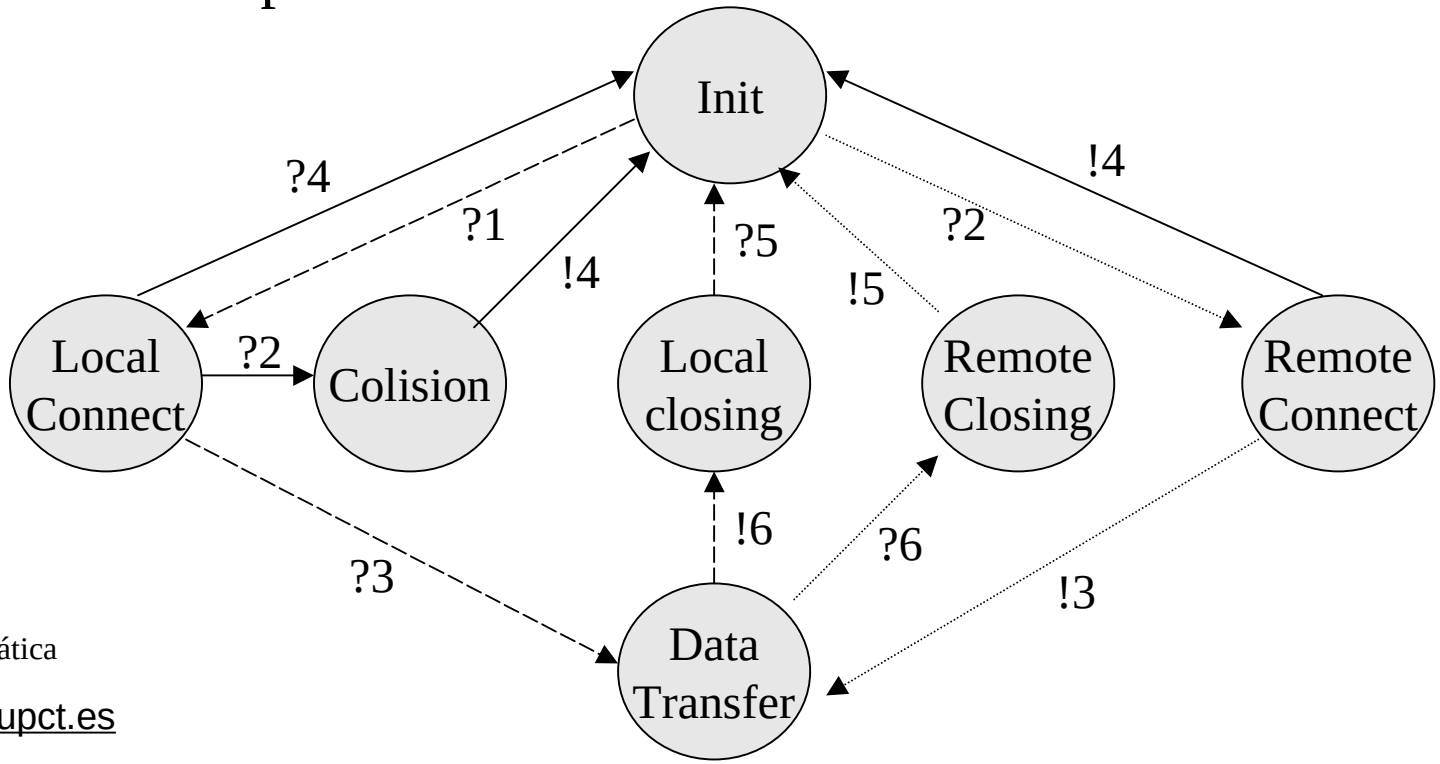
Las flechas a puntos indican una transferencia entrante satisfactoria.

Tema 7: DISEÑO DE PROTOCOLOS

Ingeniería de Protocolos y Servicios

7.6. Reglas de procedimiento.

7.6.2. Capa de Sesión



DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

7.6. Reglas de procedimiento.

7.6.2. Capa de Sesión

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.

Tema 7: DISEÑO DE PROTOCOLOS

Ingeniería de Protocolos y Servicios

7.6. Reglas de procedimiento.
7.6.2. Capa de Sesión
ESTABLECIMIENTO DE COMUNICACIÓN
Si se recibe de la capa inferior el mensaje *connect* la capa de sesión sólo debe intentar abrir un fichero local para escritura.

