

**Reemplazada por una versión más reciente**



UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

**UIT-T**

**Z.100**

**Apéndices I y II**

**(03/93)**

SECTOR DE NORMALIZACIÓN  
DE LAS TELECOMUNICACIONES  
DE LA UIT

**LENGUAJES DE PROGRAMACIÓN**

---

**DIRECTRICES METODOLÓGICAS PARA  
EL LENGUAJE DE ESPECIFICACIÓN  
Y DESCRIPCIÓN**

**BIBLIOGRAFÍA**

**Recomendación UIT-T Z.100 – Apéndices I y II**  
**Reemplazada por una versión más reciente**

(Anteriormente «Recomendación del CCITT»)

---

# Reemplazada por una versión más reciente

## PREFACIO

El Sector de Normalización de las Telecomunicaciones de la UIT (UIT-T) es un órgano permanente de la Unión Internacional de Telecomunicaciones. El UIT-T tiene a su cargo el estudio de las cuestiones técnicas, de explotación y de tarificación y la formulación de Recomendaciones al respecto con objeto de normalizar las telecomunicaciones sobre una base mundial.

La Conferencia Mundial de Normalización de las Telecomunicaciones (CMNT), que se reúne cada cuatro años, establece los temas que habrán de abordar las Comisiones de Estudio del UIT-T, que preparan luego Recomendaciones sobre esos temas.

Los Apéndices I y II a la Recomendación UIT-T Z.100, preparados por la Comisión de Estudio X (1988-1993) del UIT-T, fueron aprobados por la CMNT (Helsinki, 1-12 de marzo de 1993).

---

## NOTAS

1 Como consecuencia del proceso de reforma de la Unión Internacional de Telecomunicaciones (UIT), el CCITT dejó de existir el 28 de febrero de 1993. En su lugar se creó el 1 de marzo de 1993 el Sector de Normalización de las Telecomunicaciones de la UIT (UIT-T). Igualmente en este proceso de reforma, la IFRB y el CCIR han sido sustituidos por el Sector de Radiocomunicaciones.

Para no retrasar la publicación de la presente Recomendación, no se han modificado en el texto las referencias que contienen los acrónimos «CCITT», «CCIR» o «IFRB» o el nombre de sus órganos correspondientes, como la Asamblea Plenaria, la Secretaría, etc. Las ediciones futuras en la presente Recomendación contendrán la terminología adecuada en relación con la nueva estructura de la UIT.

2 Por razones de concisión, el término «Administración» se utiliza en la presente Recomendación para designar a una administración de telecomunicaciones y a una empresa de explotación reconocida.

© UIT 1994

Reservados todos los derechos. No podrá reproducirse o utilizarse la presente Recomendación ni parte de la misma de cualquier forma ni por cualquier procedimiento, electrónico o mecánico, comprendidas la fotocopia y la grabación en micropelícula, sin autorización escrita de la UIT.

# Reemplazada por una versión más reciente

## ÍNDICE

*Página*

Apéndice I – Directrices metodológicas para el lenguaje de especificación y descripción.....	1
I.1    Introducción .....	1
I.1.1    Objetivo .....	1
I.1.2    Alcance .....	1
I.1.3    Convenios .....	1
I.1.4    Marco .....	1
I.1.5    Usos del SDL.....	3
I.1.6    Campo de aplicación del SDL .....	4
I.2    Modelado de aplicaciones .....	7
I.2.1    Servicios y protocolos OSI .....	7
I.2.2    Servicios RDSI según las Recomendaciones I.130 y Q.65.....	13
I.3    Elaboración de una especificación SDL por pasos.....	18
I.3.1    Introducción.....	19
I.3.2    Los pasos .....	19
I.3.3    Ejemplo: el ascensor (lift).....	29
I.4    Orientación a objetos y SDL .....	35
I.4.1    Análisis orientado a objetos.....	35
I.4.2    Aplicación de los conceptos SDL orientados a objetos .....	41
I.5    Elaboración de una especificación ADT completa por pasos.....	46
I.5.1    Compleción de una especificación ADT .....	46
I.5.2    Método básico de función-constructor .....	48
I.5.3    Cuatro pasos adicionales .....	52
I.5.4    Ecuaciones para constructores.....	53
I.5.5    Limitaciones .....	54
I.6    Utilización de los gráficos de secuencias de mensajes .....	54
I.6.1    Introducción.....	54
I.6.2    Especificación del sistema mediante mecanismos de composición de MSC .....	55
I.6.3    Método para desarrollar sistemas basado en MSC, SDL y descomposición funcional.....	65
I.7    Obtención de realizaciones a partir de especificaciones SDL .....	69
I.7.1    Introducción.....	71
I.7.2    Diferencias entre sistemas reales y sistemas SDL.....	73
I.7.3    Especificaciones de realización .....	78
I.7.4    Transacciones entre el soporte físico y el lógico .....	84
I.7.5    Diseño de la arquitectura del soporte lógico.....	86
I.7.6    Diseño de la arquitectura del soporte físico.....	105
I.7.7    Directrices para el diseño de la realización por pasos .....	105
I.8    Métodos formales de validación, verificación y prueba .....	107
I.8.1    Introducción.....	107
I.8.2    Validación y verificación.....	108
I.8.3    Prueba de conformidad.....	109

# Reemplazada por una versión más reciente

*Página*

I.9	Documentos auxiliares .....	110
I.9.1	Comunicación y especificación de interfaz.....	110
I.9.2	Diagrama arborescente .....	115
I.9.3	Diagrama general de estados .....	119
I.9.4	Matriz de señales y estados.....	120
I.10	Documentación.....	120
I.10.1	Introducción.....	120
I.10.2	Apoyo de lenguajes para la documentación.....	122
I.10.3	Correspondencia de las especificaciones con los documentos.....	124
I.10.4	Cuestiones relativas a la documentación .....	124
	Referencias.....	126
	Apéndice II – Bibliografía para el lenguaje de especificación y descripción.....	129

# Reemplazada por una versión más reciente

## Apéndice I a la Recomendación Z.100

### DIRECTRICES METODOLÓGICAS PARA EL LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN

(Helsinki, 1993)

#### I.1 Introducción

##### I.1.1 Objetivo

El objetivo del presente apéndice es proporcionar directrices para la utilización eficaz del lenguaje de especificación y descripción del CCITT (SDL, *specification and description language*) dentro de cierta metodología general.

Se ha reconocido que hay muchas maneras de utilizar el SDL, que dependen de la preferencia del usuario, la aplicación a que se destina, el reglamento interno de la organización, etc. Por ello, las directrices contenidas en este apéndice no son normativas y se deja al criterio de los usuarios la decisión de aplicarlas y en qué medida. Se publican porque se considera que son útiles e importantes no sólo para la promoción del SDL sino también para el uso uniforme y la sustentación del lenguaje.

##### I.1.2 Alcance

Este apéndice contiene directrices para:

- la utilización del SDL en diferentes sectores de aplicación;
- la elaboración gradual de una especificación SDL;
- el uso de gráficos de secuencias de mensajes;
- la obtención de realizaciones a partir de una especificación SDL;
- las técnicas formales de validación, verificación y prueba;
- los diagramas auxiliares;
- los aspectos de documentación.

Las directrices no constituyen una metodología única, coherente y completa. No se pretende que sean exhaustivas, específicas, ni detalladas, ni que restrinjan indebidamente la libertad de los usuarios del SDL, sino que éstos las seleccionen e incorporen en sus metodologías generales, adaptándolas a sus sistemas de aplicación y necesidades específicas.

##### I.1.3 Convenios

Las especificaciones formales no deben confundirse con el texto o figuras en lenguaje natural, por lo cual en este apéndice las especificaciones formales se tratan como Ejemplos o se incluyen en subcláusulas separadas. Los Ejemplos se numeran si procede, por ejemplo, cuando son referenciados. Para las especificaciones formales textuales se utiliza el tipo de letra «courier».

Un ejemplo puede abarcar sólo una parte de una especificación SDL completa. Las partes omitidas se indican mediante una línea de puntos o una línea de flujo abierta.

##### I.1.4 Marco

Según la definición del diccionario, una metodología es un sistema de métodos y principios utilizados en una disciplina determinada. Un método es una manera sistemática de hacer algo, o las técnicas u organización del trabajo relacionado con un campo o asunto determinado. En el contexto de este apéndice, las disciplinas consisten en la elaboración de sistemas de telecomunicación, protocolos, etc., llamados *sistemas de aplicación* (o simplemente *sistemas* en aras de la brevedad, cuando el calificativo es deducible del contexto).

La especificación y el diseño son *actividades*. Pueden existir varios métodos alternativos para la misma actividad. Así, la evaluación de los métodos alternativos y la elección de uno de ellos es una cuestión importante en el desarrollo de una metodología y debe basarse, entre otras cosas, en las características del sistema de aplicación.

## Reemplazada por una versión más reciente

Una actividad se caracteriza por sus entradas y salidas. La ejecución de una actividad está regida por algunas reglas que responden a las políticas de la empresa. En general, una actividad es desempeñada por personas que pertenecen a la empresa y emplean útiles de trabajo (véase la Figura I.1-1). Una actividad puede ser considerada como una máquina o sistema de estados finitos y ser descrita mediante técnicas similares (por ejemplo, con el SDL).

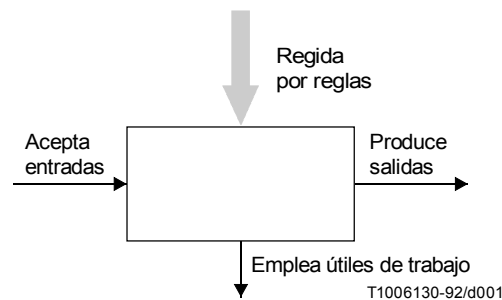


FIGURA I.1-1/Z.100

### Especificación de una actividad

Por lo común, una actividad se lleva a cabo en la representación (de parte) del sistema de aplicación. En consecuencia, la descomposición de una actividad en subactividades debe basarse en un modelo de información y en la arquitectura del sistema de aplicación.

Cada elemento de información se expresará en un lenguaje apropiado (natural, semiformal o formal). El mundo real se describe siempre en algún tipo de lenguaje natural, lo que resulta muy adecuado para expresar propósitos e intenciones, pero no sirve para expresar los detalles con precisión y sin ambigüedad. Un lenguaje formal tiene las propiedades contrarias. Una metodología debe utilizar diferentes categorías de lenguaje, de modo que se complementen y se pueda aprovechar sus ventajas.

De acuerdo con lo expuesto anteriormente, una metodología (en el contexto de este documento) consiste generalmente en los siguientes componentes:

- modelo de empresa;
- modelo de actividad;
- arquitectura del sistema;
- modelo de información;
- lenguajes;
- procedimientos de trabajo;
- reglas de tratamiento y documentación de productos;
- utilización de herramientas de trabajo.

El *modelo de empresa* describe los objetivos generales de un sistema de aplicación en términos de acciones, fines y políticas. Especifica las actividades que se desarrollan en el seno de la organización que utiliza el sistema de aplicación, las funciones que las personas desempeñan en la organización y las interacciones entre la organización, el sistema de aplicación y el entorno en el que están ubicados el sistema de aplicación y la organización.

El *modelo de actividad* determina todas las actividades correspondientes a un sistema de aplicación y sus interrelaciones. Cada actividad se describe como se ha mostrado más arriba. Nótese que el modelo de actividad se denomina comúnmente modelo de ciclo de vida. Esta expresión debe evitarse porque implica, erróneamente, una ordenación lineal de las actividades. Normalmente, un modelo de actividad es bidimensional, lo que permite la ejecución simultánea de actividades.

La *arquitectura de la aplicación* proporciona, en principio, una descomposición del sistema de aplicación en partes autónomas y reglas para la interacción entre esas partes.

# Reemplazada por una versión más reciente

El *modelo de información* determina toda la información pertinente para solicitar, producir, mantener y entregar el sistema de aplicación así como la relación entre las diferentes piezas de información.

Los *lenguajes* proporcionan medios para expresar el modelo de información con la precisión adecuada que permiten efectuar varias verificaciones y utilizar herramientas informatizadas.

Los *procedimientos de trabajo* hacen que algunas reglas comúnmente aceptadas sean aplicadas por todos los miembros del proyecto para que la labor sea eficaz y se consigan resultados de alta calidad.

Las *reglas de tratamiento y documentación de productos* complementan los procedimientos de trabajo y garantizan la capacidad de mantener la documentación de la aplicación.

Las *herramientas* (o útiles de trabajo) aumentan la productividad de los miembros del proyecto y la calidad de los productos porque efectúan tareas bien definidas y laboriosas.

NOTA – Una metodología comprende diferentes niveles de abstracción. El modelo de actividad abarca todos los niveles de abstracción, pero es probable que algunos de los demás componentes de la metodología no sean aplicables en todos los niveles de abstracción. Por ejemplo, los procedimientos de trabajo sólo son aplicables en los niveles inferiores.

## I.1.5 Usos del SDL

El SDL es un lenguaje de especificación formal. Para abreviar, en adelante se omitirá el término *formal*, que queda implícito cuando se trata de lenguajes de especificación.

Un criterio ampliamente aceptado es que el éxito de un sistema depende de que la especificación y el diseño sean exactos y completos. Para ello hace falta un lenguaje de especificación adecuado, que satisfaga las necesidades siguientes (en este documento y para ser más concisos, se utiliza el término *especificación* para englobar las actividades de especificación y diseño del sistema):

- un conjunto de conceptos bien definido; y
- especificaciones *claras, precisas, concisas y sin ambigüedades*; y
- una base para *analizar la integridad* (cualidad de completo) y *corrección* (ajuste a las normas) de las especificaciones; y
- una base para determinar la *conformidad* de las realizaciones con las especificaciones; y
- una base para determinar la *coherencia* de las especificaciones entre sí; y
- el empleo de *herramientas* informatizadas para crear, mantener, analizar y simular especificaciones.

Las especificaciones de un sistema pueden corresponder a diferentes niveles de abstracción. Una especificación sirve de base para las *realizaciones*, pero no tendrá en cuenta los detalles de la realización con el fin de:

- dar una visión global de un sistema complejo;
- posponer las decisiones relativas a la realización; y
- no excluir realizaciones válidas.

A diferencia de un programa, una especificación formal (o sea, una especificación escrita en un lenguaje de especificación) no se utilizará en un computador. Además de servir de base para las realizaciones, una especificación formal se puede utilizar para que los usuarios se comuniquen con precisión y sin ambigüedad, especialmente en lo que respecta a los pedidos y licitaciones.

El uso de un lenguaje de especificación permite analizar y simular soluciones de sistema alternativas, lo que no se consigue con un lenguaje de programación dado su coste y tiempo de ejecución. Un lenguaje de especificación ofrece al usuario un conjunto de conceptos bien definido, que aumenta su capacidad para elaborar una solución a un problema y razonarla.

### I.1.5.1 Interpretación de una especificación formal

Como ya se ha dicho, un lenguaje de especificación ofrece un conjunto de conceptos bien definido. Estos conceptos forman una especie de modelo matemático y, por tanto, puede parecer extraño a algunas personas que no están familiarizadas con la teoría matemática subyacente. Por este motivo, a continuación se explica de manera preliminar cómo aplicar el modelo formal de un lenguaje de especificación a una aplicación.

El ámbito de aplicación de un sistema se comprende mediante conceptos expresados en lenguaje natural adquiridos en un largo proceso de aprendizaje y por experiencia. La descripción de una aplicación con un lenguaje natural es, por naturaleza, descriptiva: los fenómenos se describen tal y como los percibe el observador. Para la descripción del sistema con lenguaje natural se utilizan conceptos derivados directamente de la aplicación y realización del sistema.

## Reemplazada por una versión más reciente

Cuando un sistema se especifica con un lenguaje de especificación, la especificación formal no utiliza conceptos de la aplicación ni de la realización, sino que más bien define un *modelo* que representa las propiedades significativas (principalmente el comportamiento) del sistema. Para entender este modelo es preciso establecer su correspondencia con la comprensión intuitiva de la aplicación en términos de conceptos del lenguaje natural (véase la Figura I.1-2). Esa correspondencia se puede efectuar de diversas maneras, una de las cuales consiste en elegir nombres para los conceptos presentados en la especificación formal que estén bien asociados con los conceptos de la aplicación, y otra en comentar la especificación formal. Esto es muy semejante a la interpretación de un programa o de un algoritmo que resuelve un problema tomado de la vida real.

Un modelo debe tener un *poder analítico* adecuado, como se indica en I.1.5, y un buen *poder expresivo* para facilitar la correspondencia con la aplicación. Por desgracia, estas propiedades son generalmente contradictorias: cuanto más expresivo es el modelo, tanto más difícil resulta analizarlo. Es obvio que para diseñar un lenguaje de especificación hay que encontrar una solución transaccional para estas dos propiedades. Además, es preciso insistir en que un modelo es siempre una visión simplificada de la realidad y, por consiguiente, tiene limitaciones inherentes.

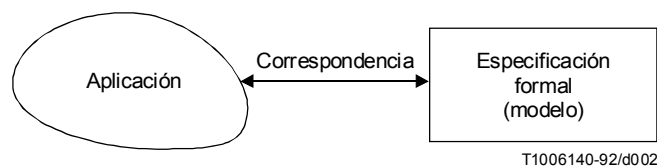


FIGURA I.1-2/Z.100

### Interpretación de una especificación formal

#### I.1.5.2 Relación entre el lenguaje y la realización

Normalmente se distingue entre lenguajes *declarativos* y *constructivos*. El lenguaje SDL es constructivo, lo que significa que una especificación SDL define un *modelo* que representa las propiedades significativas de un sistema (como ya se ha dicho). La determinación de estas propiedades significativas es una cuestión importante, que en el SDL se deja abierta, ya que *corresponde al usuario decidir cuáles han de ser estas propiedades*.

Si se decide representar sólo el comportamiento del sistema (visto en su frontera) se habla, en general, de *especificación*, y la estructura dada del modelo es simplemente una ayuda para estructurar la especificación en piezas manejables. Si se decide también representar la estructura interna del sistema, se habla, en general, de *descripción*. Es por ello que el SDL no distingue entre especificación y descripción.

Obsérvese que la estructura interna del sistema está normalmente representada en el SDL por *bloques*. No es necesario que los *procesos* y la señalización dentro de los bloques representen parte de la estructura interna del sistema, por lo que no hay que considerarlos como requisitos de realización, como algunos suponen erróneamente. En general, las organizaciones de normalización tratan la conformidad entre realizaciones y especificaciones mediante *enunciados de conformidad explícitos*. Es obvio que hay que hacer lo mismo cuando se utiliza un lenguaje constructivo.

#### I.1.6 Campo de aplicación del SDL

En la actualidad el SDL es conocido principalmente en el sector de las telecomunicaciones, aunque tiene un campo de aplicación más amplio que se puede caracterizar como sigue:

- *tipo de sistema*: en tiempo real, interactivo, distribuido;
- *tipo de información*: comportamiento y estructura;
- *nivel de abstracción*: general hasta detalles.

El SDL se concibió para los sistemas de telecomunicación, incluida la comunicación de datos, pero de hecho se puede utilizar en todos los sistemas interactivos y en tiempo real. Fue diseñado para especificar el comportamiento de ese sistema, es decir, la cooperación entre el sistema y su entorno. Se ideó también describir la estructura interna de un sistema, de manera que se pueda construir y comprender parte por parte. Esta es una característica fundamental de los sistemas distribuidos.



# Reemplazada por una versión más reciente

El SDL abarca varios niveles de abstracción, desde la visión general hasta el diseño detallado. No se previó como lenguaje de realización. Sin embargo, es posible la traducción más o menos automática de una especificación SDL a un lenguaje de programación.

La Figura I.1-3 muestra una gama de posibles usos del SDL en el contexto de la compra y el suministro de sistemas de conmutación para telecomunicaciones. En dicha figura los rectángulos representan actividades típicas cuyos nombres exactos pueden variar de una organización a otra. Cada flecha (línea de flujo) representa un conjunto de documentos que pasa de una actividad a otra; se pueden utilizar las especificaciones SDL como parte de cada uno de esos conjuntos de documentos. Esta figura es de carácter puramente ilustrativo y no se pretende que sea definitiva ni exhaustiva.

Al referirse especialmente a los sistemas de conmutación como ejemplos de funciones que pueden ser documentadas con el SDL, cabe citar: el procesamiento de las comunicaciones (tratamiento de llamadas, encaminamiento, señalización, tasación, etc.), el mantenimiento y la reparación de averías (alarmas, liberación automática en caso de fallo, configuración del sistema, pruebas periódicas, etc.), el control del sistema (control de sobrecarga) y las interfaces hombre-máquina.

La especificación de protocolos mediante el SDL se trata en las Recomendaciones de la serie X.

## I.1.6.1 Naturaleza del soporte lógico de conmutación

Se puede considerar que el soporte lógico de conmutación tiene ciertas características propias. Con mucha frecuencia funciona como un *sistema anidado* en una plataforma de soporte físico dedicada a una tarea específica (por ejemplo, un conmutador). Como es natural, el soporte físico impone límites a lo que puede hacer el soporte lógico. En comparación con otros campos de desarrollo de soporte lógico, los *requisitos funcionales* están bien definidos, en el sentido de que son precisos, más bien sencillos, inequívocos y, con bastante frecuencia, incluso están formalmente definidos. Casi nunca los requisitos cambian durante el desarrollo del soporte lógico. Una gran parte del soporte lógico de conmutación puede considerarse *reactiva* - se envía un mensaje al sistema y se supone que éste responde. El soporte lógico de conmutación suele funcionar en *tiempo real*, pero hay que destacar que a menudo los requisitos de funcionamiento son más *estadísticos* que absolutos. Por regla general, el soporte lógico de conmutación es *paralelo* y *distribuido*.

## I.1.6.2 Función del SDL en el soporte lógico de conmutación

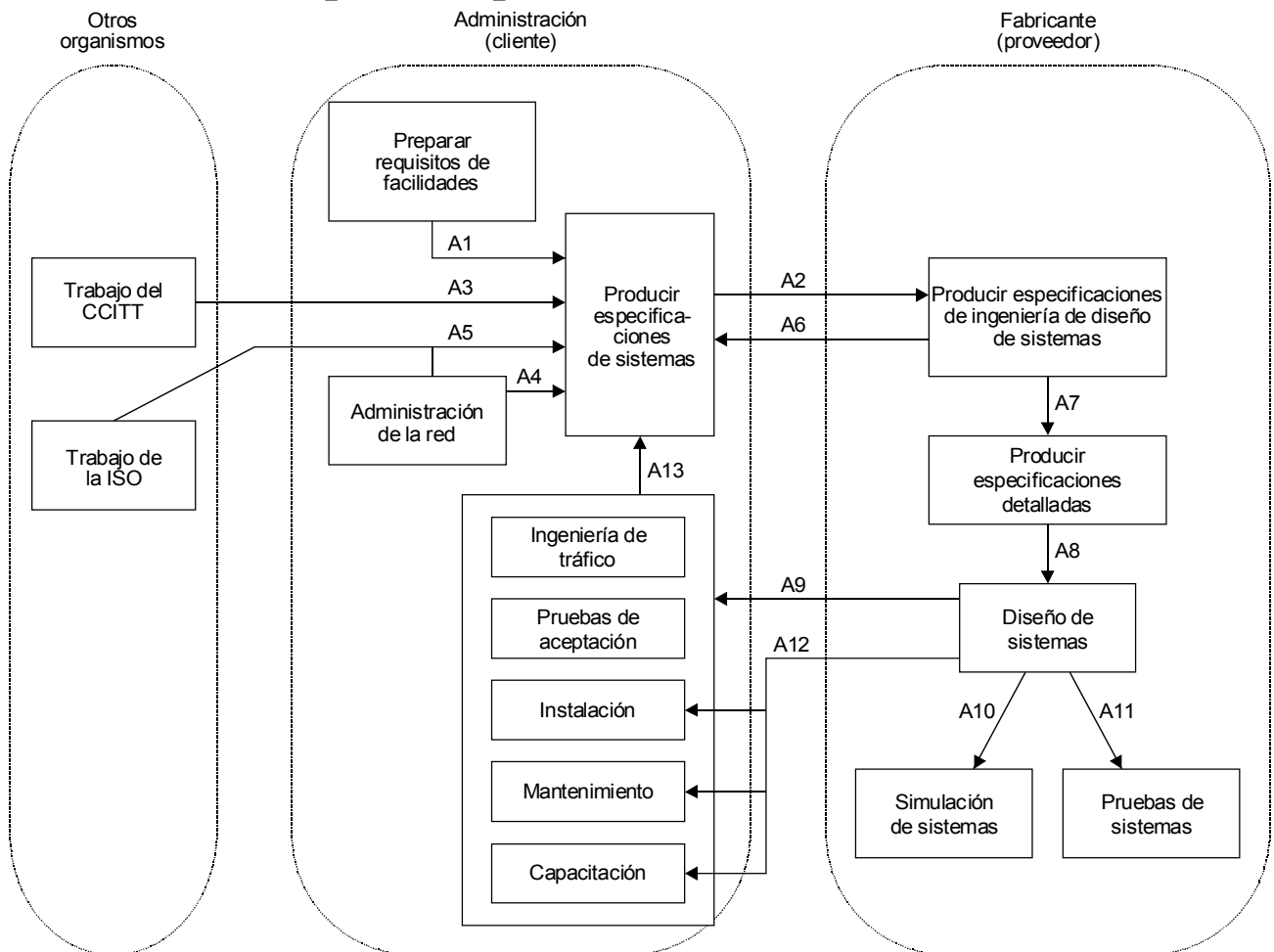
Muchos de los diseñadores que participan en la elaboración de soportes lógicos de conmutación perciben el SDL más como un *lenguaje de descripción* (que expresa cómo funciona realmente el sistema) que como un *lenguaje de especificación* (que expresa cómo debe funcionar el sistema). Para el diseñador, la actividad de elaboración del soporte lógico comienza cuando recibe la especificación de los requisitos, posiblemente escrita en SDL, a partir de la que se obtiene el sistema de aplicación orientado a la realización.

La aplicación del SDL como lenguaje de descripción para los sistemas de conmutación es una actividad disciplinada que integra múltiples aspectos de la ingeniería de soporte lógico que exigen diversas capacidades. El SDL proporciona un marco conceptual suficiente para apoyar los diferentes niveles de abstracción desde la especificación de los requisitos hasta la realización.

La prueba de especificaciones SDL plantea problemas importantes. En primer lugar, los sistemas de soporte lógico de conmutación son muy grandes y frecuentemente el diseñador sólo tiene un conocimiento general del entorno de su código. Cabe imaginar que los módulos determinados para comunicar entre sí se construyen al mismo tiempo. Ello significa que habrá que elaborar un *entorno de prueba* para cada componente. La *prueba de integración* y la *prueba de sistema* sólo pueden ser efectuadas posteriormente. En segundo lugar, los sistemas anidados no siempre proporcionan herramientas para la elaboración de un soporte lógico de calidad semejante a la de los sistemas informatizados corrientes, de manera que el proyectista tiene que conformarse con la supervisión del intercambio de mensajes y la simple eliminación de errores del programa (debugging). A veces, otra posibilidad consiste en simular el comportamiento de la especificación SDL concreta en una plataforma de soporte físico más desarrollada para garantizar el funcionamiento correcto del diseño. Otra opción es generar casos de prueba o entradas de prueba directamente a partir de la especificación SDL. En tercer lugar, es aconsejable verificar la *conformidad* del diseño SDL con respecto a la especificación de los requisitos, aunque con mucha frecuencia eso se hace de forma manual y poco rigurosa.

Después de la entrega, se debe *mantener* el diseño del sistema; ésta es una de las razones por las que la especificación SDL debe ser legible y estar bien documentada. Cada acción de mantenimiento desencadenará un nuevo ciclo de prueba, de preferencia con los mismos datos de prueba utilizados con anterioridad. Varias partes de los sistemas de conmutación están *adaptadas* a las exigencias del cliente, por lo que es menester contar con un sistema adecuado de *control de versiones*.

# Reemplazada por una versión más reciente



T1006150-92/d003

- A1 Especificación de una facilidad o característica independiente de la realización y de la red
- A2 Especificación de un sistema independiente de la realización pero dependiente de la red; incluida una descripción del entorno del sistema
- A3 Recomendaciones y directrices del CCITT
- A4 Contribuciones a la especificación del sistema que indica los requisitos de administración y operacionales de la red
- A5 Otras normas pertinentes
- A6 Descripción de una propuesta de realización
- A7 Especificación de proyecto
- A8 Especificación detallada de un diseño
- A9 Especificación completa de un sistema
- A10 Documentación adecuada de un sistema y su entorno, para la simulación del sistema
- A11 Documentación adecuada de descripción de un sistema y de su entorno, para la prueba del sistema
- A12 Manuales de instalación y funcionamiento
- A13 Contribuciones a la especificación del sistema preparadas por grupos funcionales especializados dentro de la Administración

## NOTAS

- 1 Es posible la iteración en todos los niveles.
- 2 En ciertas circunstancias, podría suministrarse a otra organización la documentación SDL que en este diagrama se indica como interna de una organización, por ejemplo, A1, A7, A8.

FIGURA I.1-3/Z.100  
Ámbito general del uso del SDL

# Reemplazada por una versión más reciente

## I.2 Modelado de aplicaciones

### I.2.1 Servicios y protocolos OSI

#### I.2.1.1 Introducción

Los conceptos de interconexión de sistemas abiertos (OSI, *open system interconnection*) utilizados en I.2.1 se definen en [1] y [2]. Para que esta subcláusula sea más autónoma, se ofrece una explicación de los conceptos clave.

Un *servicio de capa* es ofrecido por un *proveedor de servicio* para una capa determinada. El proveedor de servicio es una máquina abstracta, que ofrece una facilidad de comunicación a los *usuarios* de la capa superior siguiente. Los usuarios acceden al servicio en los *puntos de acceso al servicio* mediante *primitivas de servicio* (véase la Figura I.2-1). Una primitiva de servicio puede utilizarse para la gestión de la conexión (conexión, desconexión, reiniciación, etc.), o ser un objeto de datos (datos normales o datos acelerados). Existen únicamente cuatro tipos de primitivas de servicio:

- petición (de usuario a proveedor);
- indicación (de proveedor a usuario);
- respuesta (de usuario a proveedor);
- confirmación (de proveedor a usuario).



FIGURA I.2-1/Z.100

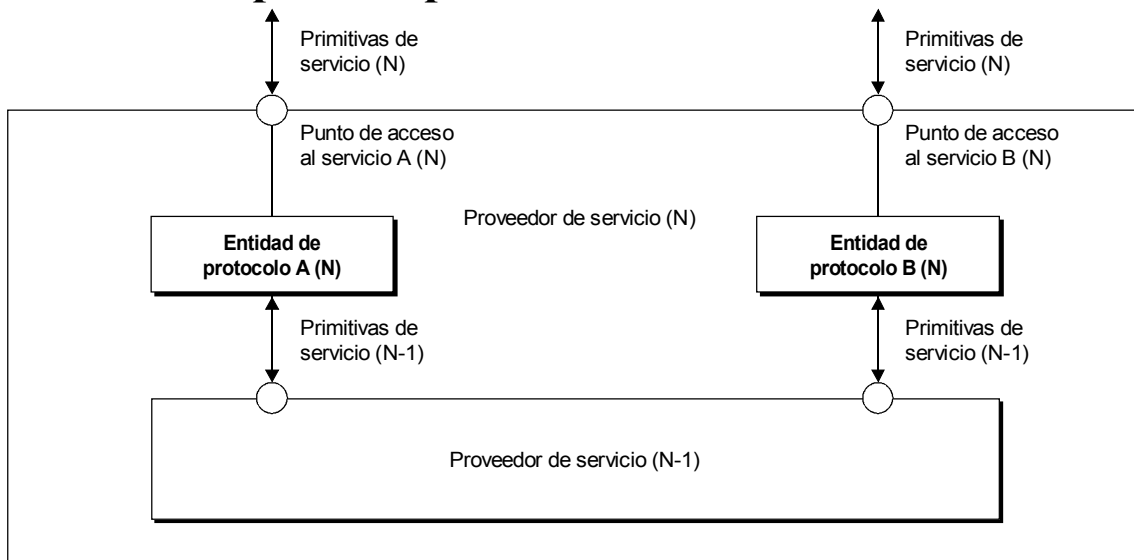
#### Proveedor de servicio de capa

Una *especificación de servicio* es una manera de caracterizar el comportamiento del proveedor del servicio, localmente, al enunciar las secuencias autorizadas de primitivas de servicio transferidas en un punto de acceso al servicio, y de extremo a extremo, así como al enunciar la relación correcta entre las primitivas de servicio transferidas en diferentes puntos de acceso al servicio. La especificación de servicio no se ocupa de la estructura interna del proveedor del servicio; toda estructura interna determinada al especificar el servicio es sólo un modelo abstracto que sirve para describir el comportamiento externamente observable del proveedor del servicio.

Salvo en la capa superior, los usuarios de un servicio de capa son *entidades de protocolo* de la capa superior siguiente, que cooperan para mejorar las características del servicio de capa, gracias a lo cual proporcionan un servicio de la capa superior siguiente. Esa cooperación se efectúa de conformidad con un conjunto predefinido de reglas de comportamiento y formatos de mensaje que constituyen un *protocolo*. Según este punto de vista, las entidades de protocolo de la capa (N) y el proveedor de servicio (N - 1) proporcionan, en conjunto, un refinamiento del proveedor de servicio (N) (véase la Figura I.2-2).

Naturalmente el refinamiento del proveedor de servicio (N) mostrado en la Figura I.2-2 puede ser mucho más complicado. Por ejemplo, puede haber entidades de protocolo de retransmisión (N) que no están conectadas a ninguna entidad de protocolo de la capa (N + 1). Esos casos no se tienen en cuenta en este documento por motivos de brevedad.

## Reemplazada por una versión más reciente



T1006170-92/d005

FIGURA I.2-2/Z.100

### Refinamiento del proveedor de servicio (N)

Las entidades de protocolo se comunican mediante el intercambio de *unidades de datos de protocolo*. Estas se transfieren en forma de parámetros de las primitivas de servicio de la capa subyacente. La entidad de protocolo emisora codifica las unidades de datos de protocolo en primitivas de servicio y la entidad de protocolo receptora decodifica las unidades de datos de protocolo de las primitivas de servicio recibidas. Un protocolo se basa en las propiedades del proveedor de servicio subyacente. El proveedor del servicio subyacente puede, por ejemplo, perder, corromper o desordenar los mensajes, en cuyo caso el protocolo debe contener mecanismos de detección y corrección de errores, resincronización, retransmisión, etc., para proporcionar un servicio fiable y, en general más potente, a la capa superior siguiente.

Hay varias maneras posibles de modelar los conceptos arquitecturales de OSI en SDL que dependen, en primera instancia, de los aspectos que se desea realizar. Primero, se describe una técnica básica y luego, otras que son variantes de la técnica básica.

En los ejemplos se utiliza lo más posible la sintaxis gráfica del SDL. Obsérvese, sin embargo, que por razones prácticas, se puede omitir cierta información exigida por las reglas sintácticas o se representa mediante una serie de puntos (...) lo cual, naturalmente, no forma parte de la sintaxis.

#### I.2.1.2 Método básico

##### Especificación de servicio

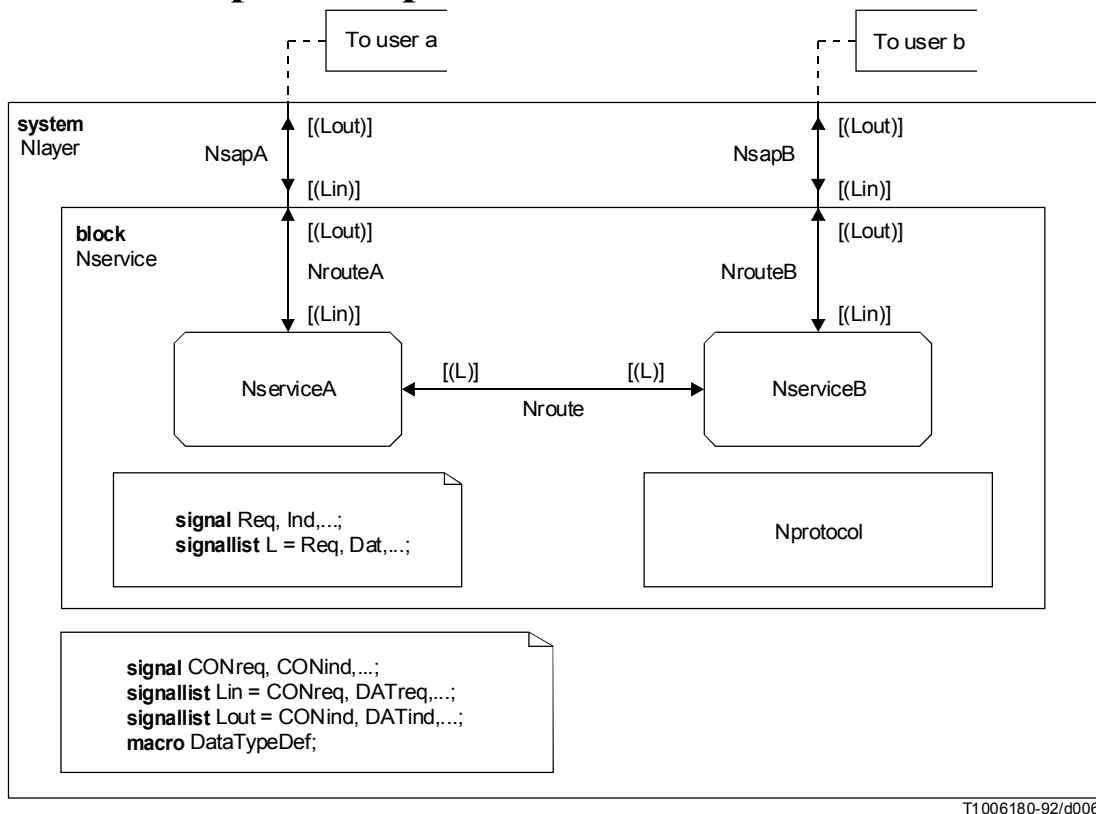
Una especificación de servicio para la capa N se puede modelar directamente como un bloque *Nservice* que contiene dos procesos: *NserviceA* y *NserviceB* (véase el Ejemplo I.2-1).

En el Ejemplo I.2-1, los usuarios de este servicio están en el entorno del sistema y pueden ser considerados como procesos capaces de comunicar con el sistema en términos de ese sistema.

Un punto de acceso al servicio está representado por un canal (*NsapA* o *NsapB*) que transporta señales que representan primitivas de servicio. Una señal puede transportar valores de los géneros (sorts) ofrecidos en la especificación de señal. Las especificaciones de género (diferentes de los géneros predefinidos) están contenidas en la especificación *macro* distante *DataTypeDef*, que se omite aquí porque no es pertinente para la explicación.

Va de suyo que, en los casos más generales, la especificación de servicio puede contener más de dos procesos, pero para abreviar sólo analizaremos dos procesos: uno para cada punto de acceso al servicio. No obstante, lo que sigue es aplicable también al caso en que haya más procesos en un punto de acceso al servicio.

# Reemplazada por una versión más reciente



EJEMPLO I.2-1

## Especificación de servicio (N) en SDL

En el Ejemplo I.2-1 se muestran algunos tipos de especificaciones de señal. Como lo sugieren algunos nombres de señal, se supone que se trata de un servicio con conexión. En el caso de un servicio sin conexión, se pueden hacer muchas simplificaciones, pero en aras de la brevedad, este caso no se analizará detalladamente.

En este ejemplo se trata de los aspectos local y de extremo a extremo de una especificación de servicio. El comportamiento local se expresa independientemente mediante los procesos *NserviceA* y *NserviceB*. Estos procesos se comunican entre sí mediante señales (*Req, Ind, ...*), que están dentro del bloque y son transportadas por la ruta de señal *Nroute*. El comportamiento de extremo a extremo se expresa mediante la correspondencia (ejecutada por cada proceso) entre las primitivas de servicio y las señales internas en *Nroute*. Los procesos *NserviceA* y *NserviceB* son imágenes especulares. El motivo de tener dos procesos y no uno es modelar lo más fielmente posible una probable situación de colisión en el proveedor de servicio.

El comportamiento no determinístico es una propiedad inherente del proveedor de servicio, porque puede rechazar las tentativas de conexión e interrumpir las conexiones establecidas por propia iniciativa.

Obsérvese que la especificación del bloque *Nservice* contiene únicamente una referencia a los procesos *NserviceA* y *NserviceB*; estos procesos están especificados por *especificaciones distantes*, situadas fuera de la especificación de bloque y no se muestran porque obligarían al lector a prestar atención a propiedades que son, por necesidad, específicas de un servicio determinado.

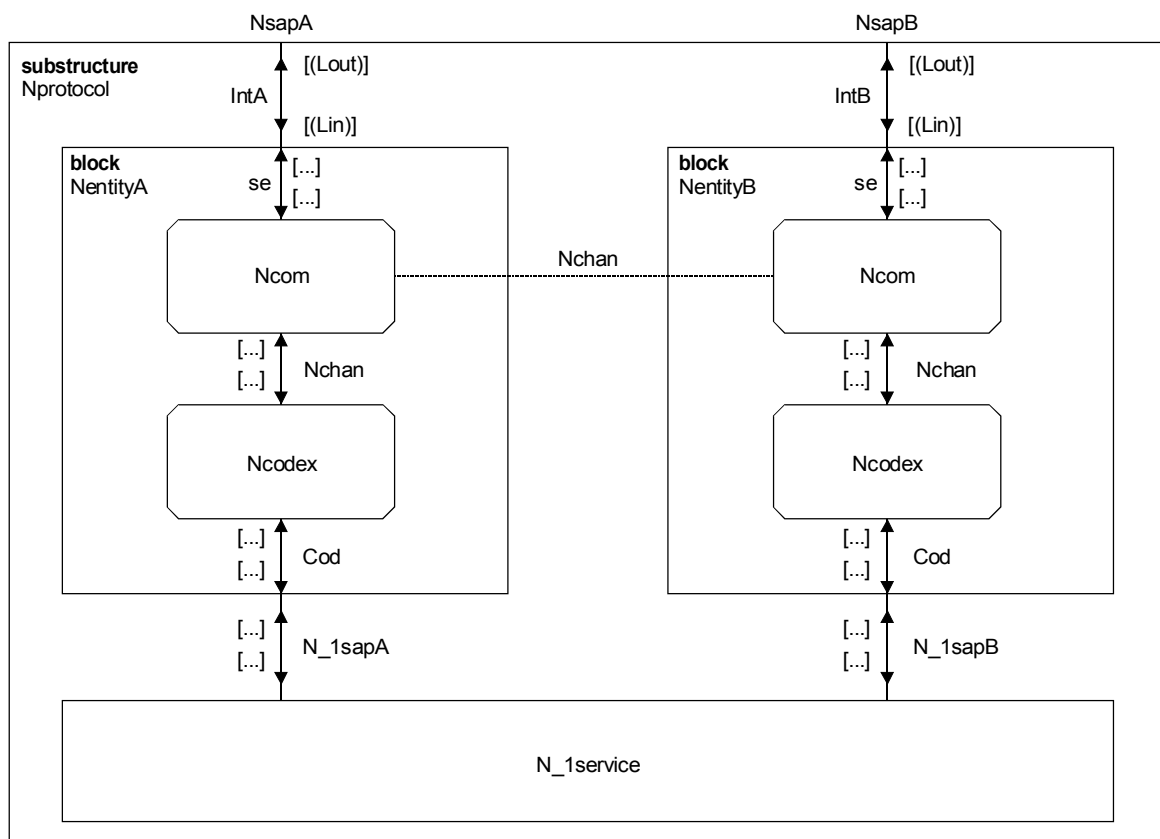
### Especificación de protocolo

La especificación de protocolo para la capa N es modelada por la subestructura *Nprotocol* del bloque *Nservice* (véase el Ejemplo I.2-1).

# Reemplazada por una versión más reciente

En el *diagrama de bloques* del Ejemplo I.2-1 se ha introducido una referencia de subestructura de bloque (un símbolo de bloque que contiene el nombre *Nprotocol* de la subestructura de bloque). La especificación de la subestructura de bloque se proporciona en un *diagrama de subestructura de bloque* distante (véase el Ejemplo I.2-2), que contiene tres bloques: *NentityA*, *NentityB* y *N\_1service*. Los primeros dos bloques representan entidades de protocolo (N), mientras que el bloque *N\_1service* representa el proveedor de servicio (N - 1). La especificación de *N\_1service* es análoga a la especificación de *Nservice* y no se muestra en este diagrama (por ser una especificación distante).

Un bloque de entidad de protocolo contiene uno o más procesos, según las características del protocolo. En este caso, se han elegido dos protocolos: *Ncom* y *Ncodex*. El proceso *Ncom* se ocupa del envío y la recepción de unidades de datos de protocolo, mientras que el proceso *Ncodex* se encarga de transmitir las unidades de datos de protocolo mediante el servicio subyacente. Desde el punto de vista conceptual, los procesos *Ncom* se comunican directamente mediante un canal implícito *Nchan* (que transporta unidades de datos de protocolo), pero en realidad se comunican indirectamente a través de los procesos *Ncodex* y el proveedor de servicio subyacente.



EJEMPLO I.2-2

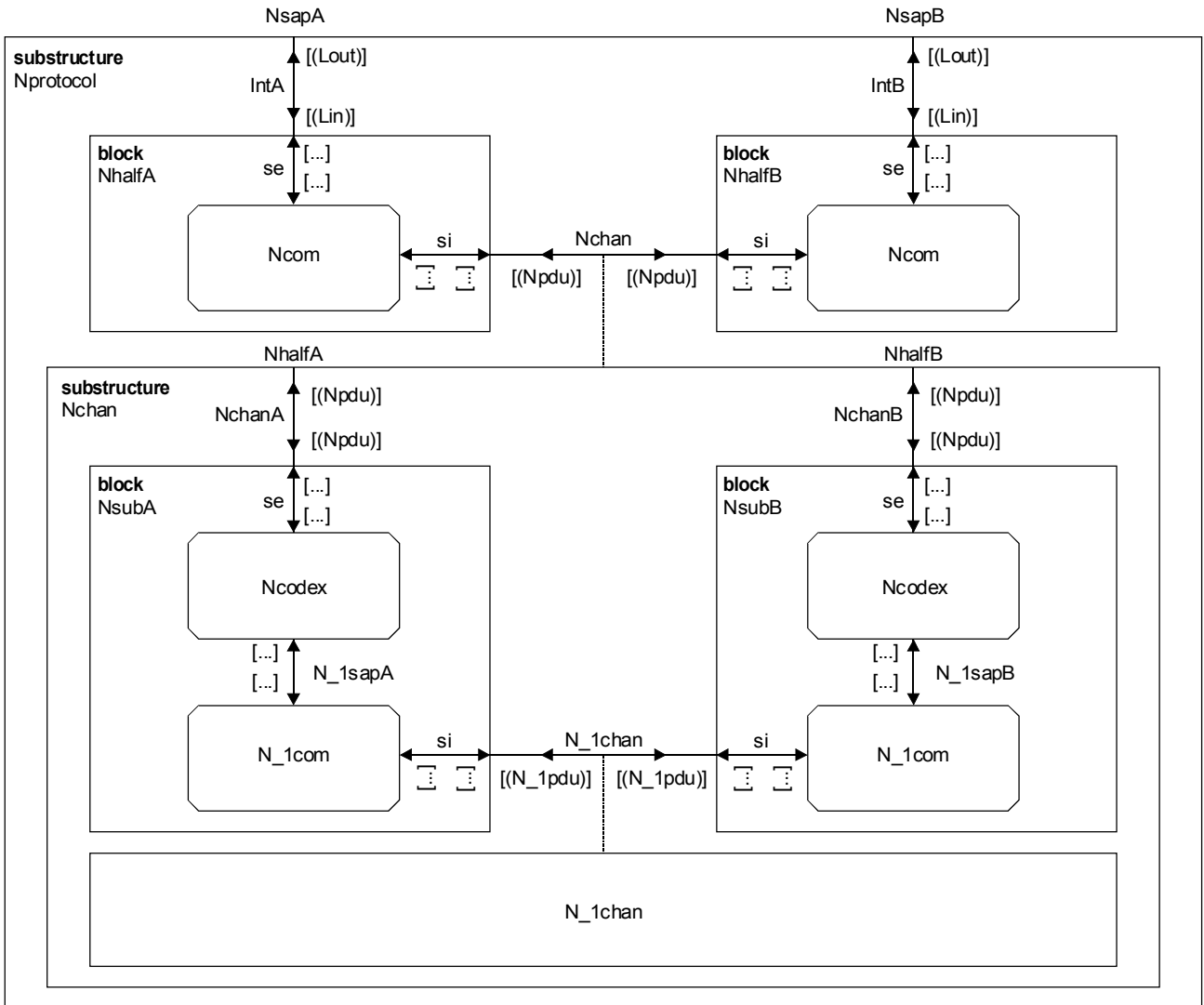
## Especificación de protocolo (N) en SDL

### I.2.1.3 Método alternativo que utiliza la subestructura de canal

Este método se establece a partir del método básico del Ejemplo I.2-2 agrupando los procesos de manera diferente, introduciendo el canal real *Nchan* y utilizando la subestructura de canal (véase el Ejemplo I.2-3). El canal *Nchan* transporta unidades de datos de protocolo indicadas por la lista de señales *Npdu*. Este método realiza la visión del protocolo y la orientación horizontal de OSI.

# Reemplazada por una versión más reciente

Obsérvese que en este método los bloques dentro de una subestructura de canal no representan entidades de protocolo y superponen dos capas adyacentes. Las primitivas de servicio están ocultas en estos bloques y son transportadas por las rutas  $N\_1sapA$ ,  $N\_1sapB$ ,  $N\_2sapA$  y  $N\_2sapB$ , etc. No obstante, la capa (N) más alta elegida se tratará por separado, como se indica en el ejemplo. Nótese también que el diagrama de sistema (véase el Ejemplo I.2-1) no es afectado por este método.



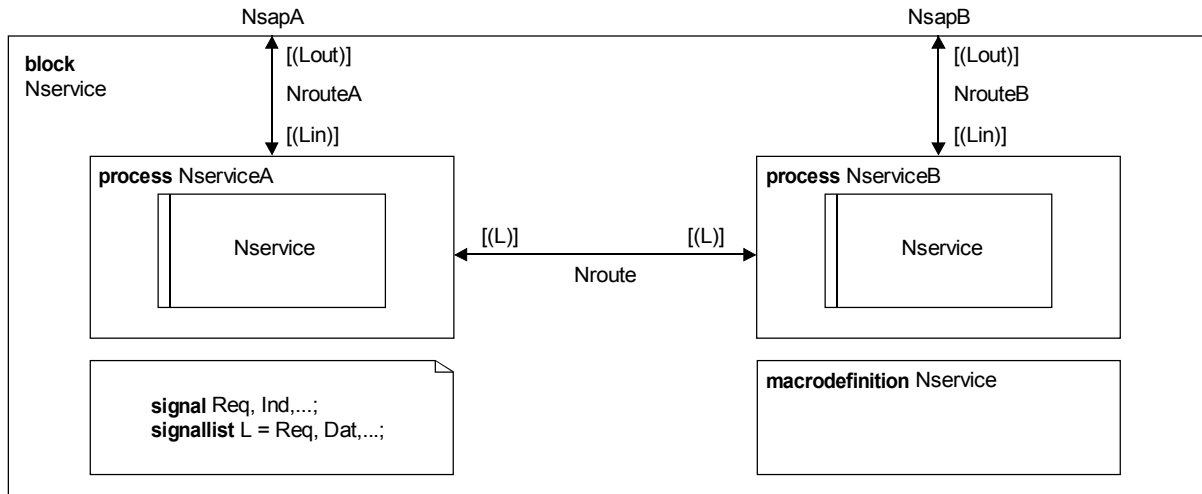
T1 006200-92/d008

EJEMPLO I.2-3  
Visión del protocolo OSI

## I.2.1.4 Arquitectura OSI simétrica

Cuando la arquitectura OSI es simétrica, es decir, cuando las entidades de los dos lados de la arquitectura OSI son imágenes especulares, las especificaciones de esas entidades son idénticas, excepto el nombre de la entidad. La especificación común sólo se puede proporcionar una vez mediante una macro (véase el Ejemplo I.2-4). En este Ejemplo, solamente se muestra el diagrama de bloques de  $Nservice$  del Ejemplo I.2-1. Téngase en cuenta que las especificaciones de servicio siempre son simétricas, y que sólo las especificaciones de protocolo pueden ser asimétricas.