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

7.6. Reglas de procedimiento.

7.6.2. Capa de Sesión

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*.

DTIC
Área Ingeniería Telemática
francesc.burrull@upct.es

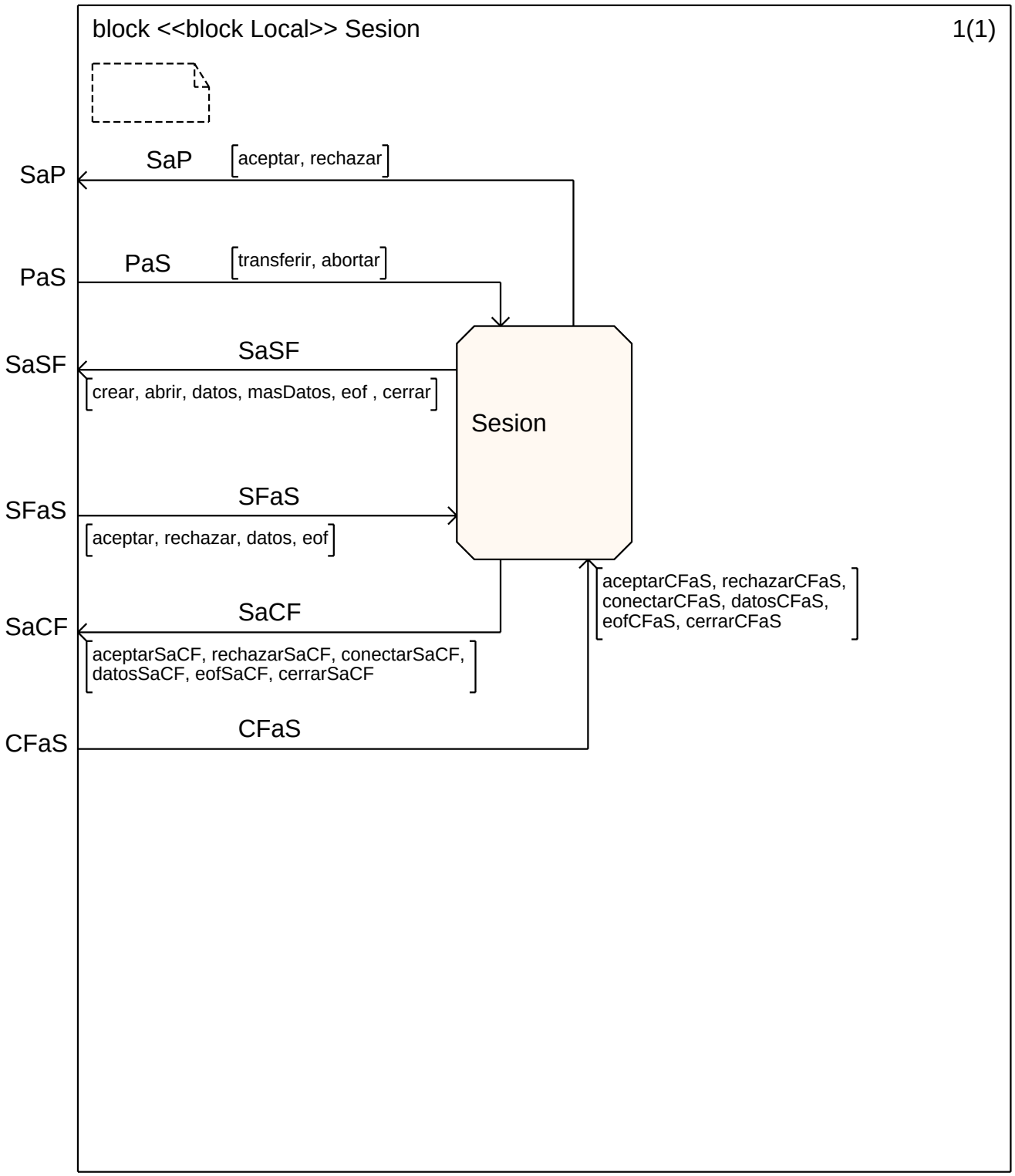
7.6. Reglas de procedimiento.