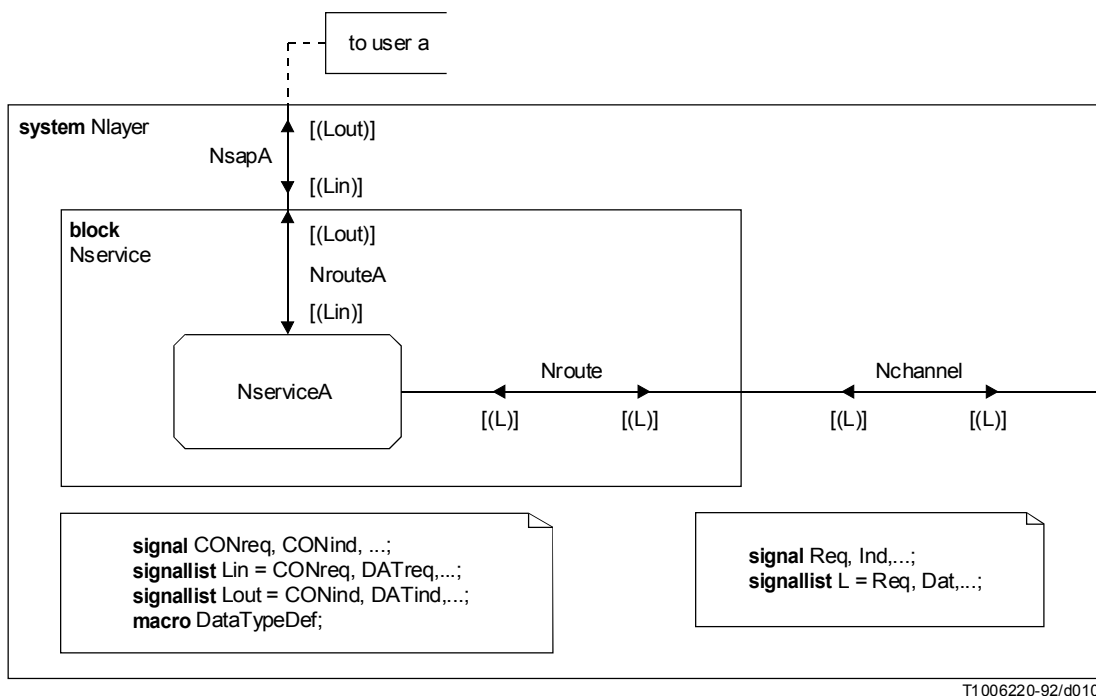
# Reemplazada por una versión más reciente



EJEMPLO I.2-4

## Utilización de macro para representar una arquitectura OSI simétrica

Un método alternativo consiste en representar sólo un lado (véase el Ejemplo I.2-5), que es una modificación del diagrama de sistema del Ejemplo I.2-1. Se ha reemplazado el canal *NsapB* por el canal *Nchannel*, que transporta las señales internas *L*. Estas señales se encuentran ahora en el nivel de sistema y sus especificaciones se han desplazado consecuentemente. Obsérvese que este método no se puede utilizar en combinación con la subestructuración de canal (mostrada en el Ejemplo I.2-3).



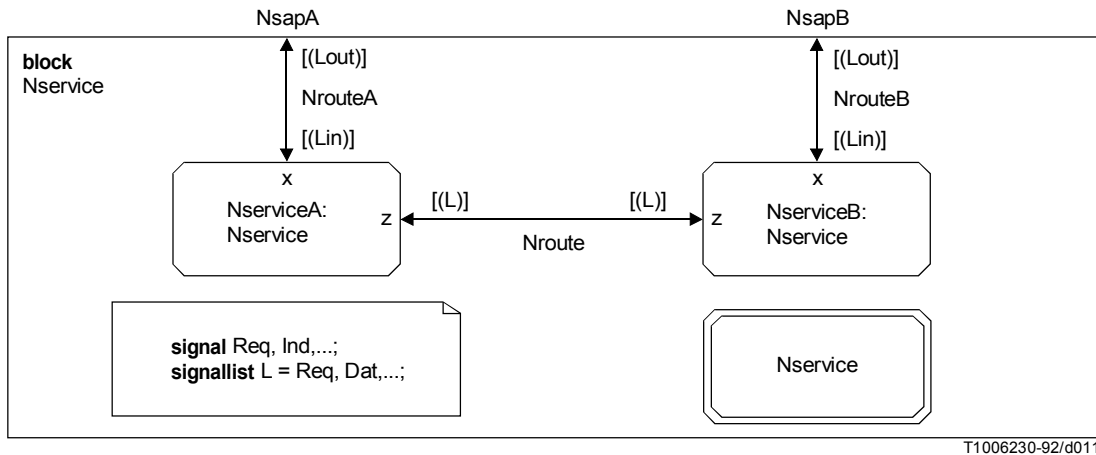
EJEMPLO I.2-5

## Representación de un solo lado de la arquitectura OSI simétrica



# Reemplazada por una versión más reciente

Un tercer método puede ser proporcionado por la instanciación de un tipo. Se modifica el Ejemplo I.2-4 y se introduce un tipo de proceso *Nservice* en el bloque *Nservice* que se instancia en *NserviceA* y *NserviceB* (véase el Ejemplo I.2-6). *x* y *z* son parámetros reales que corresponden a las puertas (gates) (no mostradas aquí) del tipo de proceso *Nservice*.



EJEMPLO I.2-6

Utilización de tipo de proceso para representar una arquitectura OSI simétrica

## I.2.2 Servicios RDSI según las Recomendaciones I.130 y Q.65

Los servicios de una RDSI se describen mediante la metodología establecida en la Recomendación I.130 [3]. Dicha metodología consta de tres etapas:

- Etapa 1: Aspectos de servicio.
- Etapa 2: Aspectos funcionales de la red.
- Etapa 3: Aspectos físicos de la red.

La etapa 2 se trata con más detalle en la Recomendación Q.65 [4], y consiste en los siguientes pasos para cada servicio:

- Paso 1: Modelo funcional – Identificación de las entidades funcionales y sus relaciones.
- Paso 2: Diagramas de flujo de información.
- Paso 3: Diagramas SDL de las entidades funcionales.
- Paso 4: Acciones de las entidades funcionales.
- Paso 5: Asignación de las entidades funcionales a emplazamientos físicos.

Hasta 1988, el SDL sólo se utilizaba en el paso 3; esta utilización del SDL se ha basado en la versión SDL80. En las nuevas Recomendaciones se prevé utilizar la versión SDL88.

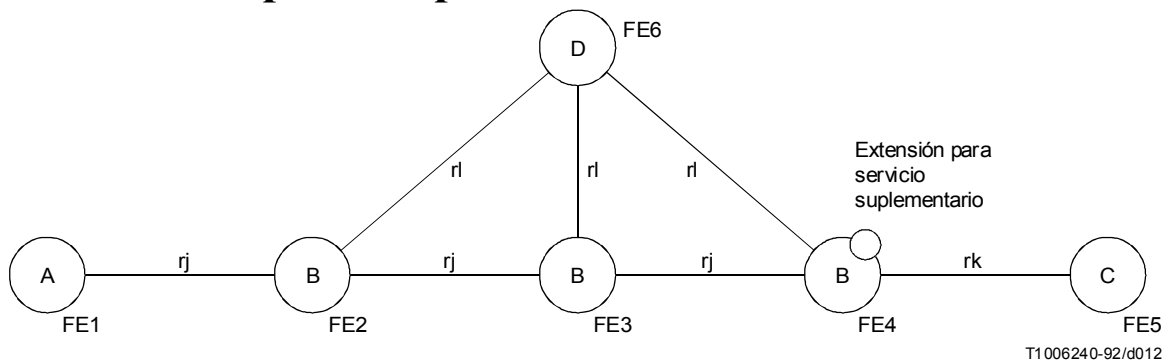
Idealmente, debería ser posible utilizar los conceptos estructurales del SDL en el paso 1 y los conceptos de comportamiento en los pasos 2 a 4 con la introducción gradual de datos. El paso 5 está fuera del alcance del SDL. La utilización del SDL en los pasos 1 a 4 de la Recomendación Q.65 se funda principalmente en que el cumplimiento de las normas aumenta la legibilidad y permite la sustentación de herramientas.

A continuación se muestra cómo se pueden hacer corresponder los conceptos de casi todos los pasos de la Recomendación Q.65 con el SDL.

### I.2.2.1 Estructura

Paso 1: El modelo funcional, la identificación de las entidades funcionales y sus relaciones concuerdan con la utilización de los conceptos estructurales en el SDL. En la Figura I.2-3 aparece un ejemplo de modelo funcional conforme a la Recomendación Q.65.

# Reemplazada por una versión más reciente



NOTA – Los elementos de la figura son:

- Nombres encerrados en un círculo (por ejemplo, *A*): Tipos de entidades funcionales.
- Nombres impresos en mayúscula junto a los círculos (por ejemplo, *FE1*): Nombres de entidades funcionales (*functional entities*) (instancias).
- Nombres impresos en minúscula entre los círculos (por ejemplo, *rj*): Relaciones entre los tipos de entidades funcionales.
- Círculo pequeño superpuesto a otro círculo: Extensión para servicio suplementario.

FIGURA I.2-3/Z.100

## Ejemplo de modelo funcional según la Recomendación Q.65

Obsérvese que este modelo distingue claramente entre los tipos de entidades funcionales (por ejemplo, *A*) y su instanciación (por ejemplo, *FE1*) al describir la estructura del sistema. En la Recomendación Q.65 se indica también que puede ser conveniente definir dos tipos de entidades funcionales como subconjuntos del mismo tipo de entidad funcional, si esos dos tipos tienen muchos puntos comunes. Estas características se pueden expresar mediante las extensiones orientadas a objetos de la versión SDL92, pero no con la versión SDL88.

El análisis ha demostrado que las «entidades funcionales» de la Recomendación Q.65 pueden hacerse corresponder con los conceptos estructurales del SDL como sigue:

- modelo → sistema;
- entidad funcional → bloque con un proceso;
- relación entre entidades funcionales → canal.

Además, cuando se utilizan constructivos orientados a objetos:

- tipo entidad funcional → tipo de bloque.

Existe una diferencia entre el modelo funcional de la Recomendación Q.65 y los diagramas de estructura SDL; estos últimos modelan normalmente sistemas abiertos.

En el Ejemplo I.2-7 se muestra el modelo funcional de la Figura I.2-3 en SDL. Obsérvese que *FE1* y *FE5* faltan en el sistema. La comunicación con estas entidades funcionales se ha modelado como comunicaciones con el entorno. El hecho de dejar *FE1* y *FE5* en el entorno significa que no se describirá su comportamiento.

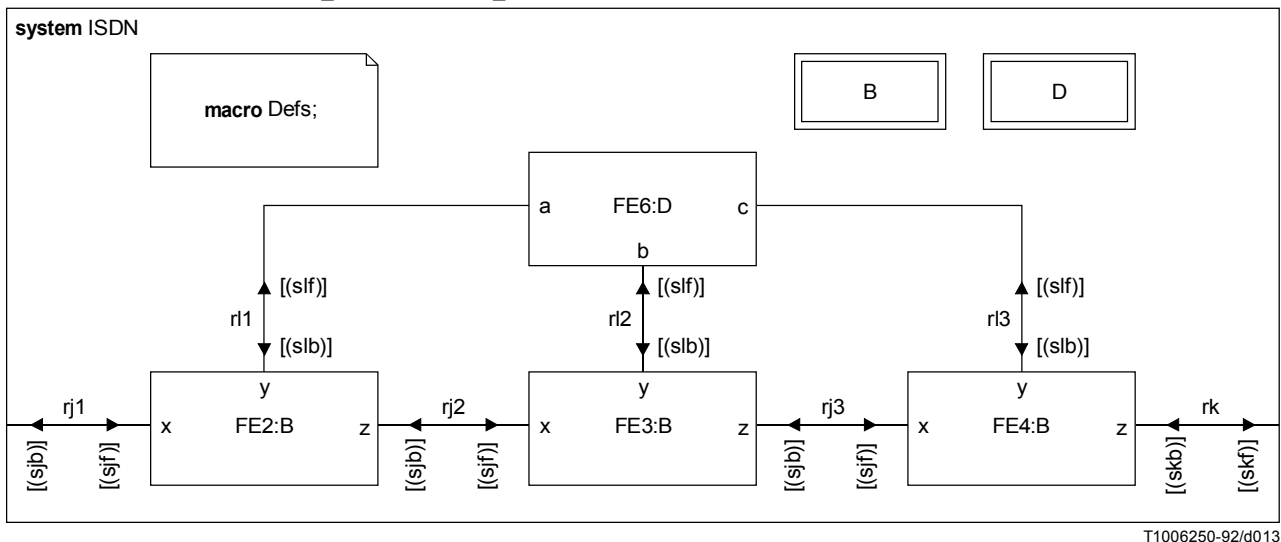
La distinción entre relación y tipo de relación se puede modelar mediante el uso de definiciones de listas de señales en los diagramas de estructura SDL. *B* y *D* son tipos de bloques en la versión SDL92.

Las principales ventajas que aportan los diagramas de estructura SDL consisten en que esos diagramas contienen definiciones de señales, tipos de datos, etc., que revisten importancia para los pasos subsiguientes de la Recomendación Q.65. En el Ejemplo I.2-7 esas definiciones se indican mediante la macro *Defs*.

### I.2.2.2 Comportamiento

El comportamiento se define en los pasos 2 a 4 de la Recomendación Q.65. La interacción entre entidades funcionales se describe en los diagramas de flujo de información a partir de los cuales se elabora un diagrama SDL (o sea, un diagrama de proceso) para cada tipo de entidad funcional.

# Reemplazada por una versión más reciente



EJEMPLO I.2-7

## Versión SDL del ejemplo de la Figura I.2-3

Los diagramas de flujo de información corresponden a los gráficos de secuencias de mensajes (MSC, *message sequence charts*) normalizados por el Grupo de Trabajo X/3 en la Recomendación Z.120 [5]. Los diagramas de flujo de información se elaboran para los casos «normales», y luego se utiliza el SDL para describir el comportamiento de las entidades funcionales.

En el paso 4 de la Recomendación Q.65 se definen varias acciones básicas para cada entidad funcional. La idea, que aún está en estudio, es reutilizar esas acciones básicas en diferentes especificaciones de servicio. Para este enfoque, el SDL ofrece dos conceptos: macro y procedimiento, es decir, cada acción básica puede ser «encapsulada» en una macro o un procedimiento. No se describirá con más detalle el paso 4.

### Diagramas de flujo de información

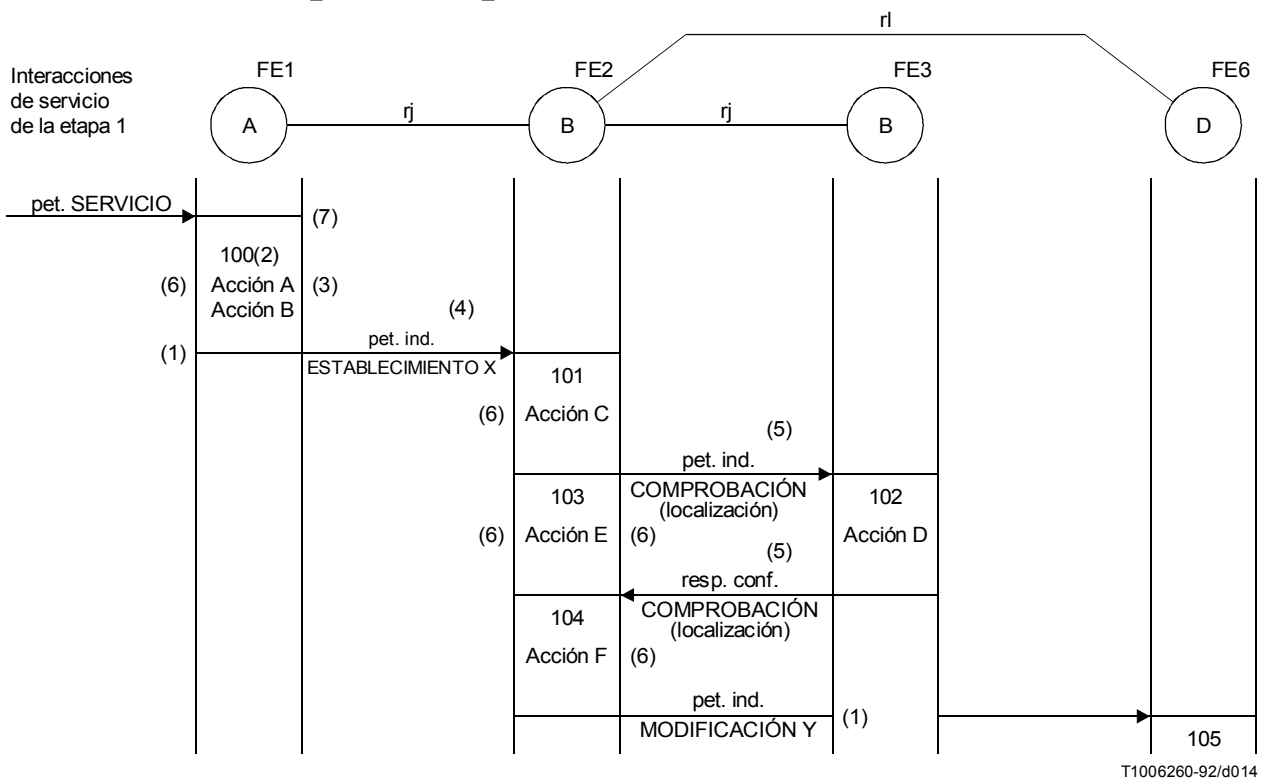
En la Figura I.2-4 se ofrece un ejemplo de diagrama de flujo de información.

En el Ejemplo I.2-8 se muestra el MSC que corresponde al diagrama de flujo de información de la Figura I.2-4. La correspondencia entre ambos es:

- nombre de entidad funcional → instancia de proceso (una por bloque);
- flujo de información → mensaje;
- información adicional → parámetros de mensaje;
- acción básica de entidad funcional → acción;
- etiquetas de diagramas SDL → comentarios.

Los diagramas de flujo de información indicados en la Recomendación Q.65 contienen básicamente la misma información que los MSC. En general, las especificaciones SDL se pueden comprobar con los MSC durante la simulación y, en algunos casos sencillos (cuando no hay creación dinámica de procesos o direccionamiento dinámico de salidas), la comprobación se puede hacer fácilmente sin simulación. Los MSC se pueden obtener de las especificaciones SDL, pero como normalmente en un MSC sólo se muestra un subconjunto del comportamiento («el comportamiento normal»), se precisa una interacción humana para obtener los MSC apropiados a partir de la especificación SDL. Otra posibilidad consiste en obtener automáticamente bosquejos de diagramas de proceso SDL a partir de los diagramas de flujo de información. Véase también el **paso 5** de I.3.

# Reemplazada por una versión más reciente



NOTA – Los elementos del diagrama son:

- Nombres que encabezan las columnas (por ejemplo, FE1): Entidades funcionales.
- Nombres en minúsculas entre círculos (por ejemplo, rj): Relaciones entre tipos de entidades funcionales.
- Nombres sobre las flechas entre columnas (por ejemplo, pet. ind. ESTABLECIMIENTO X): Flujo de información.
- Nombres entre paréntesis junto a nombres de flujo de información (por ejemplo, localización): Información adicional transmitida por el flujo de información.
- Nombres dentro de columnas (por ejemplo, Acción B, 100): Nombre de acción básica de entidad funcional.
- Números entre paréntesis [por ejemplo, (5)]: Etiquetas del diagrama SDL.

FIGURA I.2-4/Z.100

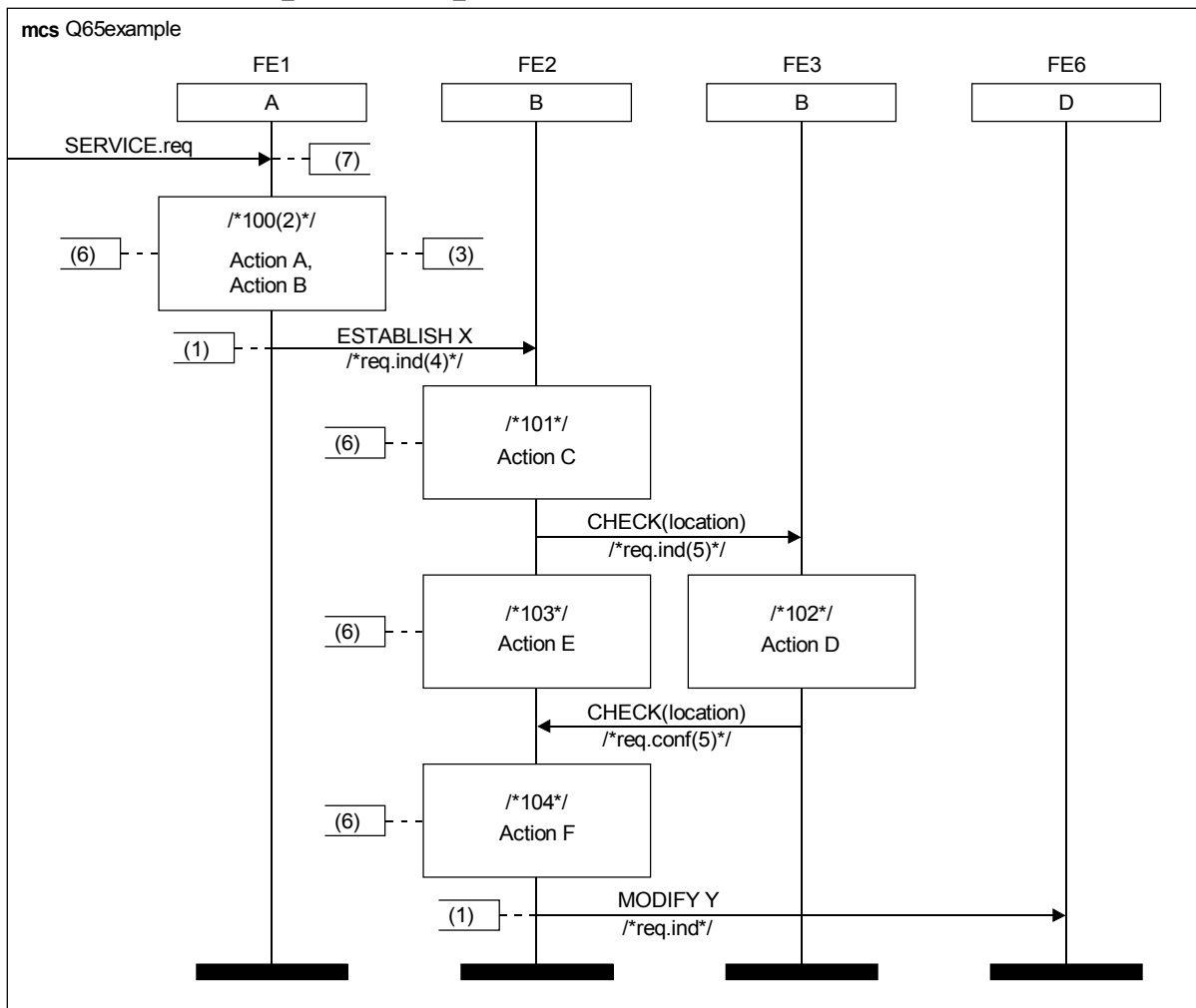
## Ejemplo de diagrama de flujo de información de la Recomendación Q.65

### Diagramas de proceso

Esta es la parte del SDL que se ha utilizado en las Recomendaciones sobre las RDSI. Se ha trabajado intensamente para actualizar los diagramas SDL hasta la versión SDL88. A continuación se ofrecen algunos comentarios sobre esta actualización. Desde el comienzo, todos los textos de los símbolos eran informales. Fue relativamente fácil llegar al convencimiento de que hacía falta formalizar los símbolos en **entrada** y **salida**, puesto que se refieren a las interfaces de los procesos (o sea, el paso precedente en la metodología establecida). Para muchos resultó (y resulta) molesto tener que poner entre comillas simples ( ' ') el texto informal de **tarea** (task) y **decisión** (decision) ya que el contenido de esas casillas es, en su mayor parte, texto informal. En el Ejemplo I.2-9 aparecen algunos ejemplos. Se han hecho oficiosamente algunas sugerencias creativas para definir una regla que distinga entre texto formal e informal, que sea más «fácil para el usuario» y menos «fácil para la herramienta». Sin embargo, parece difícil formular reglas simples e inequívocas para el texto informal, sin invalidar el uso actual de *Charstring* como texto informal.

Otro problema ha sido la costumbre de asignar secuencias descriptivas de nombres a un estado o a una señal. La primera solución propuesta por la Comisión de Estudio X consistió en utilizar caracteres con rayas para enlazar varias palabras y formar una sola, por ejemplo «hang\_up\_req.ind» en vez de «hang up req.ind». Después se volvieron a formular las reglas léxicas de SDL88, y «hang up req.ind» se convirtió en un nombre válido en el SDL. Actualmente, la utilización de los nombres en SDL88 concuerda con la utilización establecida de los nombres.

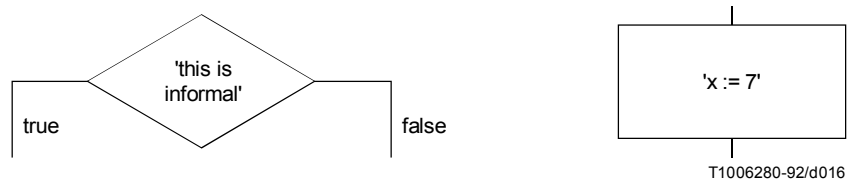
# Reemplazada por una versión más reciente



T1006270-92/d015

EJEMPLO I.2-8

Versión MSC del ejemplo de la Figura I.2-4



T1006280-92/d016

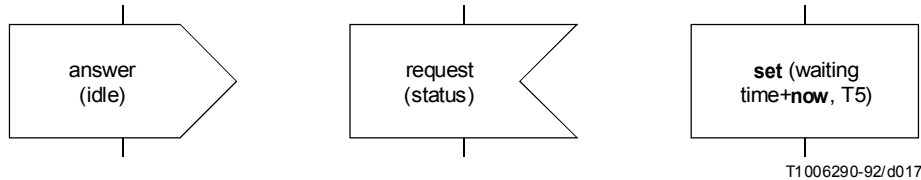
EJEMPLO I.2-9

Texto informal

# Reemplazada por una versión más reciente

## I.2.2.3 Datos

**Tarea** (task) y **decisión** (decision) son, principalmente, informales en los diagramas SDL existentes para la RDSI. Por ello, no es muy necesario definir operadores para tipos de datos. Los datos aparecen principalmente en los constructivos **input** (entrada), **output** (salida) y **timer** (temporizador), véase el Ejemplo I.2-10.



EJEMPLO I.2-10

### Utilización de los datos

En la mayoría de los casos, *Boolean* e *Integer* bastan, por ejemplo, para transportar el valor de alguna bandera o contador. También se utiliza ampliamente *Charstring*, porque permite tratar los parámetros de manera semejante al texto informal.

Podría resultar útil emplear tipos enumerados. Un **newtype** que contiene sólo literales concuerda con un tipo enumerado de datos (como, por ejemplo, en Pascal):

```
newtype Indication
literals idle, busy, congestion;
endnewtype Indication;
...
signal Check location (Indication) /*resp.conf */;
...
output Check location (busy) /*resp.conf*/;
...
```

Los datos se introducen en los pasos 1 a 4, al formalizar las especificaciones, por ejemplo, las especificaciones de señal se aplican a tipos de datos. Los datos se pueden introducir en diferentes niveles de refinamiento. Introducir datos resulta ventajoso porque se puede comprobar si los parámetros de entrada y salida, las preguntas y respuestas, etc. concuerdan en toda la especificación SDL.

## I.3 Elaboración de una especificación SDL por pasos

A continuación se ofrece un método para elaborar una especificación SDL, pero ello no entraña que el CCITT recomiende este método en particular, ya que se reconoce que una especificación SDL se puede formular con muchos métodos diferentes. Se puede considerar este método como una directriz para elaborar una especificación SDL mediante pasos bien definidos aunque los resultados de cada paso pueden no ser una especificación formal y completa. Las adaptaciones del método pueden consistir en una ordenación diferente de algunos pasos, etc.

Los pasos se identifican desde el punto de vista del usuario y su formalismo se indica mediante los constructivos de lenguaje introducidos. Un ejemplo sencillo ilustra el método.

El método que se presenta sólo introduce un subconjunto del SDL necesario para el ejemplo simple utilizado. Un método más elaborado incluiría más conceptos del lenguaje, por ejemplo, *servicios* para reducir la complejidad de los *procesos*.

# Reemplazada por una versión más reciente

## I.3.1 Introducción

La elaboración de una especificación formal comienza, en general, con una especificación informal e incompleta. El proceso de elaboración termina cuando la especificación es formal y completa.

Por «formal» se entiende «que cumple las reglas del lenguaje formal», es decir, una especificación SDL formal cumple la sintaxis concreta y la semántica estática del SDL. Además, no contiene *texto informal*.

Por «completa» se entiende «que transmite toda la información sobre el sistema de la manera prevista por el autor».

Se estima que la elaboración de una especificación completa es una tarea difícil, especialmente para aquéllos con pocos conocimientos en este campo. La capacitación en técnicas de descripción formal es bastante diferente de la que se necesita para los lenguajes de programación, por poner un ejemplo, y es preciso tener esto en cuenta en el apoyo metodológico para la elaboración de especificaciones formales.

Dado que cada etapa sigue muy de cerca las reglas SDL, es posible utilizar en gran medida las herramientas especializadas del SDL.

Se pueden formular especificaciones SDL para:

- la estructura;
- el comportamiento;
- los datos.

Es obvio que una especificación no estará completa hasta que no se traten los tres aspectos por entero. Sin embargo, antes de llegar a esa etapa se puede obtener una especificación SDL formalmente válida.

Se prevé tratar los tres aspectos simultáneamente. En la definición de los pasos se han aplicado los principios siguientes, por los que cada uno:

- es una interrupción natural del proceso de elaboración;
- debe producir un resultado autónomo, que parece tener sentido y puede ser verificado, de preferencia con herramientas;
- no depende (o depende poco) de los pasos siguientes.

Cada paso se describe mediante:

- **Descripción** – El proceso conceptual del paso. Se formula mediante conceptos generales y los conceptos SDL correspondientes (indicados en *cursiva*). En la mayor medida posible, se ofrecen procedimientos heurísticos. Se divide en un resumen de acciones y una parte de análisis.
- **Resultado** – El resultado de la ejecución del paso.
- **Ejemplo** – El paso aplicado a la especificación de un sistema de ascensor pequeño (la especificación completa aparece en I.3.3). Se ha suprimido en algunos pasos por economía de espacio.

## I.3.2 Los pasos

### Paso 1 – Frontera del sistema

#### Descripción

- Delimitar el *sistema* con respecto a su entorno. Elegir un nombre adecuado para el *sistema*. Identificar a los agentes en el entorno del *sistema* (o sea, las entidades diferentes con las que interactuará el *sistema*). Describir informalmente el propósito y las características del *sistema* en un *comentario*.
- Especificar un *canal* para cada agente identificado en el entorno y asignar un nombre adecuado correspondiente al nombre del agente.
- Introducir un *bloque* ficticio dentro del *sistema*. Este *bloque* será reemplazado más adelante por la estructura efectiva del sistema.

## Reemplazada por una versión más reciente

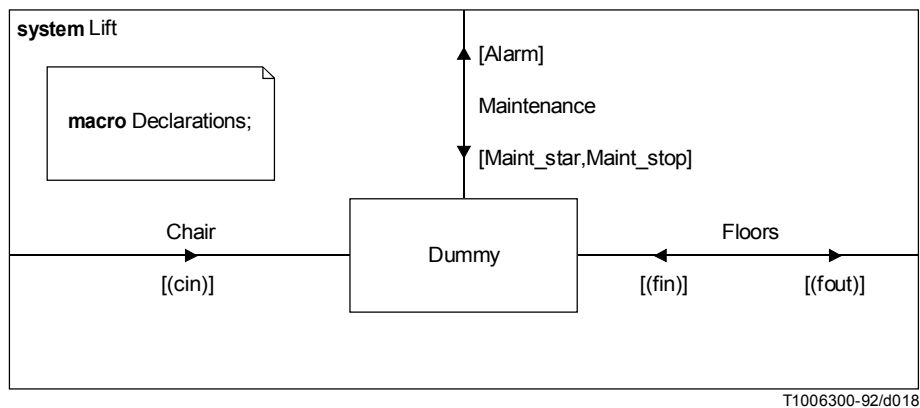
- Identificar el flujo de información, en términos de eventos discretos, a lo largo de la frontera del *sistema*. Estos eventos constituyen el alfabeto del sistema y son modelados por *señales*. Las *señales* para comunicaciones externas se especifican en el nivel de *sistemas*. Enunciar el propósito de cada *señal* en un comentario para cada *especificación de señal*. Identificar la información que transmitirán las *señales* e indicar el *género (sort)* de *parámetros de señal*. Utilizar lo más posible *syntype* y *sorts* enumerados y predefinidos. Asociar *señales* con *canales*, directamente o mediante *listas de señales*.
- Proporcionar un esquema de especificación (sin *signatura*) para cada *género (sort)* nuevo introducido.

Hay que tener en cuenta que la cantidad de *señales* se puede reducir calificándolas con *parámetros de señal*.

Es evidente que este paso es una gran simplificación de los considerables esfuerzos que hay que desplegar en las fases preliminares de determinación de los requisitos, análisis de sistemas, etc. Como preludio a este paso se pueden utilizar otras técnicas diferentes del SDL. El análisis orientado a objetos se trata con más detenimiento en el § I.4.

**Resultado** – La especificación de la frontera de un *sistema*, que no tiene definida su estructura interna.

### Ejemplo



```
macrodefinition Declarations;
```

```
/* A lift consists of a chair with GOTO(floor) buttons and floor controls where at each floor the users can ask for upwards or downwards service.
```

```
This specification does not model the mechanical part of the lift.
```

```
The service order is: If there are more floors to be served in the current direction, then continue to the next of these floors. If there is any floor to be served in opposite direction, then change direction. If there are no floors waiting to be served, stay at the current floor.
```

```
A user can control the lift by:
```

- pressing a button at any floor for service in the desired direction;
- pressing a button in the lift chair for a certain floor.

```
In addition, the specification also covers some unusual cases, such as start and stop of the operation of the lift. */
```



# Reemplazada por una versión más reciente

```
block Dummy;
/* minimal block syntax */
process Dummy; start; stop; endprocess;
endblock;
signal
  Goto(Floor),           /* request to go to the given destination */
  Floor_req(Direction,Floor), /* request for the lift from a floor */
  Open_door(Floor),     /* open door at floor */
  .....
signallist cin = Goto, .....;
signallist fin = Floor_req, .....;
signallist fout = Open_door, .....;
syntype Floor
.....
endsyntype;
newtype Direction
.....
endnewtype;

endmacro Declarations;
```

## Paso 2 – Estructura del sistema

### Descripción

- Identificar los principales constituyentes conceptuales del sistema y darles un nombre. Estos son los *bloques de sistema*. Elegir un nombre adecuado para cada *bloque* y describir el *bloque* y la relación con su entorno (su estructura de delimitación) informalmente en un *comentario* incluido en la *especificación de bloque*.
- Conectar *bloques* con la frontera de *sistema* mediante *canales*, acción ya introducida en el **paso 1**.
- Identificar el flujo de información (*canales* y *señales* asociadas) entre los *bloques*. Especificar las nuevas *señales* introducidas en el nivel de sistema, como en el **paso 1**.
- Proporcionar un esquema de especificación para cada nuevo *género* (sort) introducido en el nivel de sistema, como en el **paso 1**.

Se aconseja no tener muchos *bloques* encerrados en el mismo recuadro. Si su número es demasiado grande, es decir, que impide la visión general, se aplicará la anidación (nesting) examinada en el **paso 3**. Conforme a la heurística: un *bloque* debe tener una fuerte cohesión interna y ser fácilmente comprensible por sí mismo. Un bloque debe corresponder a un concepto, por ejemplo, «central local», «control del ascensor».

Las propiedades que habrá que tener en cuenta al identificar los *bloques* son:

- un *bloque* delimita la visibilidad: dentro de un *bloque* se pueden especificar *señales* y *géneros*;
- es posible que la comunicación entre *bloques* (o sea, los *canales*) entrañe retardos.

Se considera que un *bloque* es un sistema en sí mismo: define un alfabeto externo para el subsistema dentro del bloque y la frontera con su entorno.

Un bloque se puede especificar directamente como una instancia de bloque. No obstante, el SDL orientado a objetos distingue claramente entre *tipo* e *instancia*. Si se va a utilizar un bloque u otro bloque similar a ese bloque en alguna otra parte del sistema, se deberán identificar primero las características más básicas de los bloques semejantes en un tipo de bloque. Los bloques reales pueden instanciarse a partir del tipo general o especializaciones del tipo general. Esta observación es una simplificación de los métodos orientados a objetos, que preconizan la definición de conceptos mediante *jerarquías de tipos*, etc., véase más sobre este tema en I.4.

Un *canal* se puede asociar con un retardo de transmisión, con lo que se consiguen efectos especiales de modelado al tener varios *canales* entre dos constituyentes. Normalmente, dos *bloques* dentro de *sistema* sólo necesitan interconectarse por un *canal*, pero el modelado con más *canales* puede resultar útil en casos especiales.

En este paso ya se puede considerar la utilización de gráficos de secuencias de mensajes (MSC) para tener una visión global útil de los escenarios de comunicación típicos entre constituyentes de sistema (el uso de MSC se trata en el **paso 5** y con más detalle en I.6).

# Reemplazada por una versión más reciente

**Resultado** – Identificación de *bloques* en el nivel de *sistema*.

**Ejemplo** – En este ejemplo sólo se necesita un bloque, que se especifica directamente.

```
block Control;  
  
/* The block shall describe the controls of the complete lift system */  
  
/* minimal process syntax */  
  
process Dummy; start; stop; endprocess;  
  
endblock;
```

## Paso 3 – Partición de bloque

### Descripción

- Dividir cada *bloque* complejo en *sub-bloques* (esto es igual al **paso 2** del *sistema*).
- Repetir esta acción hasta que no queden *bloques* complejos.

Si el sistema es grande, algunos *bloques* se pueden considerar sistemas por sí mismos, y se pueden dividir conforme a las reglas establecidas para *sistema*. Ello resulta en el anidamiento de *bloques* para conseguir una mejor visión general gracias al ocultamiento de cierta información. En la estructura resultante, cada *bloque no dividido* reserva un lugar para la especificación de comportamiento. (El comportamiento se describe en etapas sucesivas mediante un número de *procesos* dentro de cada *bloque*.)

**Resultado** – Un árbol de *bloques* cuya raíz es el *sistema* y cuyas hojas son los *bloques no divididos*.

**Ejemplo** – Ninguno, puesto que el sistema *ascensor (Lift)* es tan pequeño que no es necesario dividir los bloques.

## Paso 4 – Constituyentes de bloque

### Descripción

- Identificar las actividades dentro de cada bloque no dividido. Estos son los *conjuntos de procesos* del *bloque*. Elegir un nombre adecuado para cada *conjunto de procesos*, especificar el conjunto y su relación con el entorno (el recuadro del *bloque*) informalmente en un *comentario* dentro de la *especificación de proceso*.
- Conectar *conjuntos de procesos* con *canales* en la frontera del *bloque* mediante *rutras de señal*.
- Identificar el flujo de información (*rutras de señal* y *señales* asociadas) entre *conjuntos de procesos*. Especificar las nuevas *señales* introducidas, como en el **paso 1**.
- Proporcionar un esquema de especificación para cada *género* nuevo introducido, como en el **paso 1**.

Especificar para cada conjunto de procesos el *número de instancias*, el *número inicial* y *máximo* de instancias. Para cada *bloque*, por lo menos un *conjunto de procesos* debe tener *número inicial* superior a cero. Puede ser útil indicar el *número máximo* si cada *instancia de proceso* corresponde a un recurso limitado (por ejemplo, algún dispositivo físico). Las consideraciones relativas a la identificación de *tipo* expuestas en el **paso 2** se aplican también a este paso para *tipos de procesos*.

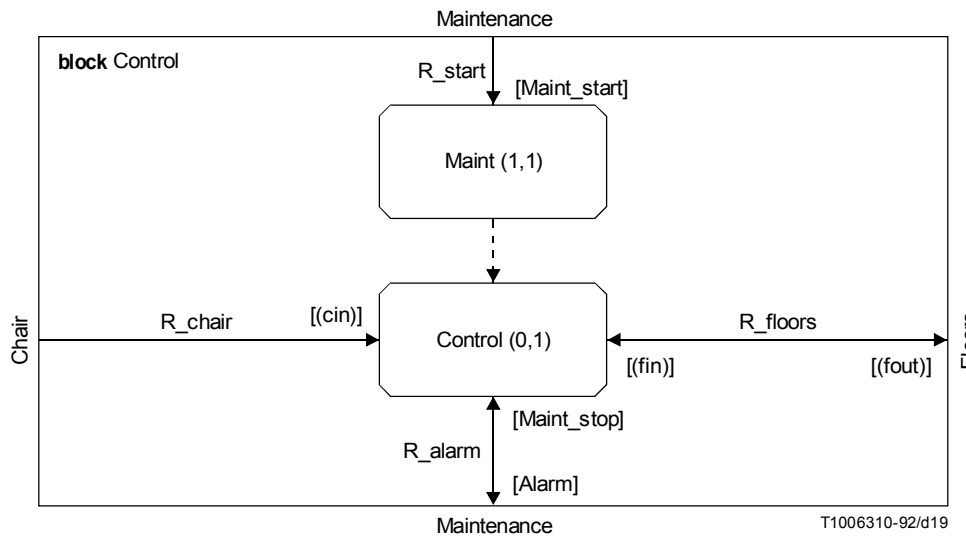
Las directrices para identificar los *conjuntos de procesos* son: cada *conjunto de procesos* representa un patrón de actividad, y este patrón puede existir en varias instancias concurrentes. La mayor parte de la heurística del **paso 2** se aplica también a este paso.

El alfabeto de entrada de un *conjunto de procesos* puede especificarse mediante *rutras de señal* o *conjunto de señales*. *Rutra de señal* es un trayecto de comunicación entre *conjuntos de procesos* dentro de un *bloque* o entre *conjuntos de procesos* y lo que está alrededor de un *bloque* (compárese con *canal*, que es un trayecto de comunicación entre *bloques* y lo que está a su alrededor).

# Reemplazada por una versión más reciente

**Resultado** – Identificación de los *conjuntos de procesos* dentro de *bloques* no divididos.

## Ejemplo



```
process Control;  
/* controls the operation of the lift */  
start; stop; /* minimal process syntax */  
endprocess;  
process Maint;  
/* initializes the operation of the lift */  
start; stop; /* minimal process syntax */  
endprocess;
```

## Paso 5 – Especificaciones de proceso esquemáticas

### Descripción

- Identificar los casos de utilización típicos y describirlos, por ejemplo, mediante gráficos de secuencias de mensajes (MSC).
- Tomar las decisiones adicionales necesarias relativas al modelado del comportamiento. Esto puede requerir la introducción de *señales* y *géneros* nuevos, que se especifican como en el **paso 1**.
- Escribir una *especificación de proceso* esquemática que abarque esos casos, pero que todavía no considere su combinación.
- Considerar la utilización de *procedimientos* para ocultar detalles y de *temporizadores* para supervisar la temporización. Introducir *sinónimos externos* para los valores no especificados.

El orden de los eventos en un eje vertical (véase el ejemplo siguiente) en un MSC proporciona una ordenación de eventos en una especificación de proceso. Esta ordenación constituye (una parte de) una *especificación de proceso esquemática*. Los MSC se pueden utilizar también en pasos anteriores para especificar la comunicación dentro del *sistema*. En este caso, un eje de un MSC puede denotar todo un *bloque*. El empleo del MSC se trata con más detenimiento en I.6.

Empezando por el símbolo de comienzo de cada *especificación de proceso*, construya un árbol de *estados* teniendo en cuenta el desarrollo «normal» de *especificación de proceso*. Si es necesario, introduzca la *creación* dinámica de *instancias de proceso*, pero los *parámetros* no se incluyen aún en *creación de proceso*. Indique los *parámetros* en *entradas* y *salidas*.

# Reemplazada por una versión más reciente

La supervisión temporal también se puede mostrar en un MSC. Según la heurística: la supervisión temporal se utiliza para modelar un periodo de tiempo dentro del modelo, supervisar la liberación de un recurso o supervisar las respuestas procedentes de fuentes no fiables (por ejemplo, otro nodo de red). La supervisión temporal se logra mediante la introducción de *temporizadores* y acciones de *fijación (set)* y *reiniciación (reset)*.

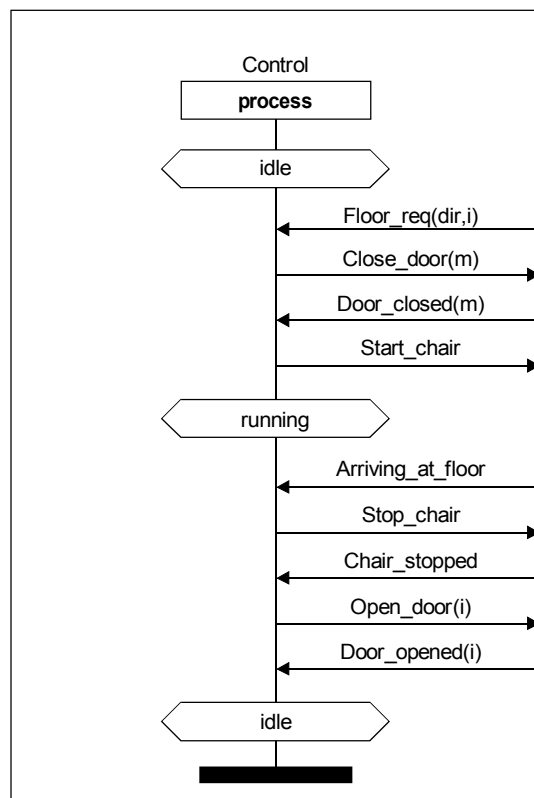
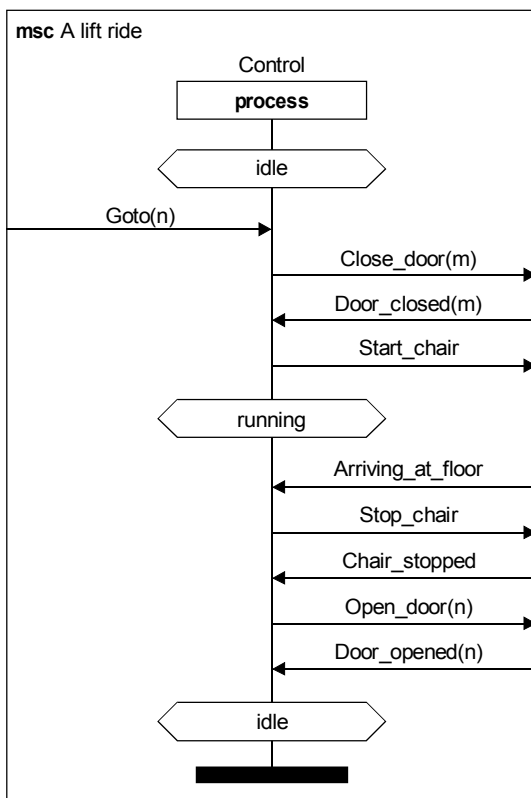
Si no se especifican algunos valores (por ejemplo, el piso más alto en el ejemplo del ascensor), el sistema descrito es *genérico*. Una manera de expresar esto es mediante *sinónimos externos*. Es importante determinar qué partes de la especificación deben ser genéricas para aumentar la utilidad de una especificación (por ejemplo, una especificación en SDL de un sistema de ascensor para un edificio con cualquier número de pisos es más útil que la especificación para un edificio, digamos, de 14 pisos).

**Resultado** – MSC y especificaciones de proceso esquemáticas.

**Ejemplo** – Es preciso tomar algunas decisiones sobre el modelado del comportamiento, por ejemplo, cómo identificar la llegada del ascensor a un piso, lo que se consigue por la recepción de una señal *Arriving\_at\_floor* desde el entorno.

En el MSC A, el *viaje en ascensor (lift ride)* describe la siguiente situación simple: un usuario que está en el *m-ésimo* piso entra en el ascensor (que lo esperaba en ese piso), pulsa el botón *goto(n)* y es transportado al *n-ésimo* piso, y la puerta del ascensor se abre al llegar a ese piso.

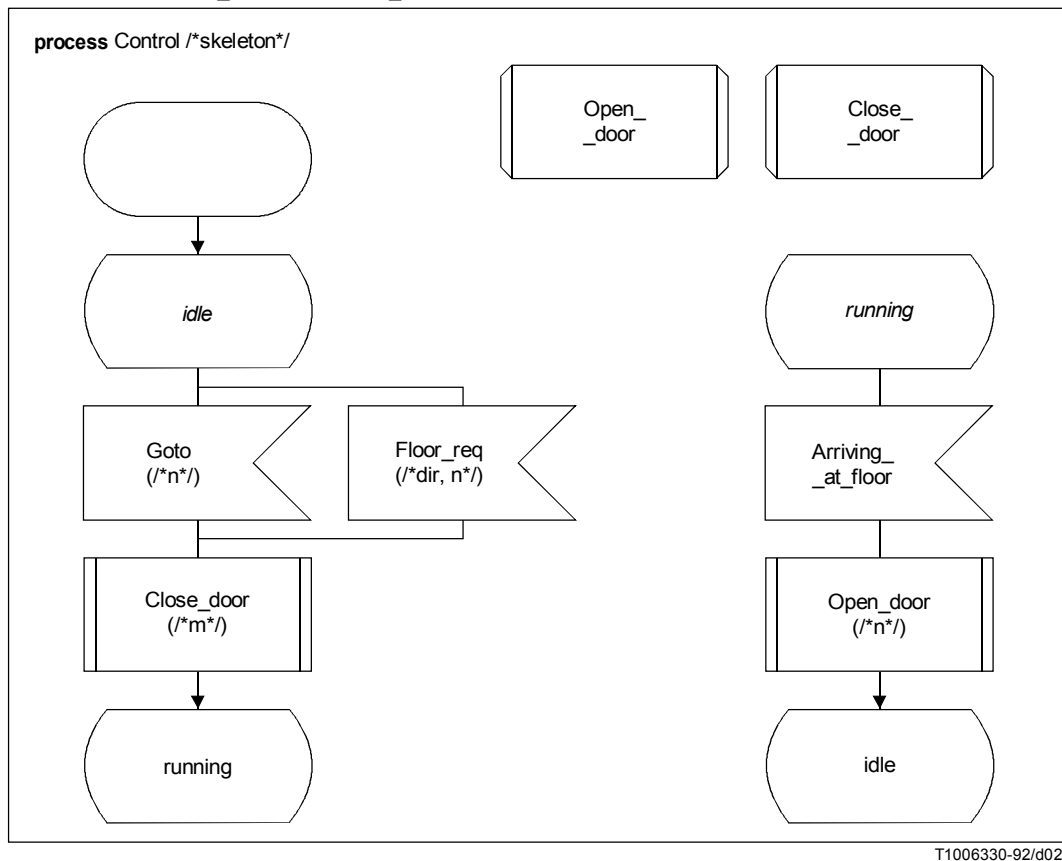
Si el ascensor está detenido en otro piso, el usuario debe llamarlo. Esta situación se describe en el MSC con *llamada del ascensor (lift request)* (el usuario se encuentra en el *i-ésimo* piso).



T1006320-92/d020

La *especificación de proceso* esquemática trata este caso de un usuario, y se obtiene a partir de los MSC mostrados más arriba (nótese que el piso *i* se trata de la misma manera que el piso *n*). Algunos detalles de señalización quedan ocultos en la especificación de proceso al utilizar los procedimientos *Open\_door* y *Close\_door* (*abrir puerta y cerrar puerta*).