7.6.2. Capa de Sesión

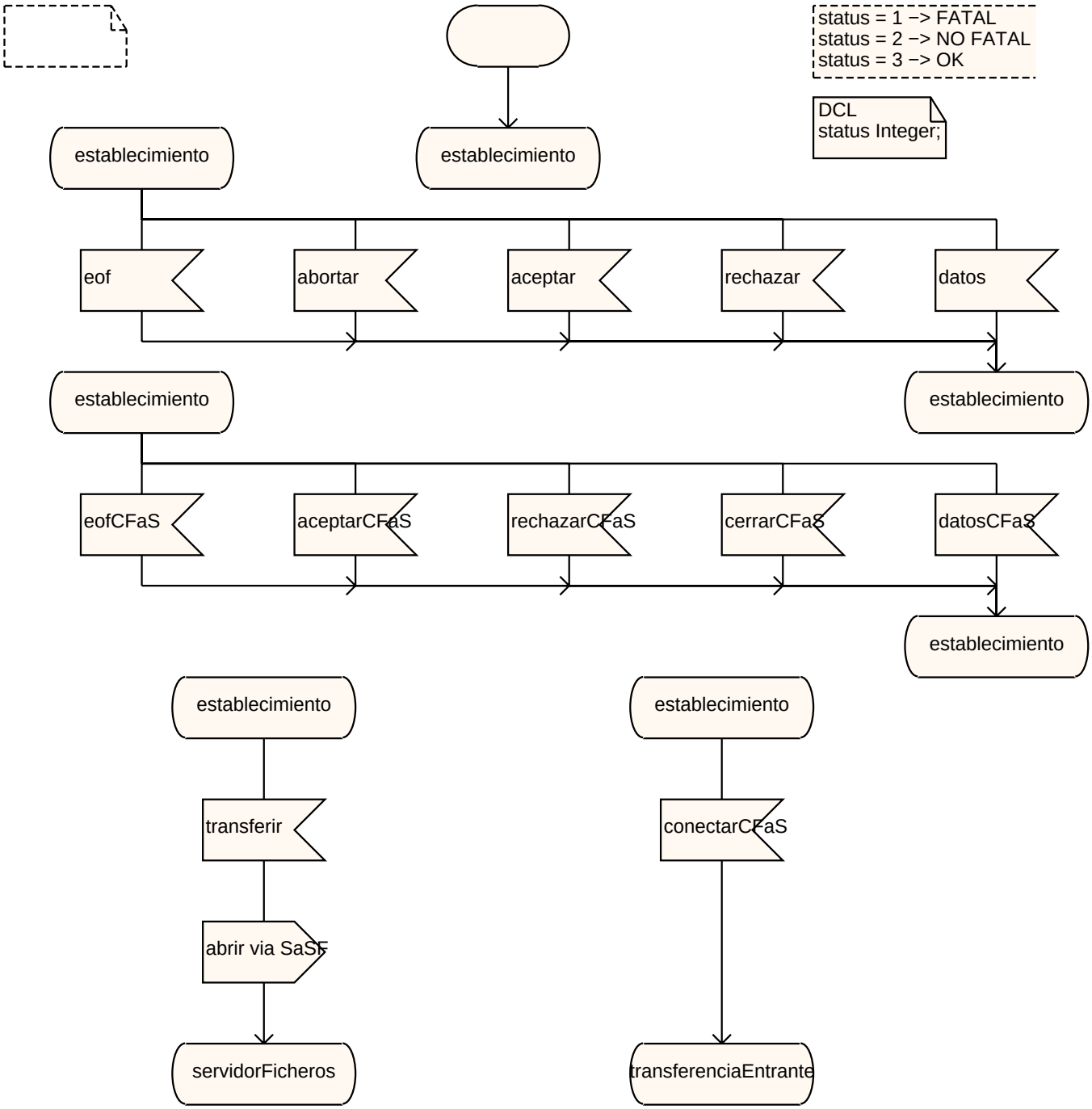
FINALIZACIÓN DE LA COMUNICACIÓN

En una transmisión saliente se devolverá a la capa superior *accept* o *reject*, en función del estado final de la transmisión.

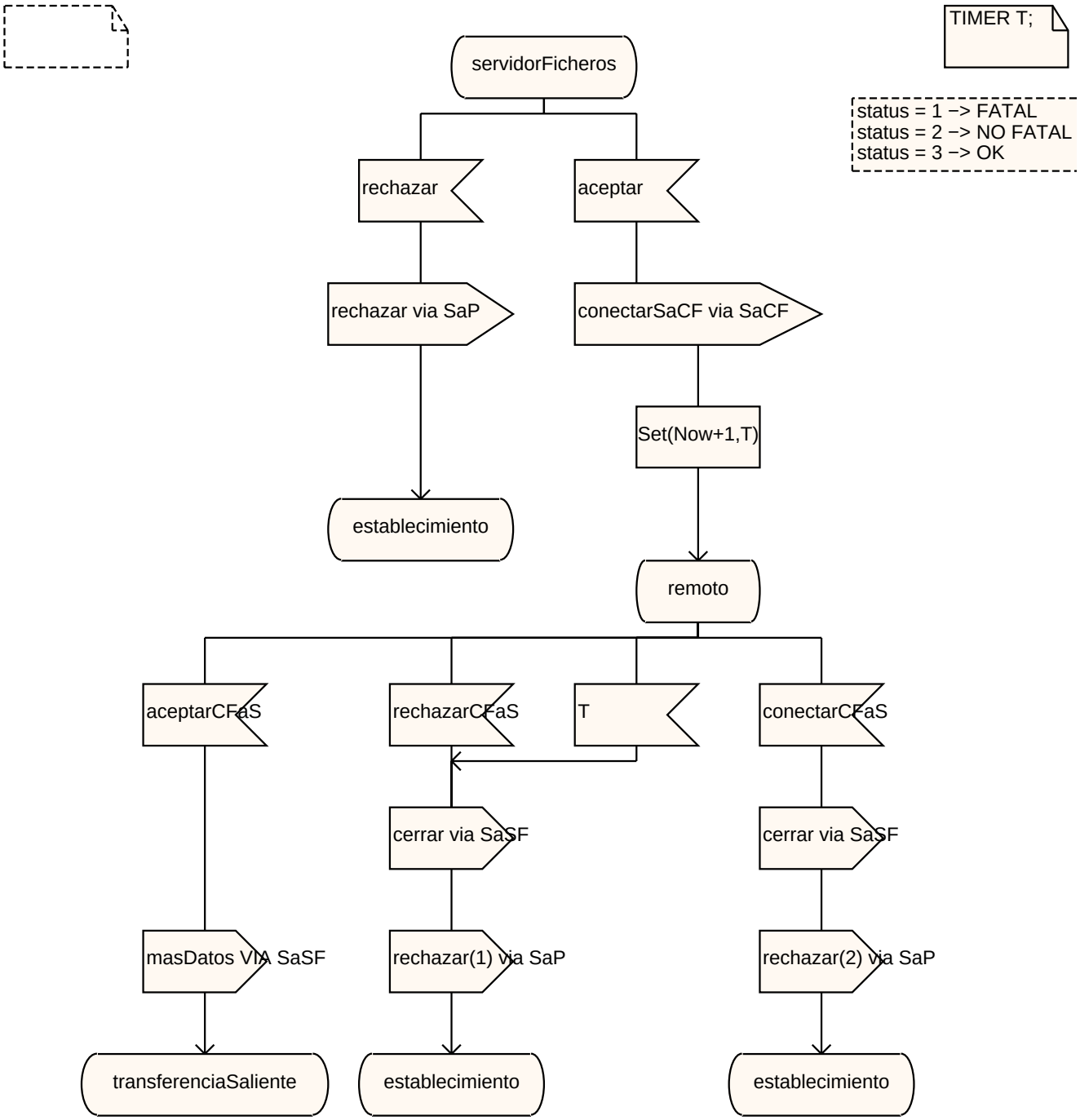
En una transmisión entrante se responderá a la capa inferior con *close* para confirmación.