## Reemplazada por una versión más reciente



### Paso 6 – Especificaciones de proceso informales

#### Descripción

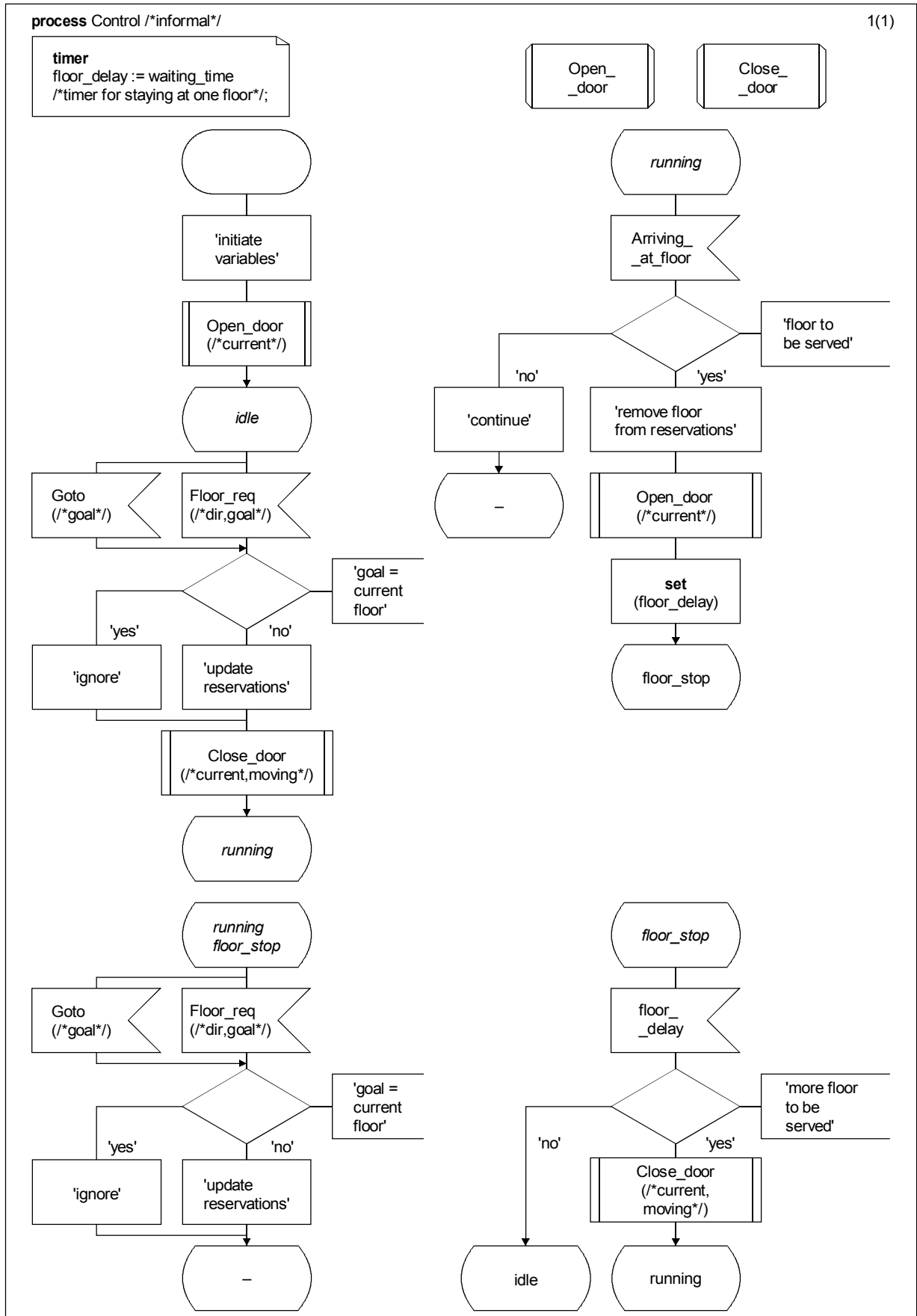
- Considerar la combinación de casos de utilización y describirla, si procede, mediante MSC.
- Identificar la información que hace falta almacenar en *procesos*.
- Introducir *tareas* y *decisiones*, pero sólo utilizar en ellas *texto informal*. Introducir posibles *estados* nuevos.
- Escribir una especificación esquemática para cada *procedimiento* e indicar *género de parámetros de procedimiento*.
- Indicar *género de parámetros de procedimiento formales*.
- Especificar cada *género* nuevo introducido, como en el **paso 1**.

La elección entre *entradas* diferentes se hace mediante un *estado*. La elección entre *salidas* se logra mediante una *decisión informal*. La elección entre *entradas* y *salidas* se efectúa mediante una combinación de *decisión informal* y *estado*.

**Resultado** – *Especificaciones de proceso informales*.

**Ejemplo** – Cuando se trata de varios usuarios, se puede recibir *Goto* y *Floor\_req* mientras el ascensor está funcionando y esas peticiones se deben almacenar en cuadro de reservas. Se introduce un nuevo estado *floor\_stop* para una parada provisional en un piso. La duración de una parada provisional está determinada por el temporizador *floor\_delay*.

# Reemplazada por una versión más reciente



T1006340-92/d022

# Reemplazada por una versión más reciente

El caso de múltiples usuarios también se puede describir mediante MSC (véase I.9.1.2).

```
macrodefinition Declarations;
.....
synonym waiting_time Duration = external;
    /* duration of stay at one floor */

endmacro Declarations;
```

## Paso 7 – Especificaciones de proceso completas

### Descripción

- Considerar también los casos insólitos, tales como situaciones de error.
- Completar las especificaciones de procedimiento.
- Verificar que se han introducido todas las combinaciones señal-estado.

Este paso termina cuando no se introducen más *nextstates* (*próximosestados*) no tratados en *transitions* (transiciones). Un *nextstate* no tratado corresponde a un estado no considerado aún. En este caso, puede resultar útil una *signal-state matrix* (*matriz de estado-de-senal*) (véase I.9.4).

**Resultado** – *Especificaciones de proceso completas pero informales.*

## Paso 8 – Especificaciones de proceso formales

### Descripción

- Identificar *géneros* para la información almacenada. Especificar la *signatura* para cada *género* introducido hasta el momento y utilizar *texto informal* para *ecuaciones*.
- Especificar *variables* para la información almacenada y *parámetros de entrada*. Especificar *parámetros de proceso formales*.
- Cambiar *texto informal* en *tareas*, *decisiones* y *respuestas a asignaciones* y *expresiones*.
- Introducir parámetros en *entradas*, *salidas*, *peticiones de creaciones* y *llamadas de procedimiento*.

La *signatura* de un *género* convierte los *operadores* con tipificación y los *literales*. Se tratará de heredar lo más posible de los *géneros* predefinidos. Identificar (en un *comentario*) el conjunto de *operadores* y *literales* que se puede emplear para representar todos los valores posibles. Estos son los constructores de *género* (véase I.5).

**Resultado** – *Especificaciones de género semiformales y especificaciones de proceso formales*

### Ejemplo

```
macrodefinition Declarations;
.....
newtype Direction
    literals up, down; /* constructors:up,down */
    operators
        change_dir: Direction -> Direction;
    axioms
        `if direction is up then down, else up`
endnewtype;
.....
endmacro Declarations;

process Control;
dcl
    goal Floor,          /* the target floor */
    towards Direction,  /* the indicated direction */
    here Floor,         /* current floor */
    moving Direction,   /* current direction of lift chair */
    table Reservations, /* table of reservations */
    .....
endprocess;
```

# Reemplazada por una versión más reciente

además:

```
task 'initiate variables';
```

se convierte en:

```
task here := bot_floor;
task moving := up;
task table := empty_serv;
```

y

```
decision 'goal = current floor';
```

se convierte en:

```
decision goal = here;
```

## Paso 9 – Especificaciones de género formales

### Descripción

- Formalizar *ecuaciones* sustituyendo las *ecuaciones* informales por ecuaciones formales.
- Añadir *ecuaciones* a las *especificaciones de género* hasta completar el conjunto de *ecuaciones*.
- En vez de emplear *ecuaciones*, especificar los operadores con *especificaciones de operador*.

Especificar primero *ecuaciones* para los constructores. Ello da los valores posibles del *género*. Luego, especificar *ecuaciones* para los *operadores* y *literales* restantes. Se considera que el conjunto de ecuaciones está completo cuando todas las expresiones que contienen *operadores* y *literales* no constructores se pueden reescribir como expresiones que sólo contienen *operadores* y *literales* constructores. Véase I.5 para más detalles.

Las especificaciones de operadores son similares a los procedimientos de devolución de valor (de hecho, se definen mediante una transformación en esos procedimientos), y se recomiendan para quienes no estén familiarizados con ecuaciones.

**Resultado** – *Especificaciones de género* completas y formales.

### Ejemplo

```
macrodefinition Declarations;
.....
newtype Direction
  literals up, down; /* constructors:up,down */
  operators
    change_dir: Direction -> Direction;
  axioms
    change_dir(up) == down;
    change_dir(down) == up;
endnewtype;
synonym waiting_time Duration = external;
/* duration of stay at one floor */
newtype Reservations array(Floor,Floor_indicator) adding
  literals empty_serv;
  operators
    goto_req: Reservations, Floor, Floor -> Reservations;
    /* marks a request to go from current floor to another floor */
.....
operator goto_req;
fpar table Reservations, here Floor, goal Floor;
returns Reservations;
start;
  decision goal > here;
    (true):
      task table(goal)!upwards := true;
    (false):
      decision goal < here;
        (true):
```



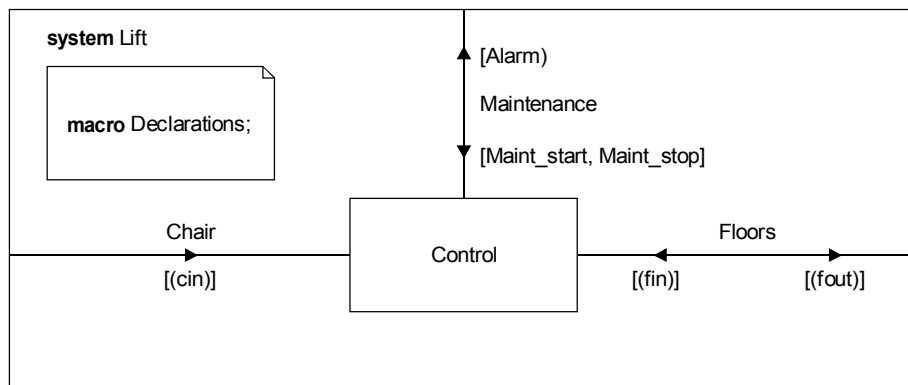
# Reemplazada por una versión más reciente

```

    task table(goal)!downwards := true;
    (false):
        /* no action */
    enddecision;
enddecision;
return table;
endoperator;
.....
endnewtype Reservations;
.....
endmacro Declarations;

```

## I.3.3 Ejemplo: el ascensor (lift)



T1006350-92/d023

```

macrodefinition Declarations;

```

```

/* A lift consists of a chair with GOTO(floor) buttons and floor controls where at
each floor the users can ask for upwards or downwards service.

```

This specification does not model the mechanical part of the lift.

The service order is: If there are more floors to be served in the current direction, then continue to the next of these floors. If there is any floor to be served in opposite direction, then change direction. If there are no floors waiting to be served, stay at the current floor.

A user can control the lift by:

- pressing a button at any floor for service in the desired direction;
- pressing a button in the lift chair for a certain floor.

In addition, the specification also covers some unusual cases, such as start and stop of the operation of the lift. \*/

**signal**

```

Alarm,                /* alarm message                */
Arriving_at_floor,   /* arriving at a floor          */
Chair_stopped(Floor), /* chair stopped at floor      */
Close_door(Floor),   /* close door at floor         */
Door_closed(Floor),  /* door closed at floor        */
Door_opened(Floor), /* door opened at floor        */
Emergency_stop,      /* stop chair immediately      */
Floor_req(Direction,Floor), /* request for the lift from a floor */
Goto(Floor),         /* request to go to the given destination */

```

## Reemplazada por una versión más reciente

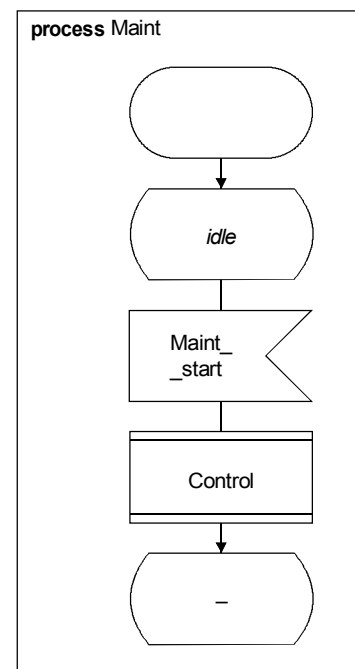
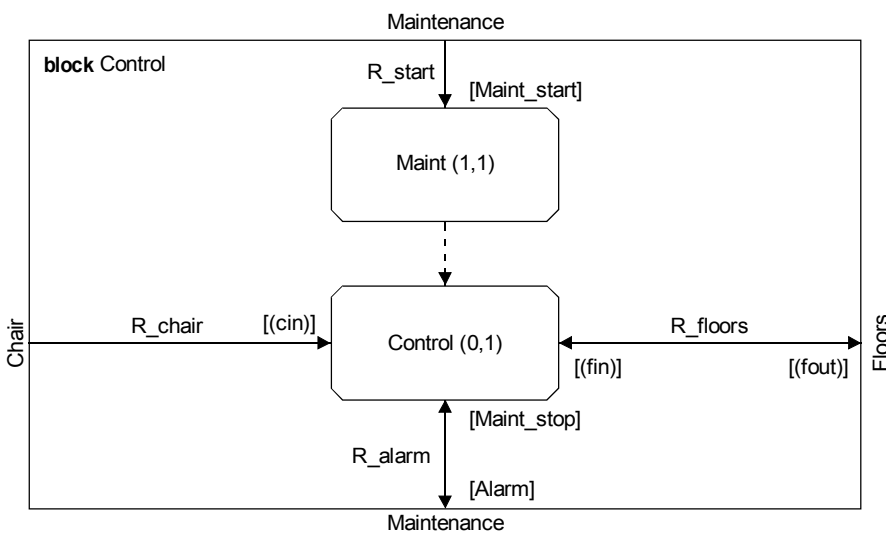
```
Maint_start,          /* start lift operation          */
Maint_stop,           /* stop lift operation           */
Open_door(Floor),    /* open door at floor           */
Restart,              /* restart stopped chair        */
Start_chair(Direction), /* start chair in the given direction */
Stop_chair(Floor);    /* stop chair at floor          */
synonym top_floor Natural = external /* the top floor */;
synonym bot_floor Natural = external /* the bottom floor */;
synonym waiting_time Duration = external;
    /* duration of stay at one floor */
signallist cin = Goto, Emergency_stop, Restart;
signallist fin = Floor_req, Door_closed, Door_opened, Chair_stopped,
    Arriving_at_floor;
signallist fout = Open_door, Close_door, Stop_chair, Start_chair;
syntype Floor = Natural
    constants bot_floor : top_floor
endsyntype;
newtype Direction
    literals up, down;
    operators
    change_dir: Direction -> Direction;
    axioms
    change_dir(up) == down;
    change_dir(down) == up;
endnewtype;
newtype Floor_indicator
    struct upwards Boolean; downwards Boolean;
endnewtype Floor_indicator;
newtype Reservations array(Floor, Floor_indicator) adding
    literals empty_serv;
    operators
    goto_req: Reservations, Floor, Floor -> Reservations;
    /* marks a request to go from current floor to another floor */
    floor_req: Reservations, Floor, Direction -> Reservations;
    /* marks a request to go from a floor in the given direction */
    floor_stop: Reservations, Direction, Floor -> Boolean;
    /* checks if the floor has requested service in the given
    direction */
    cancel_res: Reservations, Direction, Floor -> Reservations;
    /* removes a service request from floor in the given direction */
    more_floors: Reservations, Direction, Floor, Floor, Floor -> Boolean;
    /* checks if there are more floors to be served from the current
    floor to top or bottom depending on the current direction */
    more_floors!: Reservations, Direction, Floor, Floor, Floor -> Boolean;
    /* additional operator needed for iteration over a range */
operator goto_req;
fpar table Reservations, here Floor, goal Floor;
returns Reservations;
start;
    decision goal > here;
    (true):
        task table(goal)!upwards := true;
    (false):
        decision goal < here;
    (true):
        task table(goal)!downwards := true;
```

# Reemplazada por una versión más reciente

```

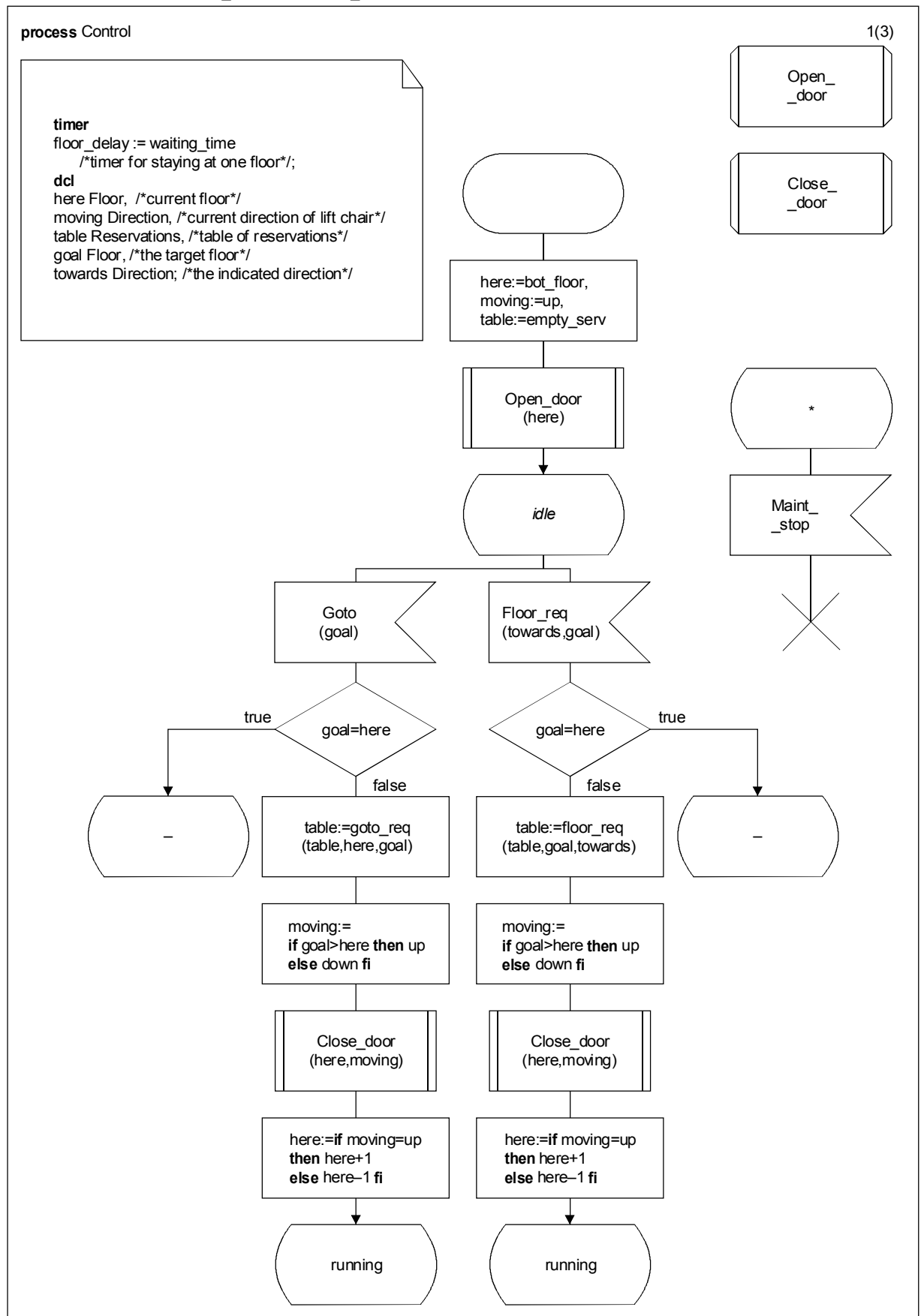
(false):
    /* no action */
enddecision;
enddecision;
return table;
endoperator;
axioms
empty_serv == Make!(Make!(false,false));
floor_req(r,goal,d) ==
    Modify!(r,goal, if d = up
        then upwardsModify!(Extract!(r,goal),true)
        else downwardsModify!(Extract!(r,goal),true) fi);
floor_stop(r,d,f) ==
    if d = up then upwardsExtract!(Extract!(r,f))
        else downwardsExtract!(Extract!(r,f)) fi;
cancel_res(r,d,f) ==
    Modify!(r,f, if d = up
        then upwardsModify!(Extract!(r,f),false)
        else downwardsModify!(Extract!(r,f),false) fi);
more_floors(r,d,f,top,bot) ==
    if d=up then
        if f>=top then false else more_floors!(r,d,f+1,top,bot) fi
    else
        if f<=bot then false else more_floors!(r,d,f-1,top,bot) fi
    fi
more_floors!(r,d,f,top,bot) ==
    upwardsExtract!(Extract!(r,f))
    or downwardsExtract!(Extract!(r,f))
    or if d = up then
        if f>=top then false else more_floors!(r,d,f+1,top,bot) fi
    else
        if f<=bot then false else more_floors!(r,d,f-1,top,bot) fi
    fi
endnewtype Reservations;
endmacro Declarations;

```



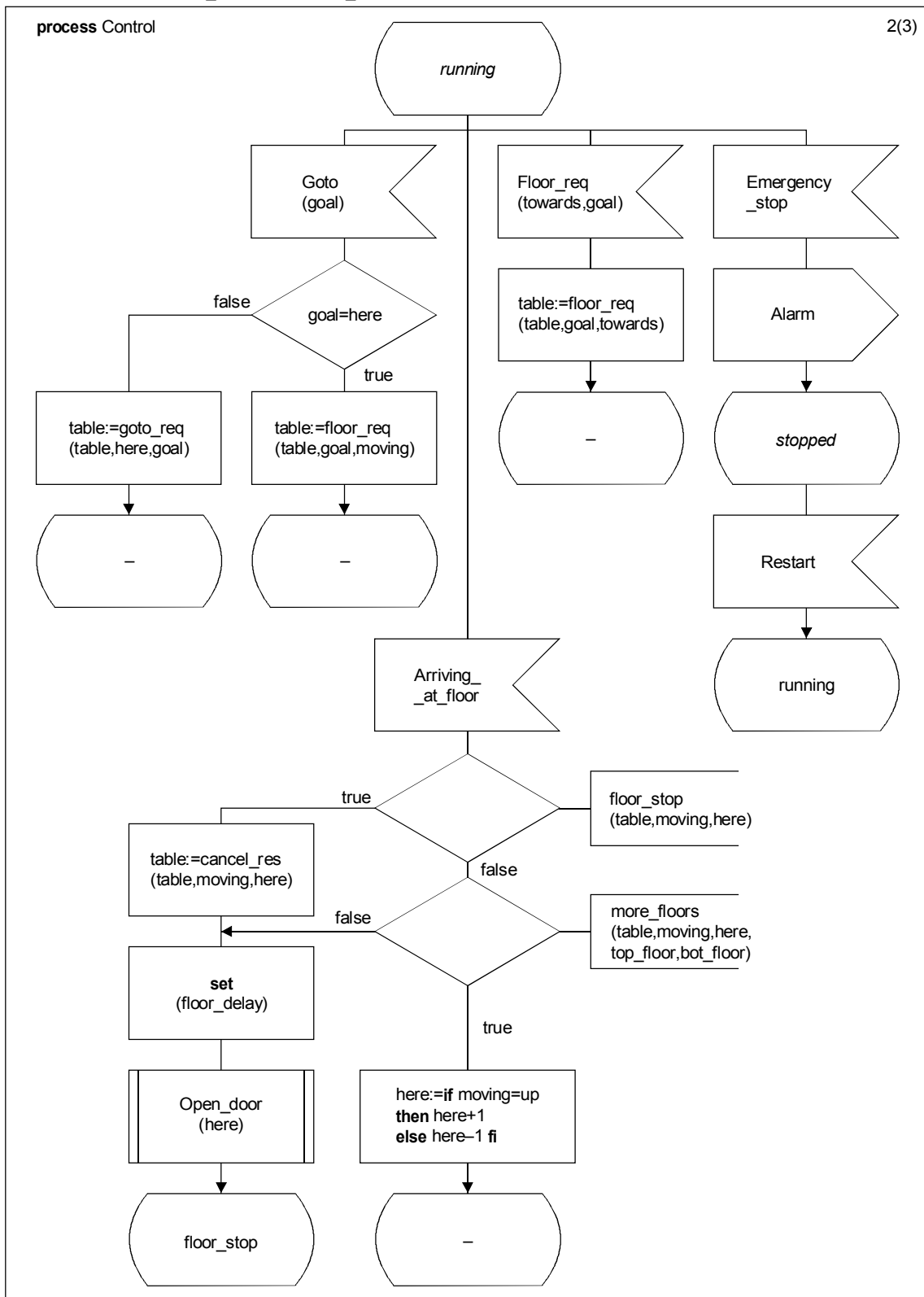
T1006360-92/d024

# Reemplazada por una versión más reciente



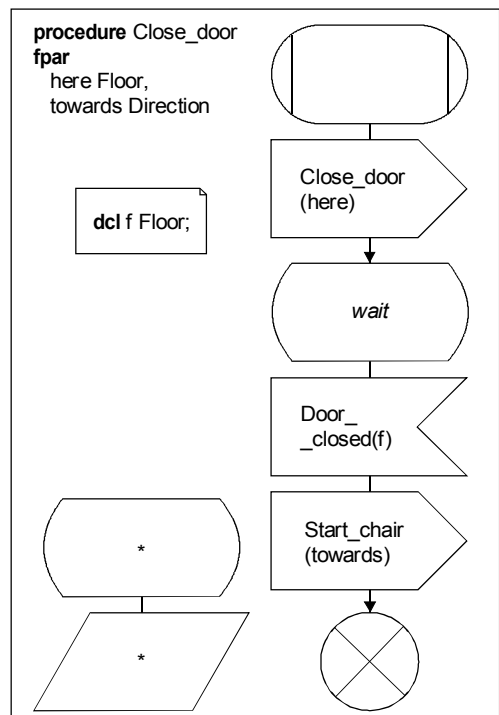
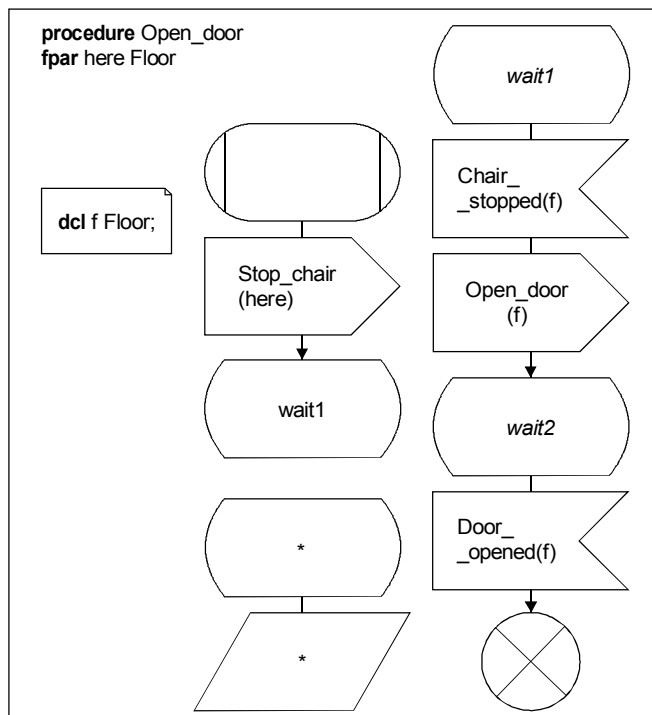
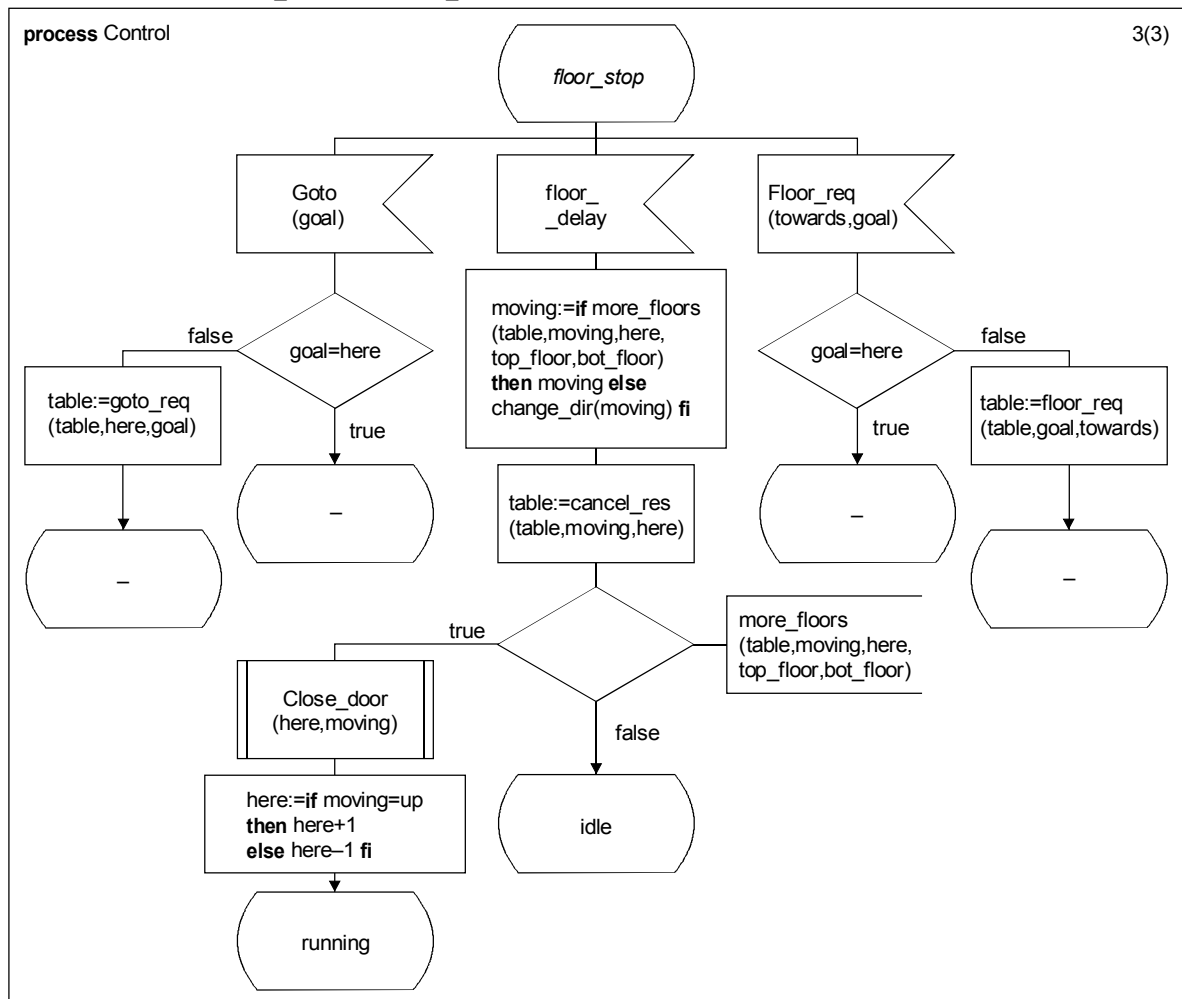
T1006370-92/d025

# Reemplazada por una versión más reciente



T1006380-92/d026

# Reemplazada por una versión más reciente



T1006390-92/d027

# Reemplazada por una versión más reciente

## I.4 Orientación a objetos y SDL

### 1.4.1 Análisis orientado a objetos

La orientación a objetos es una perspectiva sobre la manera de estructurar sistemas y descripciones de sistemas, que puede sugerir incluso algunas formas de organización del proceso de desarrollo de sistema, pero que no proporciona un método completo.