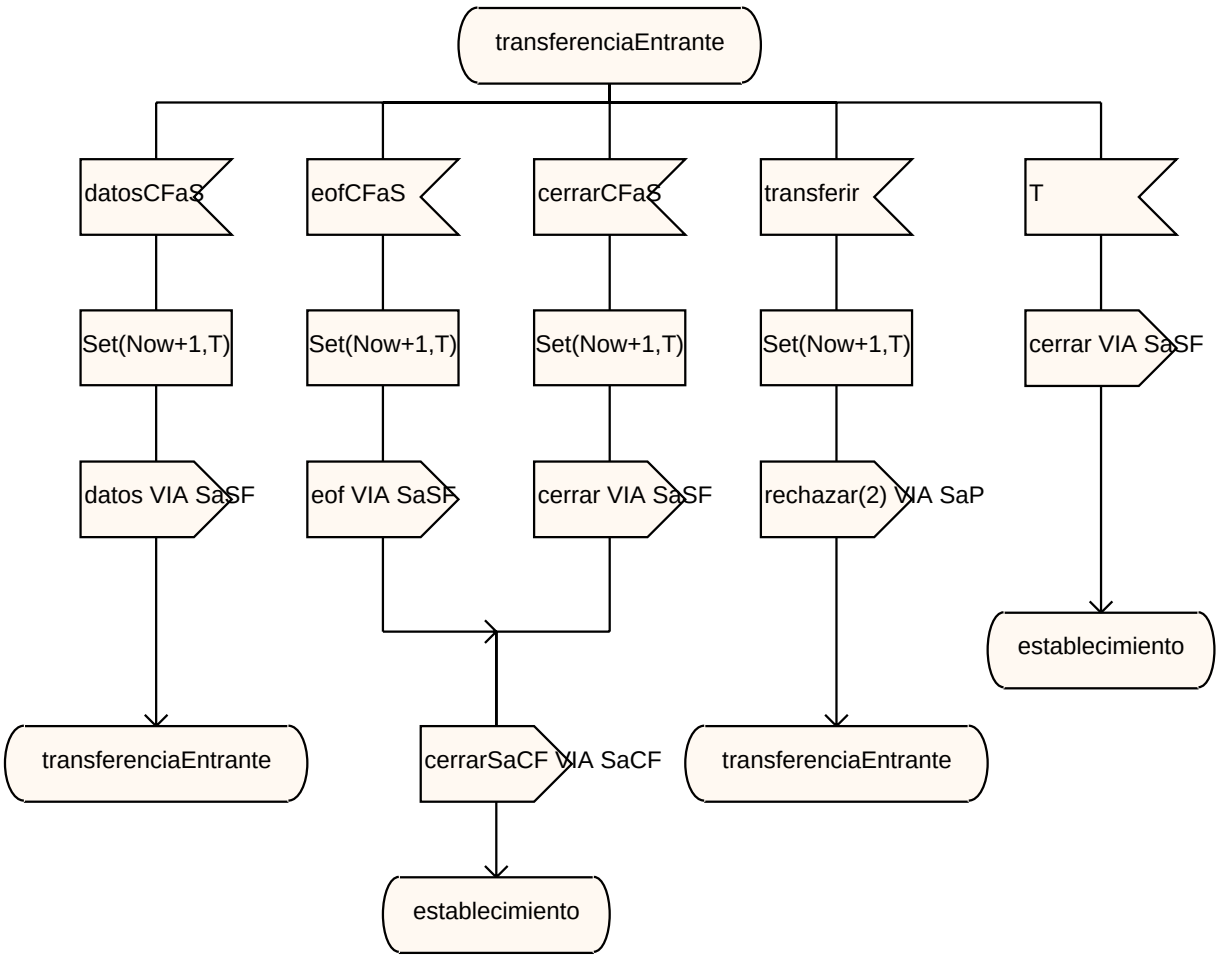

block <<block Local>> Sesion
1(1)
SaP
SaP
[aceptar, rechazar]
PaS
PaS
[transferir, abortar]
SaSF
SaSF
[crear, abrir, datos, masDatos, eof , cerrar]
Sesion
SFaS
SFaS
[aceptar, rechazar, datos, eof]
SaCF
SaCF
[aceptarSaCF, rechazarSaCF, conectarSaCF, datosSaCF, eofSaCF, cerrarSaCF]
CFaS
CFaS
[aceptarCFaS, rechazarCFaS, conectarCFaS, datosCFaS, eofCFaS, cerrarCFaS]