La orientación a objetos se asocia con frecuencia a la programación, interfaces de usuario gráficos y bases de datos. Aplicada a la programación, se asocia a menudo sólo con el código de reutilización. Sin embargo, se da el caso de que un enfoque orientado a objetos es igualmente adecuado en las fases muy preliminares de un proceso de desarrollo de sistema, cuando se hace hincapié en la comprensión del dominio de aplicación y en la primera especificación y análisis de una aplicación. En esta situación, la orientación a objetos no se aplica primordialmente debido a la reutilización, sino porque supone una manera de pensar nueva y provechosa. Dos razones principales por las que se aplica la orientación a objetos en el análisis son:

- El análisis conducirá a una mejor *comprensión* de lo que se supone que tiene que hacer el sistema. ¿Cuáles son las necesidades de los clientes y los requisitos del sistema?
- Se considera una ventaja que *no es necesario cambiar paradigmas* cuando se pasa de la especificación a la realización.

Una parte de la comprensión consiste en identificar los correspondientes *fenómenos* específicos de la aplicación, y clasificarlos en *conceptos*. Los fenómenos se representan mediante *objetos*, los conceptos, mediante *tipos* de objetos. Los conceptos se organizan en jerarquías de conceptos, lo que contribuye a la comprensión de las similitudes y diferencias entre fenómenos. Los tipos que representan conceptos específicos de la aplicación bien demostrados son candidatos a la reutilización en muchas aplicaciones diferentes. A este respecto, es importante tener una visión general de la orientación a objetos, pues permite definir conceptos mediante aspectos de datos y acciones. El conjunto de conceptos relacionados con la aplicación posee un gran valor para las personas que participan en el desarrollo del sistema puedan comprenderlo y razonarlo. El conjunto de conceptos relacionados con la aplicación también constituirá una base de conocimientos comunes importante para las fases siguientes del ciclo de vida del sistema (especificación funcional, diseño, realización y operación/mantenimiento). En el desarrollo del sistema participan diferentes grupos de personas, y esos conceptos facilitarán la comunicación y la interacción entre estos grupos.

Cuando se puede aplicar la orientación a objetos a las especificaciones funcionales formales, se obtendrán especificaciones cuyos elementos esenciales perdurarán en la realización.

- La representación de elementos externos al sistema por componentes del sistema especificado será importante también en las realizaciones.
- La identificación de conceptos específicos de aplicaciones (sectores) clave, representándolos como tipos, y representando las jerarquías de conceptos mediante jerarquías de subtipos conducirá a definiciones de tipos que son valiosas también para las realizaciones. Las jerarquías de subtipos que resultan de la aplicación de jerarquías de conceptos serán más estables que las jerarquías de subtipos que se inventan únicamente para la realización. Las jerarquías de conceptos específicos de una aplicación son, con frecuencia, el resultado de muchos años de experiencia de usuarios y expertos en la materia.
- La estructura del sistema especificado puede hacerse muy semejante a la estructura del sistema realizado.

El hecho de que partes fundamentales de la especificación sean estables desde el diseño hasta la realización, no solo resultará útil para el mantenimiento del sistema sino que también mejorará la calidad de la especificación en tanto que *contrato* entre el suministrador del sistema y el cliente.

El análisis orientado a objetos se ilustra mediante un ejemplo casi real: un sistema de *Banco*. El ejemplo se construye a partir de conceptos ya desarrollados (en relación con un sistema de *Control de Acceso* almacenado en una biblioteca). Esto refleja la práctica de la industria, donde los nuevos sistemas no se elaboran partiendo de cero sino utilizando, en gran medida, sistemas y componentes ya existentes.

El análisis se puede dividir aproximadamente en cuatro fases: *comprensión*, *búsqueda*, *generalización* y *especialización*.

#### Paso 1 – Descripción textual

- Bosquejar el sistema que hay que especificar mediante descripciones textuales y dibujos.

En la mayoría de los casos, ya se dispone de algunas descripciones sobre el sector que se está modelando, que pueden ser especificaciones de necesidades de un cliente, manuales y descripciones informales de sistemas anteriores o, simplemente, bosquejos informales destinados a otros fines.

# Reemplazada por una versión más reciente

Un *banco* es un lugar en el que se manipula *dinero*. El dinero se puede manipular de muchas maneras: *efectivo* y *cheques*. El dinero también se puede manipular sin ningún medio físico, indicando su valor, por ejemplo, por medios electrónicos, transacciones internas de las *cuentas*. Se destacan aquellas partes del banco en que se trabaja con representaciones físicas del dinero.

En el banco hay *ventanillas*. Algunas (llamadas *caja*) están atendidas por *cajeros*; otras están automatizadas y se denominan *minibancos*.

Algunos *clientes* prefieren utilizar el minibanco, donde insertan su *tarjeta* y escriben su código personal en un teclado para tener acceso a su cuenta y recibir dinero en efectivo. También reciben otros servicios, tales como la *impresión* de su estado de cuenta. El minibanco tiene una *pantalla*, que sirve de dispositivo interactivo con el cliente.

Otros clientes prefieren los cajeros, con los que pueden hablar y comunicarse. El cajero tiene su propia pantalla y teclado, y puede prestar más servicios que los minibancos automáticos.

Este paso proporciona una comprensión muy preliminar de lo que es todo el sector de la aplicación. Ayuda a definir las fronteras del sistema e indica los conceptos que deben ser incluidos en el diccionario (véase lo que sigue).

## Paso 2 – Diccionario

- Hacer u obtener un diccionario del sector de que se trata. El diccionario se mantendrá actualizado durante todo el desarrollo.

En este paso, la comprensión aumenta con la enumeración de los conceptos pertinentes. Algunos de esos conceptos se encontrarán en diccionarios y otros tendrán que ser definidos.

La noción clásica de concepto se caracteriza por:

- la *extensión*, el conjunto de fenómenos que abarca el concepto,
- la *intensión*, el conjunto de propiedades que caracterizan de alguna manera los fenómenos en la extensión del concepto,
- la *designación*, el conjunto de nombres con el que se conoce el concepto.

La representación de conceptos mediante tipos de instancia se ajusta a este patrón: las instancias pertenecen a la extensión, la definición del tipo proporciona la intención, y el nombre del tipo es su designación.

En la descripción textual, los *nombres*, que representan conceptos clave del sector objeto de análisis, están aislados. Se presentan en letra cursiva en el **paso 1**. En la terminología orientada a objetos, estos conceptos clave son candidatos a *tipos*, a partir de los cuales se generarán los objetos. Durante los pasos siguientes del análisis, se actualizará y complementará el diccionario.

El banco podría tener el siguiente diccionario inicial:

<i>Cuenta</i>	Representación informatizada de dinero asociada con un poseedor
<i>Banco</i>	Lugar donde se manipula dinero
<i>Tarjeta</i>	Medio de identificación personal
<i>Efectivo</i>	Representaciones físicas de dinero garantizadas por el Estado
<i>Caja</i>	Ventanilla con un empleado (cajero)
<i>Cheque</i>	Representación de dinero con diversos garantes
<i>Ventanilla</i>	Puertos de comunicación con el sistema bancario
<i>Cliente</i>	Procesos externos que señalizan al banco y reciben dinero de éste
<i>Pantalla</i>	Medio de visualización para transmitir mensajes escritos pero volátiles al cliente o al cajero
<i>Tecla</i>	Parte de un teclado
<i>Minibanco</i>	Cajero automático
<i>Dinero</i>	Señales últimas de un banco
<i>Anotación impresa</i>	Mensaje escrito en papel (señal para el cliente)



# Reemplazada por una versión más reciente

## Paso 3 – Agregación

- Hacer un bosquejo de una instancia típica del sistema objeto de análisis. Etiquetar los elementos del bosquejo.
- A partir de la instancia típica, describir las relaciones de «consiste-en» («*consists-of*») entre los conceptos.

Al observar un banco, pueden verse las piezas de que consiste, y las piezas que forman cada una de las piezas, etc. La agregación consiste en agrupar partes separadas en un todo, y tiene que ver con relaciones de «consiste-en», que es una relación importante en la mayoría de los paradigmas de análisis y, por consiguiente, también en la orientación a objetos. Al especificar las relaciones «consiste-en» se debe tener cuidado en distinguir entre conceptos (tipos) e instancias de tipos. Para evitar confusiones, los conceptos se indicarán con mayúscula inicial (por ejemplo, *Banco*) y para las instancias se utilizarán minúsculas (por ejemplo, *banco*). En la Figura I.4-1 se muestran las relaciones «consiste-en» de una instancia de banco determinado que tiene cuatro instancias de ventanillas determinadas.

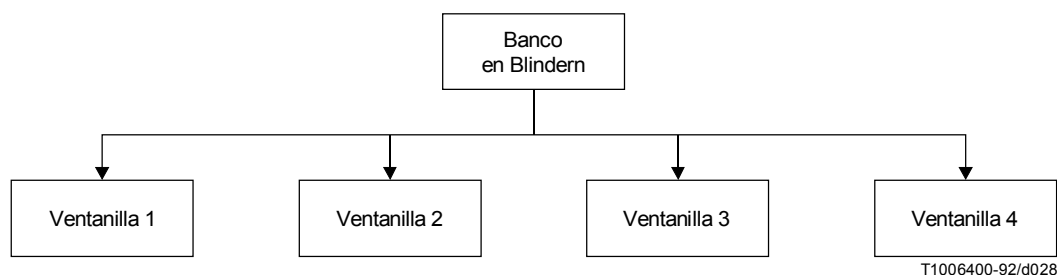


FIGURA I.4-1/Z.100

### Relaciones de «consiste-en» entre instancias

Un análisis de sistema se ocupa, sobre todo, de los sistemas en general más que de sus instancias específicas. Lo mismo puede decirse de los bancos y ventanillas. En consecuencia, la relación «consiste-en» entre *Banco* y *Ventanilla* en general resulta de interés (véase la Figura I.4-2).

En esta figura se trata realmente de tipos de relación, que son asociaciones entre conceptos (tipos de objetos). Un tipo de relación tiene una cardinalidad, que indica el número máximo de instancias de cada tipo en cuestión. También es posible indicar que algunas instancias son facultativas (flecha de trazo interrumpido), es decir, que pueden no formar parte de ciertas instancias del banco.

## Paso 4 – Similaridades

- Para cada concepto incluido en el diccionario, debe preguntarse si todos los objetos que pertenecen a la extensión del concepto tienen las mismas propiedades. Si no es el caso, deben hallarse las especializaciones, que pueden ampliar o no el diccionario.
- Durante la especialización, ampliar la descripción del diccionario con propiedades que aumentan la comprensión del concepto.

En el paso anterior se han analizado las relaciones entre instancias. En este paso se trata de las relaciones entre conceptos.

Como se ha dicho a propósito de los pasos anteriores, *Minibanco* y *Caja* son especializaciones de *Ventanilla*, lo que significa que todos los minibancos y las cajas son ventanillas, y que cualquier cosa general que se diga sobre las ventanillas será válida para los minibancos y las cajas. La jerarquía de especialización se muestra en la Figura I.4-3.

# Reemplazada por una versión más reciente

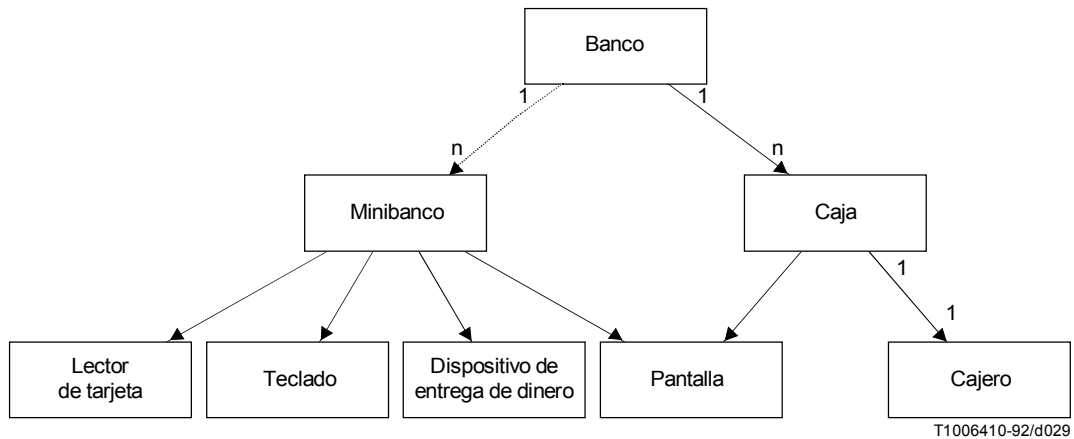


FIGURA I.4-2/Z.100

Relaciones de «consiste-en» entre conceptos

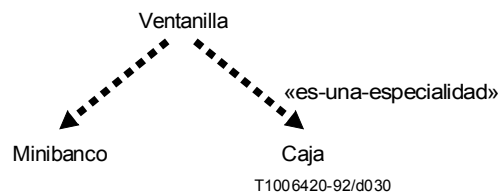


FIGURA I.4-3/Z.100

Jerarquía de especialización de ventanilla

Una vez elaboradas esas jerarquías de especialización, habrá que profundizar en la definición de lo que caracteriza a un *Minibanco* o a una *Caja*, lo que los distingue entre sí y con respecto a la noción general de *Ventanilla*. En esas relaciones de especialización se trata de describir los aspectos generales y comunes una sola vez. La descripción de *Ventanilla* comprenderá todas las características comunes, mientras que la descripción de *Minibanco* consistirá únicamente en los elementos específicos de los minibancos.

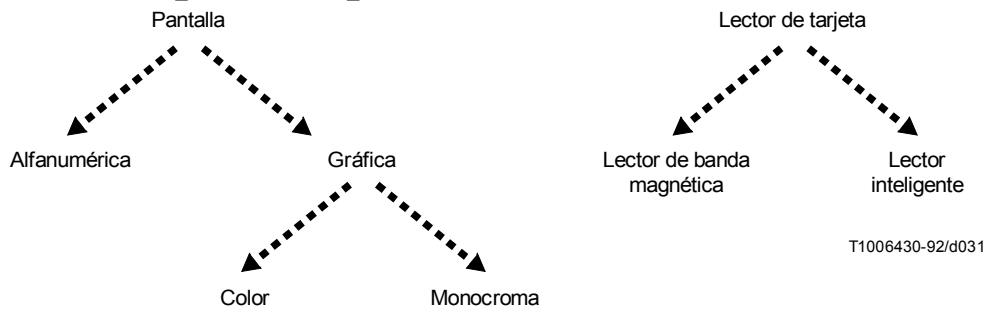
Como se muestra en la Figura I.4-4, el ejemplo del banco tiene más jerarquías de especialización.

## Paso 5 – Búsqueda en la biblioteca

- Ver si hay una *Y* en una biblioteca que es similar a una *X* en el diccionario. Cuando se encuentre una similitud, hacer *X* una especialización directa de *Y* o reestructurar la biblioteca creando una *Z* que es una generalización de *X* e *Y*.

Siguiendo la interpretación cabal de *Bank*, se desea ver si hay piezas que residen en una biblioteca que pueden ser utilizadas en el diseño del sistema *Banco*. Encontramos que hay muchas similitudes entre el sistema *Control de Acceso* (véase la Figura I.4-5) y *Bank*. Una *Estación Local* es bastante similar a una *Ventanilla*.

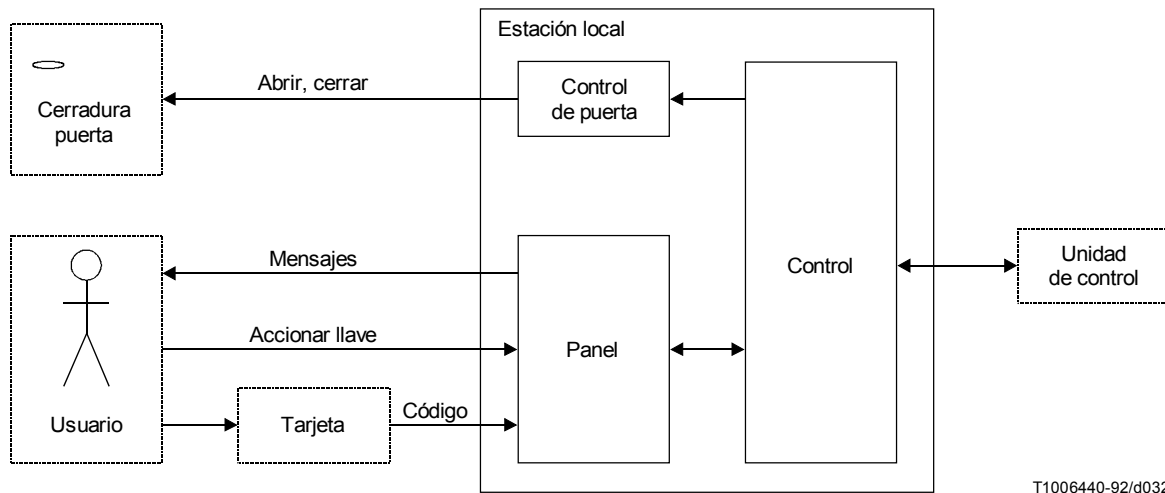
# Reemplazada por una versión más reciente



T1006430-92/d031

FIGURA I.4-4/Z.100

## Jerarquías de especialización de Pantalla y Lector de tarjeta



T1006440-92/d032

FIGURA I.4-5/Z.100

## Sistema Control de Acceso

¿Cómo pueden aprovecharse las similitudes entre *Control de Acceso* y *Banco*? Hay dos diferentes enfoques, que dependen de la relación de especialización entre el concepto de biblioteca y el concepto de aplicación: o bien el concepto de aplicación es una especialización directa del concepto de biblioteca, o no hay relación de especialización directa entre ellos, pero son aún muy similares.

Si se trata de la relación de especialización anterior, está bastante bien. Se heredarán grandes partes de la descripción de la biblioteca y sólo es necesario especificar características adicionales (véase la Figura I.4-6). El problema potencial es que aunque la relación de especialización parece correcta a primera vista, puede no admitir completamente una

## Reemplazada por una versión más reciente

investigación más minuciosa. Las unidades de la biblioteca, que no se diseñaron teniendo en mente un banco, pueden tener algunas estructuras internas que no son lo que se desea para un banco. Pueden no ser «virtuales» y entonces no pueden ser redefinidas. Si este es el caso, se debe llegar a la conclusión de que aunque los conceptos generales parecen estar en relación de especialización directa, no lo están después de todo.

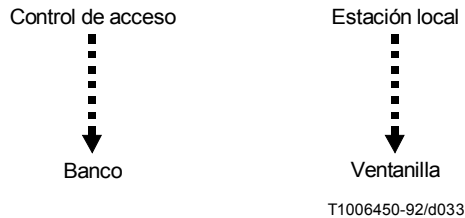


FIGURA I.4-6/Z.100

### Relación de especialización directa entre aplicación y biblioteca

El segundo caso es que sabemos que *Control de Acceso* y *Banco* son similares, pero que no están en una relación de especialización directa. Por consiguiente, ambos son especializaciones de un concepto común, más general, como se muestra en la Figura I.4-7.

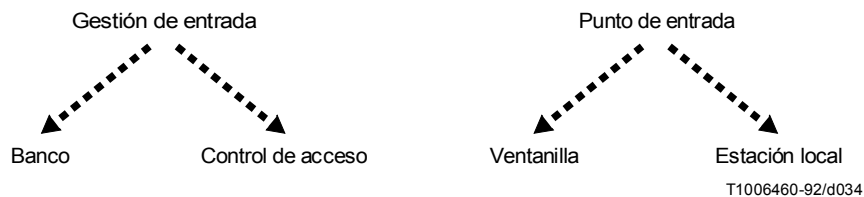


FIGURA I.4-7/Z.100

### Relación de especialización indirecta entre la aplicación y la biblioteca

El análisis es ahora más completo. Habrá que reestructurar la biblioteca introduciendo los conceptos nuevos y más generales *Gestión de Entrada* y *Punto de Entrada* y convertir las «antiguas» unidades de biblioteca *Control de Acceso* y *Estación Local* en especializaciones directas de las nuevas unidades de biblioteca.

Se han conseguido dos cosas. Primero, la biblioteca es ahora más general y, por tanto, más aplicable a nuevas aplicaciones, diferentes de *Control de Acceso* y *Banco*. Segundo, se han creado unidades de biblioteca de las cuales los conceptos *Banco* son especializaciones directas.

Además, no se ha perdido nada, pues *Control de Acceso* es concretamente idéntico a lo que era antes de la reestructuración. Se ha logrado la compatibilidad hacia adelante y hacia atrás.

Esta reestructuración de la biblioteca tiene algunas repercusiones. Se observará que cuanto más generales son los conceptos, tanto más virtuales son los tipos. (Ello, a su vez, disminuye el poder de análisis, pero no necesariamente el poder descriptivo.) Al hacer que *Control de Acceso* sea idéntico a lo que era antes (de manera que el análisis anterior sigue siendo válido para sus especializaciones), ha de tenerse cuidado para «finalizar» los tipos virtuales heredados de los nuevos conceptos generales que no eran virtuales en *Control de Acceso* original.

# Reemplazada por una versión más reciente

Otra repercusión muy típica e importante de la búsqueda en la biblioteca es que, al establecer la similaridad entre *Control de Acceso* y *Banco*, pueden aparecer características de *Control de Acceso* que se pueden aplicar a *Banco*, pero en las que no se pensó en el contexto de *Banco*. Un ejemplo posible es *Estación de registro* en *Control de Acceso*. Una estación de registro para las transacciones bancarias puede ser una buena idea.

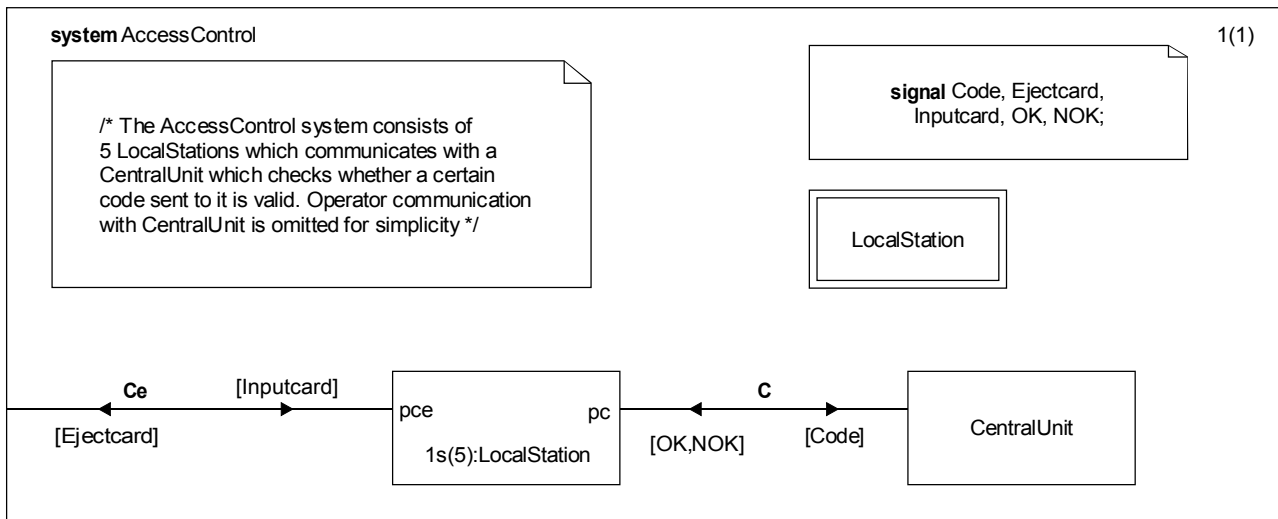
## I.4.2 Aplicación de los conceptos SDL orientados a objetos

En el punto anterior se describe un análisis orientado a objetos independientes del lenguaje. En éste, se profundizará lo expuesto en el **paso 4** y el **paso 5** aplicando los conceptos SDL orientados a objetos.