process <<block Local/block Sesion>> Sesion

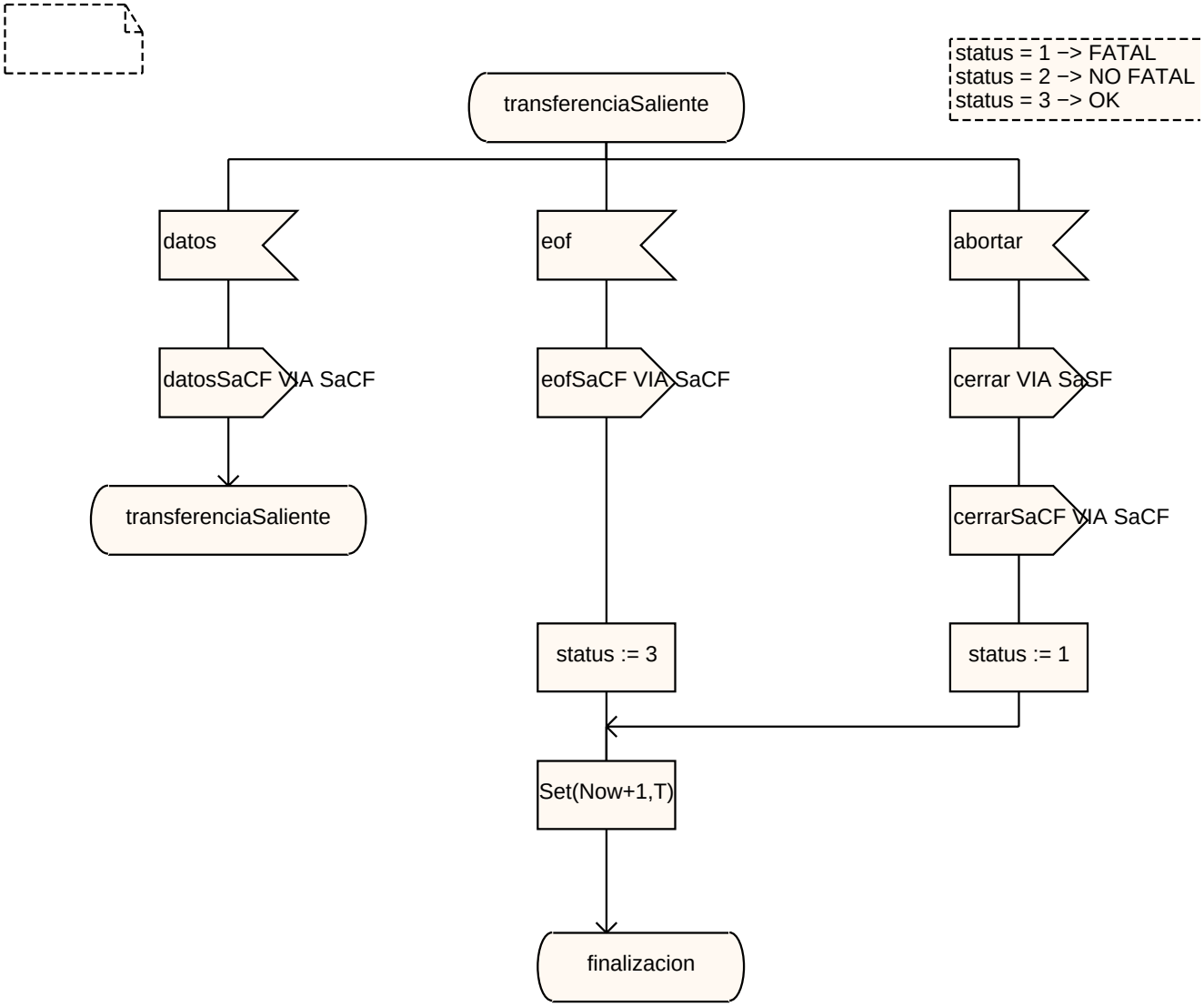


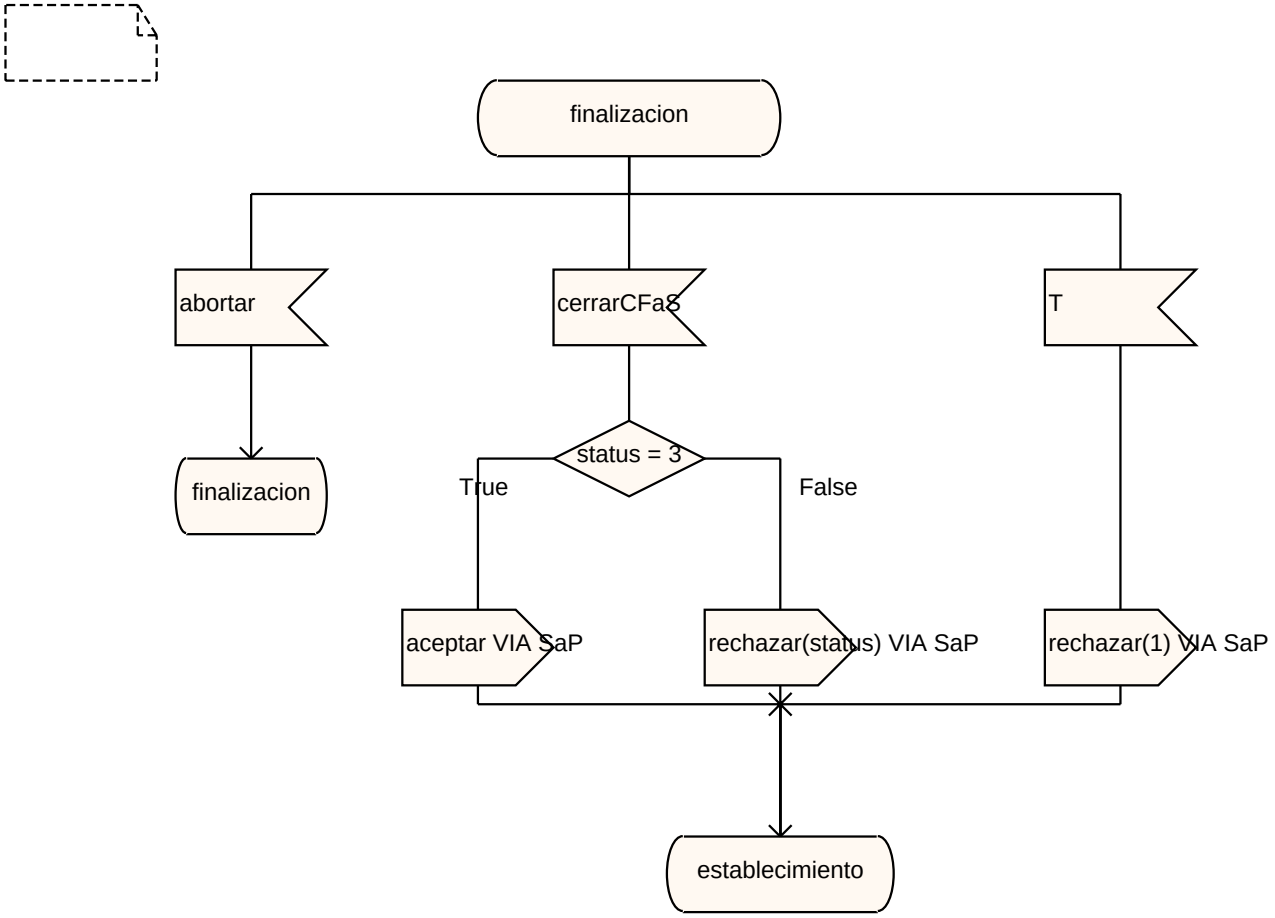
process <<block Local/block Sesion>> Sesion




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




finalizacion
abortar
cerrarCFaS
T
finalizacion
status = 3
True
False
aceptar VIA SaP
rechazar(status) VIA SaP
rechazar(1) VIA SaP
establecimiento

7.6. Reglas de procedimiento.

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.

7.6. Reglas de procedimiento.

7.6.3. Capa de Control de Flujo

Los mensajes entrantes se desencapsulan y se mandan a la capa superior.