### Paso 6 – Generalizaciones

- Conseguir componentes flexibles mediante la introducción de *tipos virtuales*. Asegurarse de que esos tipos reciben nombres generales adecuados. Equilibrar la flexibilidad de los tipos virtuales mediante el empleo de **atleast** para restringir la posibilidad de redefiniciones.
- Conseguir la independencia de *señales* y *géneros* mediante la introducción de *parámetros de contexto*. Equilibrar esta independencia restringiendo los parámetros de contexto.

El punto de partida es un sistema *Control de Acceso* con algunos tipos bastante generales (véase el Ejemplo I.4-1).



T1006470-92/d035

### EJEMPLO I.4-1

#### Diagrama del sistema *Control de Acceso*

Uno de los tipos centrales es el tipo de bloque *Estación Local (LocalStation)* que (en una versión) se define como en el Ejemplo I.4-2.

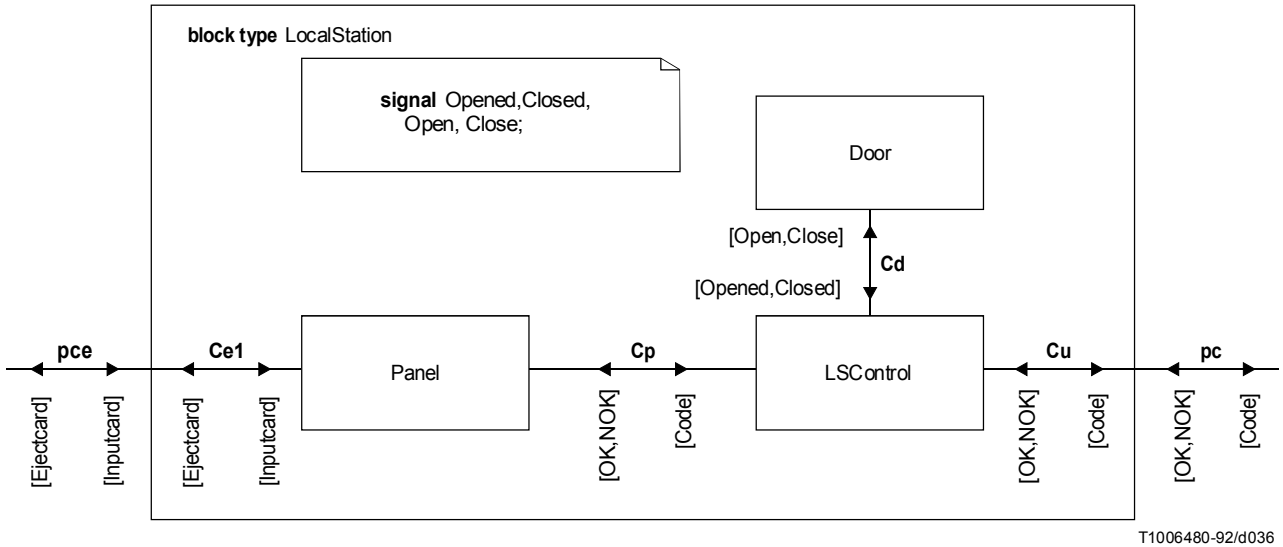
Aunque puede verse que una *Ventanilla* se puede modelar de forma más o menos parecida, no todos los nombres de componentes son correctos, y la definición no es lo suficientemente flexible para este nuevo fin.

Será preciso generalizar para tener una definición de tipo más general, que se puede especializar en *Estación Local (LocalStation)* (del *Control de Acceso (AccessControl)*) y en *Ventanilla (Counter)* (del *Banco (Bank)*). Hace falta flexibilidad más que posibilidad de análisis.

La generalización se hará en dos dimensiones. Primero, se tratará de que las componentes sean ajustables, es decir, redefinibles en especializaciones. Esto se logra mediante tipos virtuales. Segundo, se puede suponer que las señales de un sistema *Banco* son diferentes de las de un sistema *Control de Acceso*. Por ello, se generalizará en los nombres de

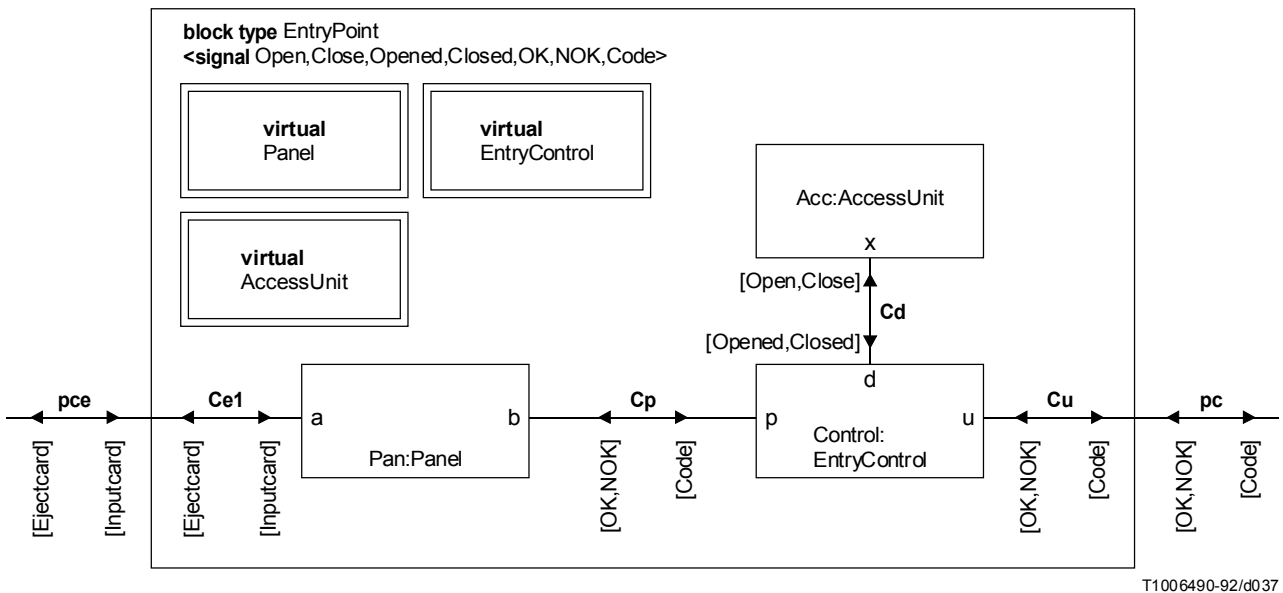
# Reemplazada por una versión más reciente

señales. Esto se logra mediante parámetros de señales de contexto (véase el Ejemplo I.4-3). Obsérvese que el tipo de bloque *Punto de Entrada (EntryPoint)* está parametrizado, y que las instancias no se pueden obtener directamente del mismo.



EJEMPLO I.4-2

*Estación local del (antiguo) Control de Acceso*



EJEMPLO I.4-3

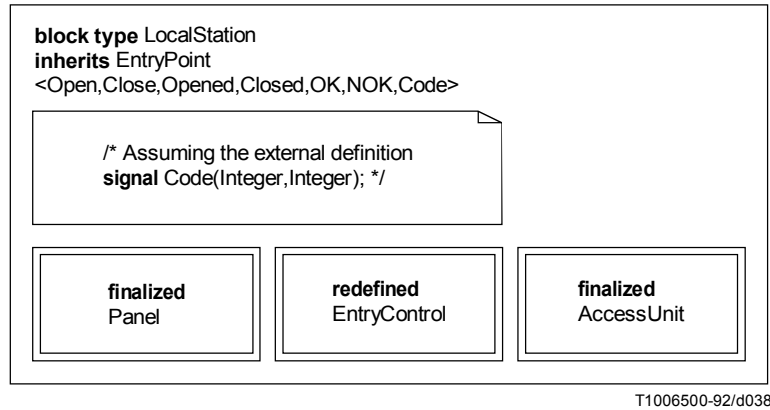
*Tipo de bloque general Punto de Entrada*

# Reemplazada por una versión más reciente

## Paso 7 – Especializaciones

- Al especializar, debe cuidarse de guardar el equilibrio correcto entre flexibilidad y posibilidad de análisis conseguido gracias a la utilización de **atleast** y **finalized** para restringir los tipos virtuales.

Al utilizar el tipo de bloque *Punto de Entrada* (*EntryPoint*), se puede definir *Estación Local* (*LocalStation*) como una especialización (véase el Ejemplo I.4-4).

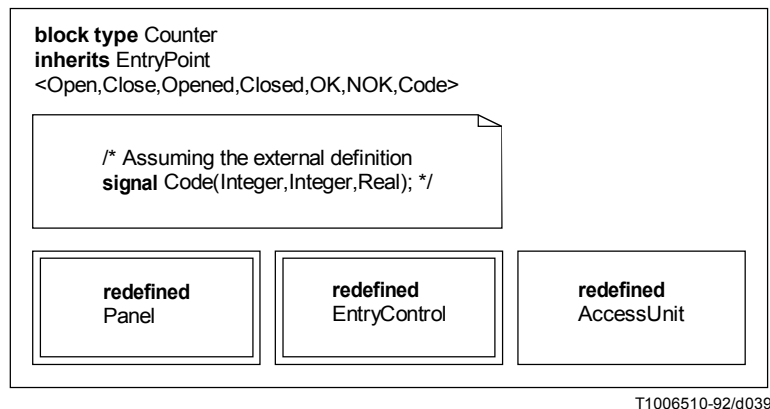


EJEMPLO I.4-4

### Tipo de bloque general *Estación Local*

Se observa que *Panel* (*Panel*) y *Unidad de Acceso* (*AccessUnit*) están **finalized** para restringir la flexibilidad y aumentar la posibilidad de análisis. Además, esto se hace, en este caso, para lograr la compatibilidad funcional entre la nueva *Estación Local* (*LocalStation*) y la antigua. No se muestra el nivel de sistema, pero se indica que en el nivel de sistema hay una definición de señal de los parámetros reales, y la definición de la señal *Código* (*Code*) con dos parámetros de entero resulta de especial interés.

Ahora se puede definir análogamente la *Ventanilla* (*Counter*) del banco (véase el Ejemplo I.4-5).



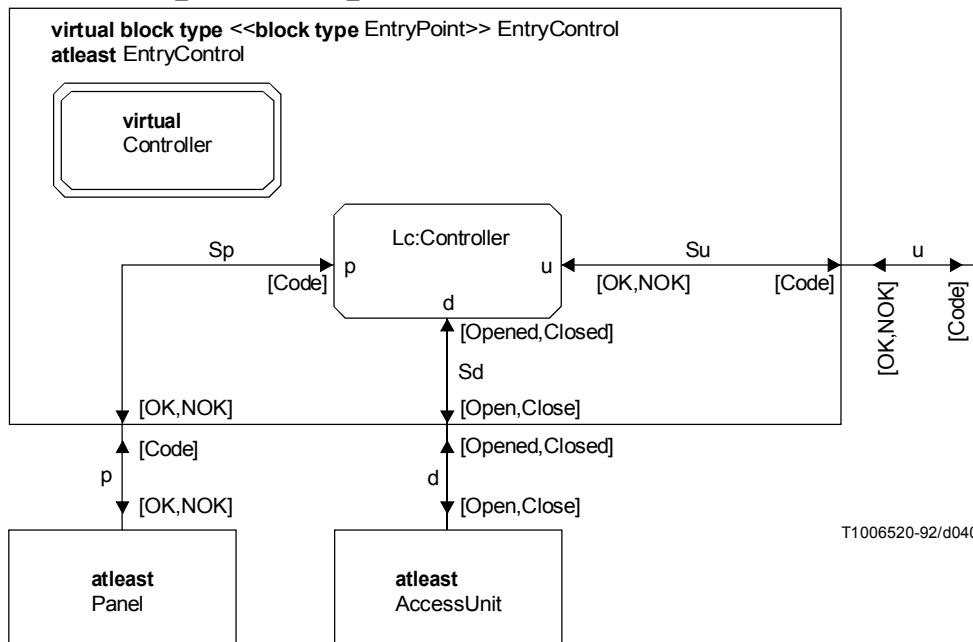
EJEMPLO I.4-5

### Tipo de bloque *Ventanilla*

Nótese que se supone que el sistema *Banco* (*Bank*) tiene una señal *Código* (*Code*) que tiene tres parámetros (a diferencia de los dos parámetros del sistema *Control de Acceso*) (*AccessControl*).

En el Ejemplo I.4-6 se muestra el tipo de bloque *Control de Entrada* (*EntryControl*).

# Reemplazada por una versión más reciente

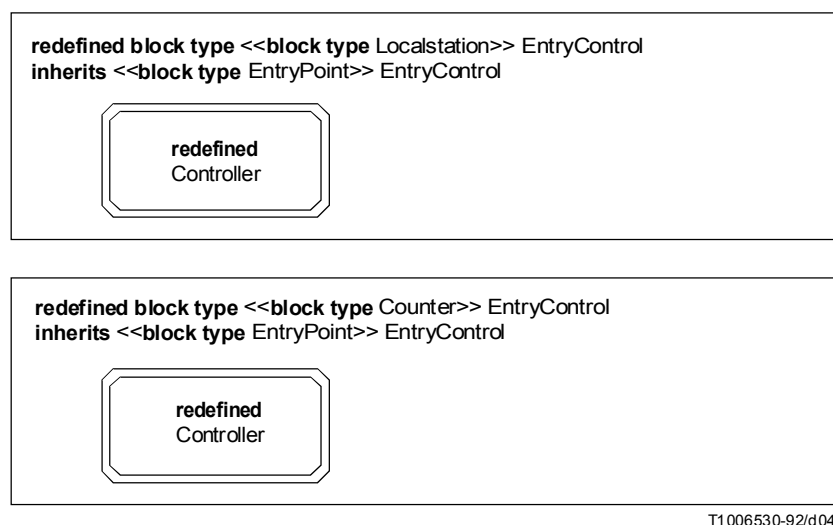


EJEMPLO I.4-6

## Tipo de bloque *Control de Entrada*

Esta es la definición por defecto del tipo virtual, y se observa que especifica restricciones mediante una cláusula **atleast**. La restricción consiste en que todas las redefiniciones deberán ser especializaciones de ésta. Este tipo de bloque propiamente dicho tiene un tipo de proceso virtual.

Las redefiniciones correspondientes de los sistemas *Control de Acceso* y *Banco* son realmente muy sencillas (véase el Ejemplo I.4-7). Obsérvense las calificaciones largas que se necesitan para identificar los diagramas. Ello obedece a que el empleo de tipos virtuales aumenta la cantidad de nombres que entran en colisión. Los usuarios del SDL no tienen que ocuparse de este problema, ya que seguramente será tratado por las herramientas.



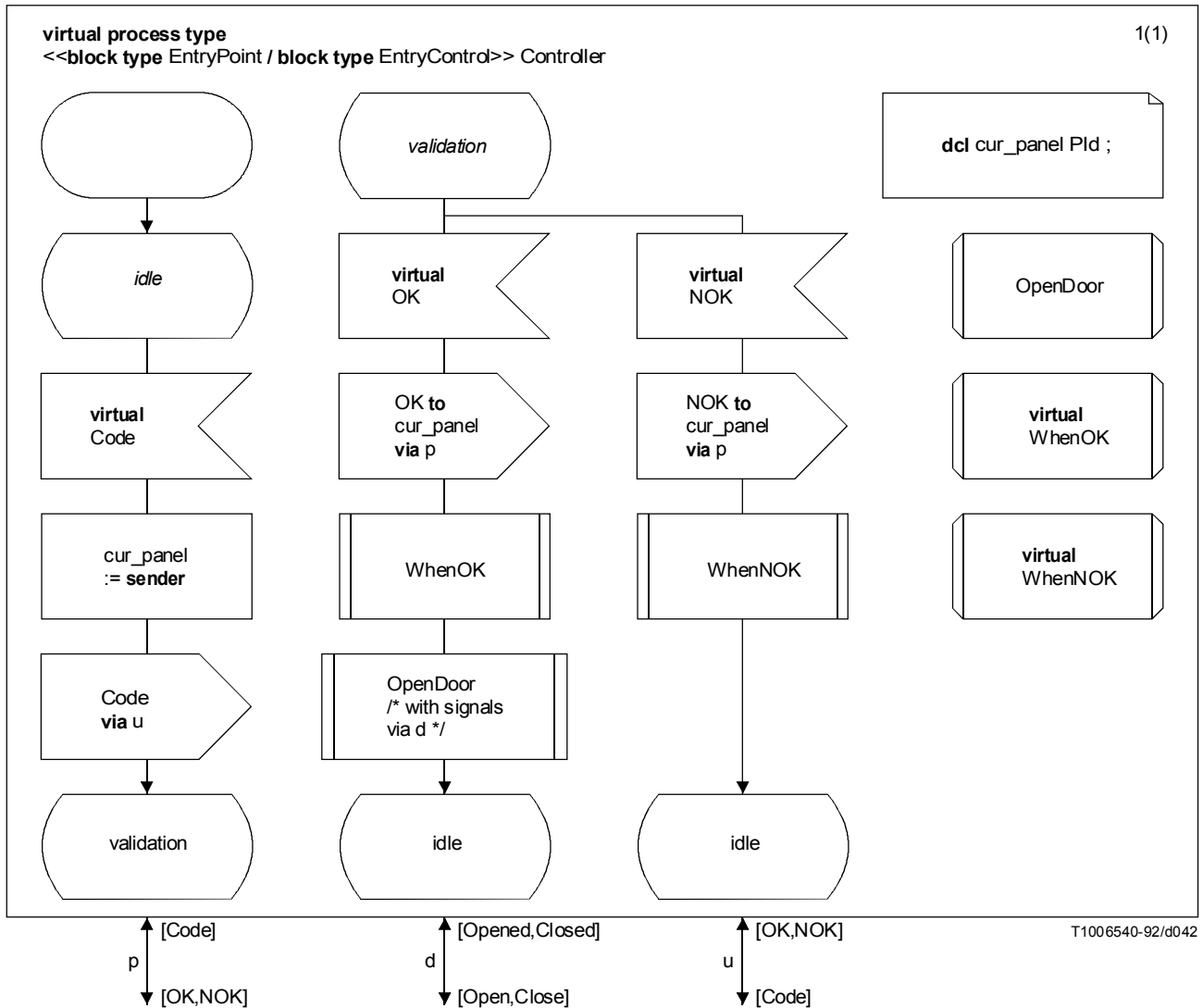
EJEMPLO I.4-7

## Redefiniciones de *Control de Entrada*



# Reemplazada por una versión más reciente

A continuación se ofrece la definición del tipo de proceso *Controlador (Controller)* (véase el Ejemplo I.4-8).



EJEMPLO I.4-8

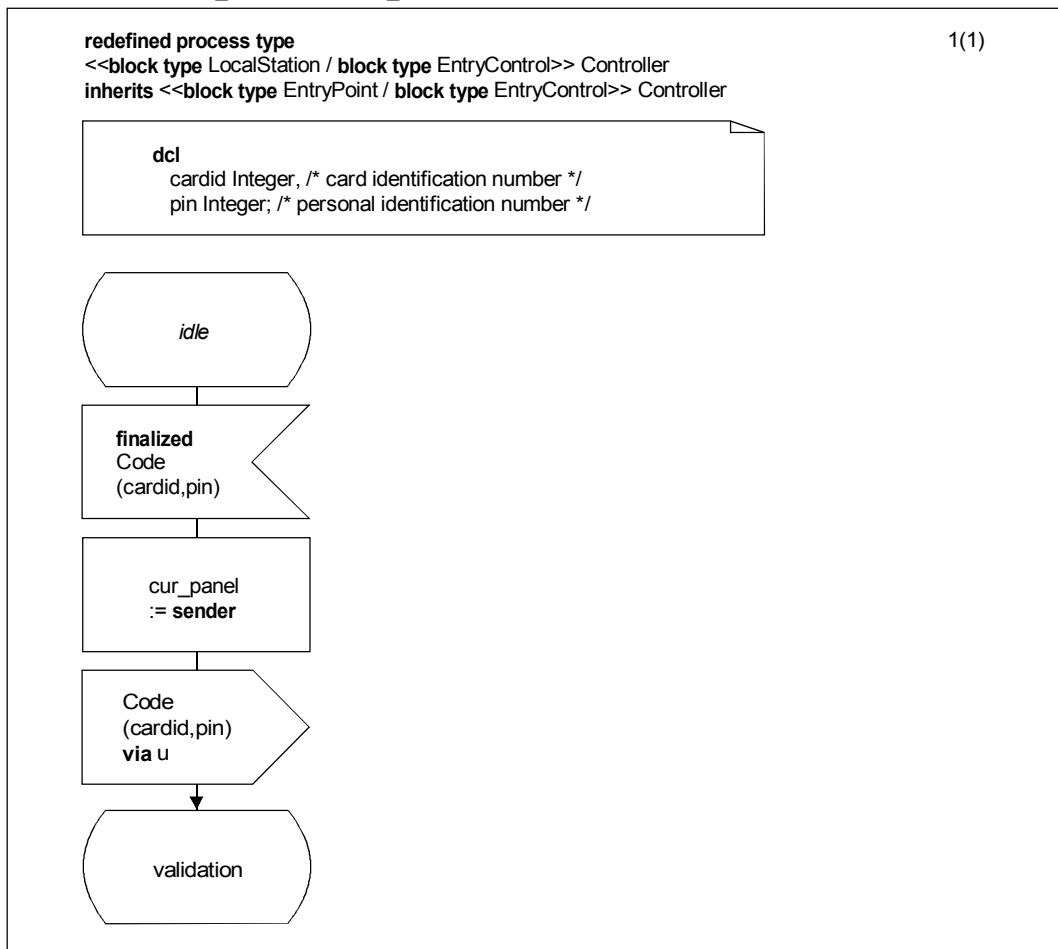
## Tipo de proceso *Controlador*

Una vez más, se observa que se da flexibilidad a las partes internas de la definición al declararlas **virtuales**. En este ejemplo todas las transiciones son virtuales. En los Ejemplos I.4-9 e I.4-10 se muestra cómo se ha redefinido en especializaciones.

La redefinición del Ejemplo I.4-9 trata de la recepción de la señal *Code (Código)*. La transición utiliza dos parámetros de señal: *cardid* y *pin*. (La definición de la señal puede ser la misma, puesto que se permite que una entrada tenga menos parámetros que los especificados en la definición de señal, y se descartará la información suplementaria transportada por la instancia de señal.)

La redefinición del Ejemplo I.4-10 se relaciona también con la recepción de señales: *Code* y *OK*. Las transiciones utilizan parámetros de señal nuevos: *cardid*, *pin*, *amount* y *leftonaccount*.

# Reemplazada por una versión más reciente



T1006550-92/d043

## EJEMPLO I.4-9

### Redefinición de *Controlador* en *Control de Acceso*

## I.5 Elaboración de una especificación ADT completa por pasos

Se afirma que los géneros predefinidos de la Recomendación Z.100 son completos. ¿Por qué esas definiciones son completas? Para responder a esta pregunta, primero hay que determinar en qué consiste la completación. En este punto se explicará y demostrará el método de función constructor (CFM, *constructor-function method*). El CFM presta cierta ayuda en la elaboración de especificaciones (ADT, *abstract data types*) completas de las pruebas de desarrollo aceleradas.

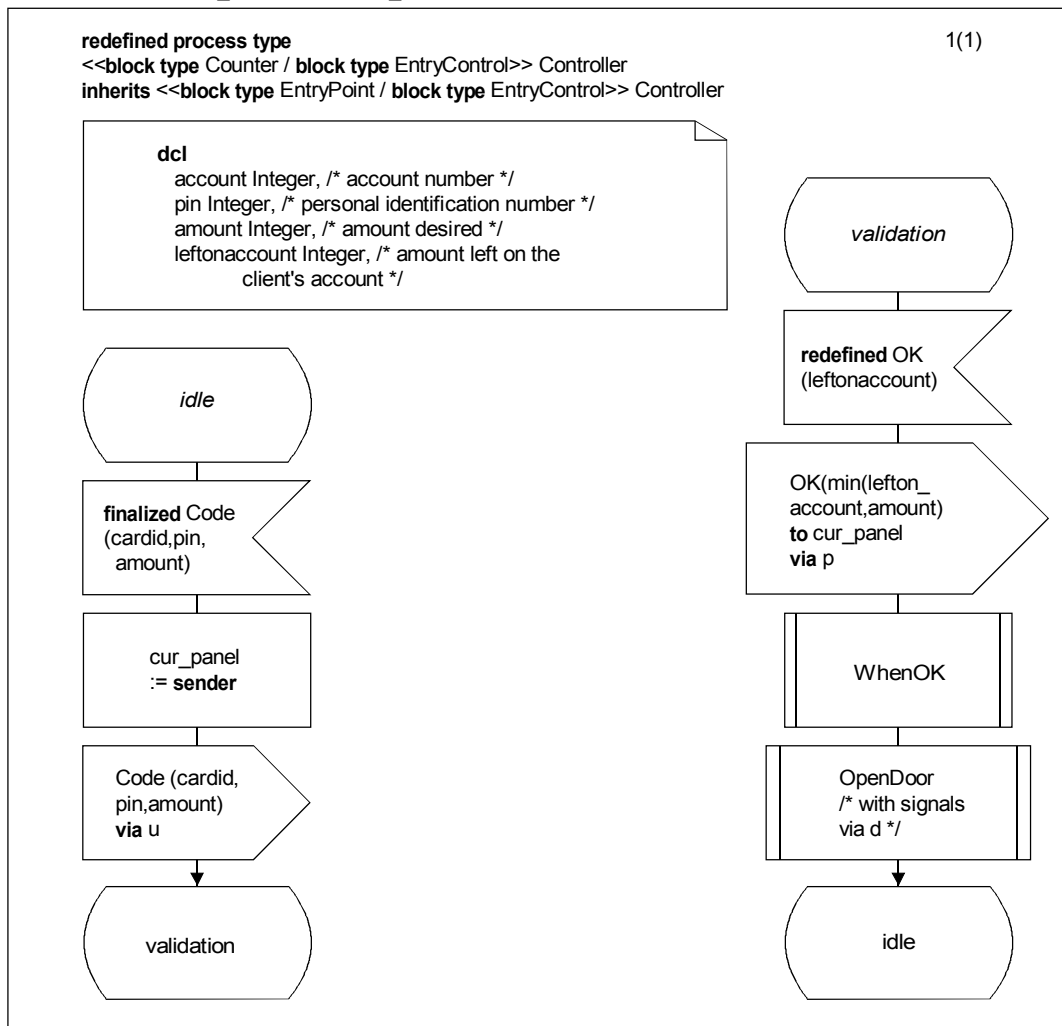
En lo que sigue se supone que el lector está familiarizado con las bases matemáticas de ADT (las firmas generan términos divididos en clases de equivalencia por ecuaciones; las clases de equivalencia definen los valores de los géneros).