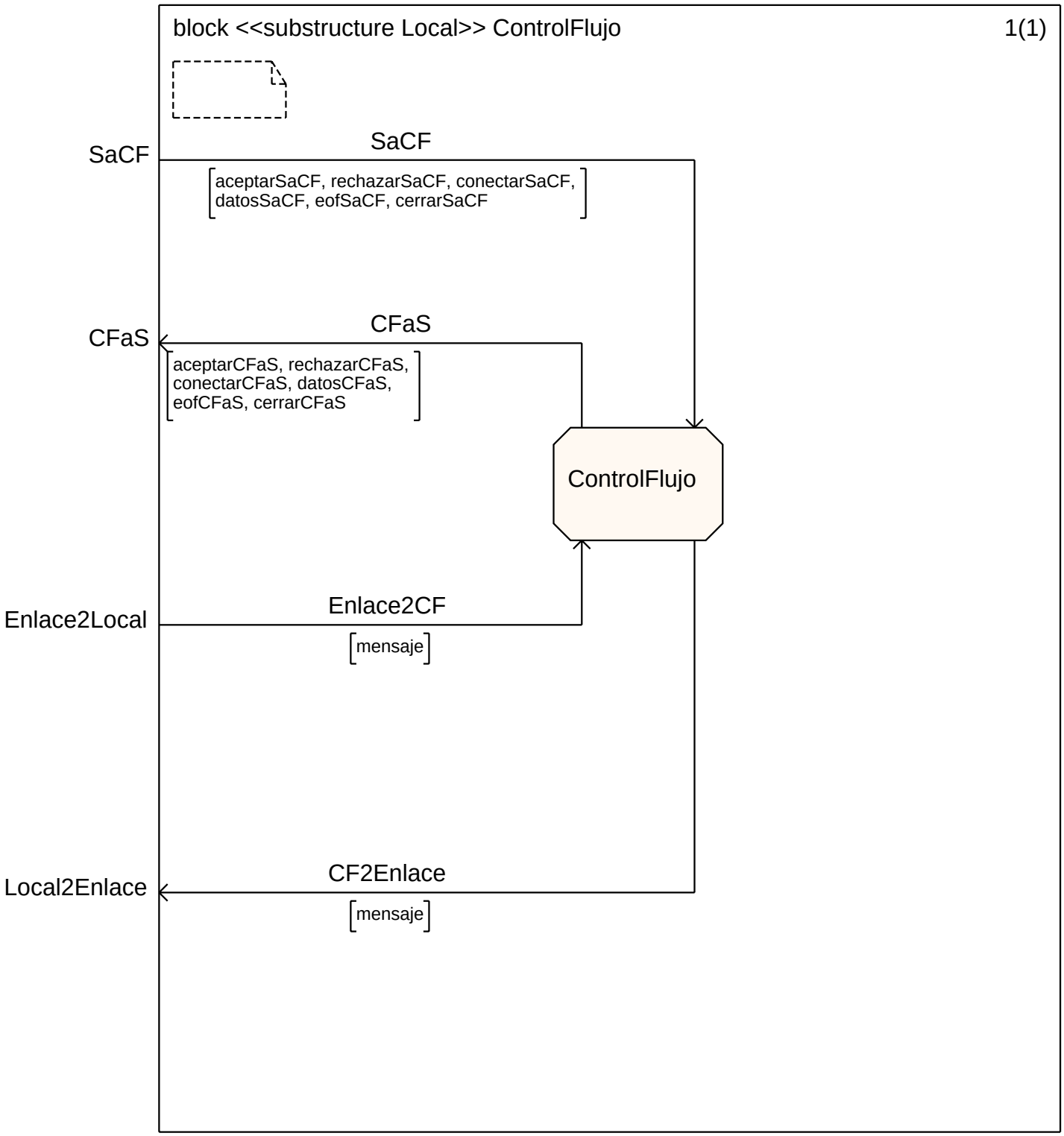
Al recibir un ack se libera el espacio del bufer que ocupaba el correspondiente mensaje. Si era el más viejo se desliza la ventana.

En recepción se mantiene un array de los mensajes que han llegado para enviarlos a la capa superior bien ordenados.

7.6. Reglas de procedimiento.

7.6.3. Capa de Control de Flujo

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

7.6. Reglas de procedimiento.

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).

7.7. Resumen.

Se ha desarrollado un modelo por capas jerárquicas que realizan funciones independientes. Cada capa añade funcionalidad.

Se ha obtenido un modelo de validación: el proceso de diseño no ha terminado. Hay que validar el modelo y derivar una implementación.

Sin el modelo de validación es prácticamente imposible hallar todos los errores (bloqueos, recepciones inesperadas, segmentos de código que no se ejecutan,...)