### I.5.1 Completación de una especificación ADT

No se puede dar una definición absoluta de la completación. Un género está especificado completamente cuando sus valores corresponden a los valores que el especificador pretendía definir. Puesto que resulta difícil medir la intención de una persona, la completación de una especificación de género no se puede definir formalmente<sup>1)</sup>.

<sup>1)</sup> En la teoría de los sistemas de reescritura, «completa» y «completación» tienen un significado formal, pero en este punto, «completa» se utiliza en sentido no formal.

# Reemplazada por una versión más reciente



T1006560-92/d044

## EJEMPLO 1.4-10

### Redefinición de *Controlador en Banco*

Se puede hacer algo con la definición relativa de compleción (relativa con respecto a la intención de una persona). Para empezar, hay que tener en cuenta que dos términos destinados a representar valores diferentes no se ponen en la misma clase de equivalencia. Segundo, hay que cuidar que todos los términos se pongan en una de las clases de equivalencia intencionadas (= valores) pues si no, estos términos se convertirán en valores nuevos, no intencionados.

La intención de la especificación del género *Boolean* es crear dos valores de género *Boolean*, de forma que los dos literales pertenezcan a clases de equivalencia diferentes. Ello significa que en las ecuaciones se tendrá cuidado de no poner *true* y *false* en la misma clase de equivalencia, y de no crear ninguna otra clase de equivalencia además de las clases de equivalencia a las que pertenecen *true* y *false*. Las ecuaciones se formulan de tal manera que una expresión *Boolean* arbitraria se puede reducir desde el interior.

En cada expresión, la subexpresión más interna que no es un literal puede ser reducida a un literal mediante la aplicación de una o más ecuaciones. La aplicación repetida de este procedimiento reduce cada expresión *Boolean* a un literal, independientemente de su complejidad. De este modo, se puede concluir que hay, como máximo, dos valores.

En principio, no es posible mostrar que *true* y *false* no están en la misma clase de equivalencia. Sin embargo, las funciones definidas en el género *Boolean* son funciones matemáticas bien conocidas y que se comportan adecuadamente.

# Reemplazada por una versión más reciente

Cabe señalar que hay que tener cuidado incluso con funciones matemáticas que se comportan bien. Una manera de definir todos los números naturales (género *Nat*) es utilizar el literal *0* y el operador de sucesor *succ*<sup>2)</sup>. Para expresar la igualdad de dos números naturales con el operador *eq*, se «pelan» las expresiones:

```
succ( x ) = succ( y ) == x = y ;
succ( x ) = 0 == false ;
0 = 0 == true ;
x = y == y = x ;
```

Hasta aquí no se plantea ningún problema. Supóngase ahora que un operador *fac*: *Nat* → *Nat* se ha definido mediante las ecuaciones:

```
fac( 0 ) == succ( 0 ) ;
fac( succ(n) ) == succ( n ) * fac( n ) ;
```

donde *\** es el operador matemático de multiplicación, y hay que determinar si el término *fac(6) = fac(7)* pertenece o no a la misma clase de equivalencia que *true*. Esto requiere aplicar muchas ecuaciones (primero, muchas multiplicaciones, lo que exige aún más adiciones, y después hay que «pelar» *succ fac(6)* veces, según la primera ecuación mencionada más arriba). Por ello, un especificador avisado añadirá la ecuación:

```
fac( x ) = fac( y ) == x = y ;
```

Las consecuencias son drásticas: *true* y *false* son el mismo valor. La prueba se realiza así:

```
fac( 0 ) = fac(succ( 0 )) == succ( 0 ) = succ( 0 ) == true
```

pero, por otra parte,

```
fac(0) = fac(succ(0)) == 0 = succ(0) == false
```

y se sigue que

```
true == false
```

## I.5.2 Método básico de función-constructor

A continuación se presenta el método de función-constructor (CFM). El método se basa en una «declaración de intención» del especificador relativa a los valores de los géneros. En el método, los operadores y literales se dividen en «constructores» y «funciones», de ahí el nombre del método. Los términos «función» y «constructor» no tienen relación con ningún constructivo SDL, y sólo tienen significado en el contexto del método.

En los dos primeros párrafos se ofrece la definición de «constructor» y «función». En el tercero se presentan los primeros cuatro pasos del método (es decir, el CFM básico).

### I.5.2.1 Constructores

Al estudiar las especificaciones ADT, se observará que, en general, hay un número pequeño de operadores<sup>3)</sup> que juntos pueden generar al menos un término en *cada* clase de equivalencia *intencionada*. Un conjunto de operadores que «abarcan» un género se denominan los «constructores» de ese género.

#### Ejemplo

<u>Sort</u>	<u>Constructors</u>
Boolean	true, false
Integer	0, 1, plus, neg
Tree	empty, leaf, node
IntSet	EmptyIntSet, Insert

La primera oración de este párrafo contiene tres palabras fundamentales:

*cada*

No se debe omitir el operador *neg* en *Integer*: sin ese operador, no habría un término de constructores únicamente (término constructor) en la clase de equivalencia del término *minus (0, 1)*.

<sup>2)</sup> No se hace así en los géneros predefinidos.

<sup>3)</sup> En lo que sigue, los literales se consideran operadores sin argumentos.

# Reemplazada por una versión más reciente

*intencionada*

La subjetividad permanece, pero una vez que se han determinado los constructores, existe una «declaración de intención»: todos los demás operadores no generan valores que no puedan ser descritos mediante un término constructor.

*pueden*

El conjunto de constructores no es, en general, único. Para el género *Boolean* se podría haber seleccionado  $\{true, Not\}$  o  $\{false, Not\}$  como constructores.

No es posible dar demasiadas indicaciones para seleccionar el conjunto de constructores. En general, un número pequeño de constructores es mejor que uno grande. El número de constructores deberá ser, al menos, finito. Con frecuencia, se exige a los experimentadores que encuentren un conjunto de constructores para los que el método genera ecuaciones «intuitivamente atractivas». (Los expertos en ADT tienen cierta experiencia en la selección de un conjunto de constructores que genera ecuaciones «adecuadas», el resto es, en su mayor parte, intimidación.)

No se cumple que cada valor de un género se pueda describir de una manera única con algún término constructor, aunque ello simplifica la utilización del método. A modo de ilustración: resulta probable que se tenga la intención de que los términos

```
Insert( 1, Insert( 0, EmptyIntSet ) ) e
Insert( 0, Insert( 1, EmptyIntSet ) )
```

representen el mismo conjunto de enteros (por ejemplo,  $\{0, 1\}$ ).

## 1.5.2.2 Funciones

La definición de las funciones es sencilla: cada operador que no es un constructor es una función. No hay que dejarse confundir por el nombre: en el CFM, un literal (y, por consiguiente, constante) puede ser una función.

Es importante que durante la utilización no se cambien los conjuntos de constructores y funciones para los géneros. Por ejemplo, no se recomienda utilizar los constructores  $\{true, false\}$  en la especificación del género *Boolean*, cuando en la especificación de algún otro género  $\{true, Not\}$  se consideran los constructores.

## 1.5.2.3 El método

En 1.5.2.1 se señala que la selección de los constructores es una declaración de intención: una función (o una expresión con varias funciones) no debe generar valores que no puedan ser descritos mediante un término constructor. Se utilizará esta declaración de intención.

Los siguientes pasos se aplicarán a cada función, una por una.

### Paso 1

- Determinar para cada argumento de una función todas las formas posibles que puede tomar ese argumento si sólo se utilizan constructores con variables como argumento(s) solamente.

Obsérvese que, estrictamente, se debería utilizar «identificador de valor» y no «variable», puesto que el valor de una «variable» no varía.

**Ejemplo** – (al que se hará referencia en otros pasos). Supóngase que hay una función

```
X: Intset, Tree -> IntSet
```

(se podría pensar en el conjunto de enteros (resultado) que aparece tanto en el conjunto de enteros (argumento) como en el nodo-hoja del árbol). El primer argumento de *X* puede tener una de las formas siguientes:

```
EmptyIntSet,
Insert( j, s ) con j de la clase Integer y s de la clase IntSet.
```

El segundo argumento puede tener una de las formas siguientes:

```
empty,
leaf( j ) con j de la clase Integer,
node( b1, b2 ) con b1 y b2 de la clase Tree.
```

La expresión  $Insert(j1, EmptyIntSet)$  no pertenece a la lista de formas del primer argumento porque el constructor *Insert* tiene un constructor como argumento (es decir, *EmptyIntSet*). Por la misma razón,  $node(b3, leaf(0))$  tampoco puede ser una forma del segundo argumento.

# Reemplazada por una versión más reciente

## Paso 2

- Elaborar una lista de aplicaciones de la función con todas las formas de combinaciones posibles para los argumentos. Las variables de argumentos diferentes deben tener nombres diferentes (se denominarán de nuevo si es necesario), y todas las variables utilizadas se colocarán en una cualificación.

**Ejemplo** - Para la función  $X$  del ejemplo anterior, el **paso 2** arroja la siguiente lista cuantificada de aplicaciones:

```
for all j, j1, j2 in Integer,
      s in IntSet,
      b1, b2 in Tree
X( EmptyIntSet, empty )
X( EmptyIntSet, leaf( j ) )
X( EmptyIntSet, node( b1, b2 ) )
X( Insert( j, s ), empty )
X( Insert( j1, s ), leaf( j2 ) )
X( Insert( j, s ), node( b1, b2 ) )
```

Nótese que se ha denominado de nuevo  $j$  en la penúltima aplicación.

En una variante del CFM, el **paso 2** se sustituye por:

- Elaborar una aplicación dando variables para todos los argumentos de una función.

En este caso, los **pasos 3** y **4** (véase más abajo) no cambian, pero especialmente el **paso 4b** se aplicará más a menudo. La elección entre el CFM que se presenta aquí y esta variante es cuestión de gusto.

## Paso 3

- Tomar cada aplicación resultante del **paso 2** como el lado izquierdo de una ecuación, y escribir el lado derecho utilizando solamente:
  - 1) constructores,
  - 2) variables del lado izquierdo,
  - 3) funciones completamente definidas, y
  - 4) la función propiamente dicha, siempre y cuando las expresiones de los argumentos sean más pequeñas o iguales que las del lado izquierdo, y al menos una de las expresiones debe ser estrictamente más pequeña.
- Si no se consigue esto, se debe ir al **paso 4**.

No es fácil explicar lo que significan *más pequeña o igual y estrictamente más pequeña*. En general, significan que una expresión se ha obtenido por eliminación de algunos operadores, pero esto no es siempre válido. Lo fundamental es evitar las definiciones circulares.

**Ejemplo** - Si la interpretación de  $X$  es la que se sugiere en el **paso 1** (el conjunto de enteros (resultado) que aparece tanto en el conjunto de enteros (argumento) como en un nodo-hoja del árbol), pueden darse las ecuaciones siguientes. Se omite la cuantificación en aras de la brevedad.

```
X( EmptyIntSet, empty ) == EmptyIntSet ;
X( EmptyIntSet, leaf( j ) ) == EmptyIntSet ;
X( EmptyIntSet, node( b1, b2 ) ) == EmptyIntSet ;
X( Insert( j, s ), empty ) == EmptyIntSet ;
X( Insert( j1, s ), leaf( j2 ) ) == ? ;
X( Insert( j, s ), node( b1, b2 ) ) == Union(X(Insert( j, s ), b1),
                                           X(Insert( j, s ), b2 ) ) ;
```

Las primeras cuatro ecuaciones no plantean problemas. En la quinta, hay que distinguir entre los casos en que  $j1$  y  $j2$  son iguales y no iguales, y por tanto, se espera el **paso 4**.

La sexta ecuación utiliza la función *Union* plenamente definida con anterioridad y la llamada recursiva de  $X$ . Se permiten estas llamadas recursivas porque el primer argumento de  $X$  sigue siendo igual, mientras que el segundo argumento es estrictamente más pequeño (ambos con respecto al lado izquierdo).

## Paso 4

- Hay dos razones posibles para no poder proporcionar la segunda mitad de la ecuación en el **paso 3**:
  - 1) El lado derecho de una relación entre variables. Se ha de ir al **paso 4a**.
  - 2) El lado derecho depende de la estructura de una o más variables. Se ha de ir al **paso 4b**.

# Reemplazada por una versión más reciente

**Ejemplo** - En el caso de la quinta ecuación para  $X$ , el lado derecho depende de la relación entre las variables  $j1$  y  $j2$ .

## Paso 4a

- En este paso, una ecuación se divide en varias ecuaciones condicionales poniendo la relación requerida entre las variables en una condición. No se modifica el lado izquierdo de la ecuación obtenido en el **paso 2**.
- Escribir las relaciones entre variables como ecuaciones (no condicionales) y/o expresiones booleanas, utilizando solamente:
  - 1) constructores,
  - 2) variables,
  - 3) funciones completamente definidas, y
  - 4) la función propiamente dicha, siempre y cuando las expresiones de los argumentos sean más pequeñas o iguales que las del lado izquierdo, y al menos una de las expresiones debe ser estrictamente más pequeña.
- Verificar que las condiciones de una misma aplicación son complementarias, o que en el caso de condiciones que se superponen, el lado derecho es igual.

**Ejemplo** - Las condiciones requeridas para proporcionar los lados derechos para la aplicación.

```
X( Insert( j1, s ), leaf( j2 ) )
```

son

```
j1 = j2  
j1 /= j2
```

Estas condiciones son complementarias. Por tanto, se pueden formular las ecuaciones siguientes:

```
j1 = j2 ==> X( Insert( j1, s ), leaf( j2 ) ) == Insert( j1, EmptyIntSet )  
j1 /= j2 ==> X( Insert( j1, s ), leaf( j2 ) ) == X( s, leaf( j2 ) )
```

La primera ecuación condicional satisface las condiciones del **paso 3**, pues el lado derecho sólo utiliza constructores y una variable. La segunda ecuación condicional cumple las condiciones porque la llamada recursiva tiene como su primer argumento una expresión que es estrictamente más pequeña, y el segundo argumento no se amplió.

## Paso 4b

- En este paso, una ecuación se divide en varias ecuaciones sustituyendo una de las variables por varios términos. Determinar todas las formas de la variable de la que depende la estructura del lado derecho, como en el **paso 1**. Sustituir la variable por cada una de estas formas, de manera que los nombres de variables en esas formas no coincidan con nombres de variables que ya aparecen en el término de aplicación (es decir, el lado izquierdo). Añadir las nuevas variables introducidas en la cuantificación y completar las ecuaciones.

**Ejemplos** - Supóngase un operador  $Fib : Nat \rightarrow Nat$  ( $Fib$  viene de Fibonacci). Para  $Nat$  se supone que los constructores son el literal  $0$  y  $succ : Nat \rightarrow Nat$ . La aplicación de los **pasos 1, 2 y 3** produce:

```
for all n in Nat  
  Fib( 0 ) == 1 ;  
  Fib( succ( n ) ) == ? ;
```

Dado que  $Fib$  se define mediante  $Fib(0)=Fib(1)=1$  y  $Fib(n)=Fib(n-1)+Fib(n-2)$  para  $n>1$ , la segunda ecuación para  $Fib$  depende de la estructura de  $n$ , y se aplica el **paso 4b**. El resultado es:

```
for all n in Nat  
  Fib( succ( 0 ) ) == 1 ;  
  Fib( succ( succ( n ) ) ) == plus( Fib( succ( n ) ), Fib( n ) ) ;
```

El lado derecho de la última ecuación cumple las condiciones del **paso 3** porque «plus» es una función plenamente definida y los argumentos son llamadas recursivas de  $Fib$  con un argumento más pequeño que el argumento del lado izquierdo.

En algunos casos, hará falta aplicar más de una vez el **paso 4a** y/o el **paso 4b**.

Estos cuatro pasos forman el método función-constructor básico completo. Ha de destacarse que el CFM básico es seguro en lo que respecta a la compleción, pero a expensas de tiempo y papel. Con frecuencia, un grupo de ecuaciones se puede sustituir por una sola.

# Reemplazada por una versión más reciente

También se debe señalar que el CFM básico no genera ecuaciones cuando el operador más externo, en el lado derecho y en el izquierdo, es un constructor. Sin embargo, en algunas ocasiones estas ecuaciones resultan útiles.

## I.5.3 Cuatro pasos adicionales

Como ya se ha dicho, el CFM es seguro, pero produce más texto que el estrictamente necesario. Un buen método no sólo ofrece seguridad sino, también, un resultado legible. Los siguientes pasos están encaminados en ese sentido.

### I.5.3.1 Reducción de ecuaciones mediante variables adicionales

En el ejemplo ofrecido en I.5.2.3, parece que si el primer argumento de la función  $X$  es igual al conjunto vacío, el segundo argumento no influye en el lado derecho. Esto significa que las ecuaciones

```
X( EmptyIntSet, empty ) == EmptyIntSet ;
X( EmptyIntSet, leaf( j ) ) == EmptyIntSet ;
X( EmptyIntSet, node( b1, b2 ) ) == EmptyIntSet ;
```

se pueden resumir en:

```
X( EmptyIntSet, b ) == EmptyIntSet ;
```

donde  $b$  es una variable del género *Arbol (Tree)*.

#### Paso 5

- Sustituir un conjunto de ecuaciones con un lado derecho que no varía y con lados izquierdos con un argumento que varía por una ecuación en la que una variable sustituye al argumento que varía.

Es posible fusionar los **pasos 5 y 2**, pero generalmente a expensas de la seguridad. Si no se pueden reemplazar todas las formas por una variable, se puede obtener, por ejemplo:

```
X( EmptyIntSet, leaf( j ) ) == Insert( j, EmptyIntSet ) ;
X( EmptyIntSet, b ) == EmptyIntSet ;
```

En este caso, se puede obtener la equivalencia sustituyendo  $leaf(j)$ , por  $b$ .

```
Insert( j, EmptyIntSet ) == EmptyIntSet
```

Esto se puede interpretar como «Insert es una función», lo que contradice la «declaración de intención», pero tiene sentido, porque esa equivalencia significa que sólo hay un valor en *IntSet*.

### I.5.3.2 Reducción de las ecuaciones por conmutatividad

Si existe una función  $f$  que es conmutativa en un par de argumentos, entonces se puede reducir el número de ecuaciones.

#### Paso 6

- Si existe una función  $f$  para la cual la ecuación  $f(\dots, x, \dots, y, \dots) == f(\dots, y, \dots, x, \dots)$ ; (conmutatividad) se cumple para algún argumento  $x$  e  $y$ , luego
  - 1) añadir la ecuación de conmutatividad;
  - 2) buscar pares de ecuaciones en las que un lado izquierdo concuerda con el lado izquierdo de la ecuación de conmutatividad y el otro lado izquierdo concuerda con el lado derecho de la ecuación de conmutatividad, y suprimir una ecuación de cada par.

**Ejemplo** - Supóngase que hay que definir un operador  $eq : Tree, Tree \rightarrow Boolean$  para la igualdad entre árboles definida por el usuario. Después del **paso 2** se tienen las aplicaciones siguientes:

```
eq( empty, empty ) (1)
eq( empty, leaf( j ) ) (2)
eq( empty, node( b1, b2 ) ) (3)
eq( leaf( i ), empty ) (4)
eq( leaf( i ), leaf( j ) ) (5)
eq( leaf( i ), node( b1, b2 ) ) (6)
eq( node( b1, b2 ), empty ) (7)
eq( node( b1, b2 ), leaf( j ) ) (8)
eq( node( b1, b2 ), node( b3, b4 ) ) (9)
```



## Reemplazada por una versión más reciente

Después del **paso 3**, parece que  $eq(t1, t2) == eq(t2, t1)$ , y esta ecuación se inserta en el **paso 6**, mientras las ecuaciones 4, 7 y 8 se suprimen, pues forman pares con las ecuaciones 2, 3 y 6, respectivamente.

Se puede combinar el **paso 6** con el **paso 2**, pero a expensas de la seguridad.

### I.5.3.3 Reducción de ecuaciones mediante funciones adicionales

Si analizamos una vez más las ecuaciones de la función  $X$  en  $IntSet$ , veremos que su primer argumento aparece con frecuencia sin modificaciones en el lado derecho. Ello indica un posible cambio a una situación en la que primero alguna función opera sobre el segundo argumento antes de que su resultado se combine con el primer argumento. Para ser más concretos: para  $X$  se buscan funciones  $X1$  y  $X2$  de modo que

$$X(s, b) = X1(s, X2(b));$$

En este caso, la función  $Intersection : Intset, IntSet \rightarrow Intset$  para  $X1$  y  $Projection : Tree \rightarrow IntSet$  para  $X2$  serían adecuadas. Suponiendo que  $Intersection$  se especifica de alguna manera, esta división reduce el número de ecuaciones de siete (para  $X$ ) a cuatro (una para  $X$  y tres para  $Projection$ ).

#### Paso 7

- Si hay una función  $f$  en la que aparecen uno o más argumentos (casi) inalterados en los lados derechos, entonces tratar de encontrar nuevas funciones  $f1, \dots, fn, C$  de modo que
  - 1)  $f1, \dots, fn$  opere en (subconjuntos de) los argumentos cambiantes de  $f$ ,
  - 2)  $f$  se puede expresar en  $C$  aplicada a los argumentos no cambiantes de  $f$  y a los resultados de  $f1, \dots, fn$ .
- Las nuevas funciones que son «artificiales» se pueden ocultar a los usuarios de ADT mediante un signo de exclamación como último carácter en los nombres de función.

**Ejemplo** - Véase más arriba.

### I.5.3.4 Combinación de ecuaciones condicionales

En muchos casos, hay dos ecuaciones condicionales en una aplicación, y las condiciones son una expresión booleana y su negación. Estas se pueden combinar mediante un término condicional en el lado derecho.

#### Paso 8

- Si para alguna aplicación se utilizan ecuaciones condicionales, y hay dos ecuaciones condicionales con el formato:

$$\begin{aligned}c &==> lhs == rhs1 ; \\ \text{not}(c) &==> lhs == rhs2 ;\end{aligned}$$

estas ecuaciones condicionales se sustituyen por la ecuación no condicional

$$lhs == \text{if } c \text{ then } rhs1 \text{ else } rhs2 \text{ fi};$$

El término condicional del lado derecho de una ecuación no se debe utilizar en ninguna otra circunstancia que no sean las descritas en el **paso 8**. Ello obedece a la tendencia de programar en las ecuaciones, muchas veces con términos condicionales anidados, lo que dificulta su lectura y comprensión.

### I.5.4 Ecuaciones para constructores

Con el CFM básico no se puede generar ecuaciones con un constructor como el operador más externo (llamadas ecuaciones de constructor) en ambos lados del signo  $==$ . Sin embargo, en algunos casos es necesario expresar la equivalencia entre términos constructores.

Un ejemplo es el género  $Integer$  con los constructores  $0$  (literal),  $succ, neg : Integer \rightarrow Integer$ . Esto tiene en cuenta términos como  $succ(neg(...))$ . Sin embargo, es posible poner un término constructor en cada clase de equivalencia si se utiliza  $neg$  como máximo una vez como el operador más externo. Esto se puede utilizar también para simplificar algunas ecuaciones de funciones.

#### Paso 9

- Para aquellos constructores  $C$  para los que se quiere formular ecuaciones de constructor, se crea una función  $C^f$ . A estas funciones se les aplica el CFM sin este **paso 9**. Después, se suprimen las marcas  $^f$  y las ecuaciones que tienen el mismo lado derecho e izquierdo.

El resultado del **paso 9** es seguro, pero generalmente se generan ecuaciones superfluas (pero inocuas).

# Reemplazada por una versión más reciente

El **paso 9** se presenta como el último paso del CFM, porque comenzar marcando constructores como funciones sería bastante confuso. Sin embargo, una vez que el lector se haya familiarizado con el CFM, se recomienda que formule ecuaciones para los constructores antes de definir las funciones, porque esas ecuaciones muestran bien la estructura de los términos constructores.

**Ejemplo** - En el caso de *Integer*, las ecuaciones de constructor para *neg* y *succ* tienen sentido. La aplicación normal del CFM da:

```
negf( 0 )           == 0 ;
negf( succ( n ) )   == neg( succ( n ) ) ;
negf( neg( n ) )    == n ;
negf( 0 )           == succ( 0 ) ;
succf( succ( n ) )  == succ( succ( n ) ) ;
succf( neg( 0 ) )   == succ( 0 ) ;
succf( neg( succ( n ) ) ) == neg( n ) ;
succf( neg( neg( n ) ) ) == succ( n ) ;
```

Los otros subpasos del **paso 9** dan:

```
neg( 0 )           == 0 ;
neg( neg( n ) )    == n ;
succ( neg( 0 ) )   == succ( 0 ) ;
succ( neg( succ( n ) ) ) == neg( n ) ;
succ( neg( neg( n ) ) ) == succ( n ) ;
```

Las ecuaciones tercera y quinta de la última lista son superfluas, puesto que están implicadas en las ecuaciones primera y segunda, respectivamente.

## I.5.5 Limitaciones

En estas Directrices, la presentación del CFM se hace en el momento en que se decide construir un género a partir de cero. El método no da ninguna directriz sobre cuándo hacerlo, ni cuándo basar una especificación de género en géneros predefinidos más el constructor de género *Structure*. Esto rebasa el alcance del CFM.

Dentro del alcance del CFM estaría, por ejemplo, la utilización de **error!** y **nameclass**, pero no se trata de eso aquí. Una cuestión de más envergadura es que el CFM presentado se basa en la teoría de los sistemas de reescritura reductivos, véase [6]. Se trata de algo bastante restrictivo. Considérese, por ejemplo, la función *Qsort* : *IntList* → *IntList* (quicksort) con las ecuaciones:

```
Qsort( EmptyIntList ) == EmptyIntList ;
split( il, i ) == pair( l1, l2 ) ==>
  Qsort( MkString( i ) // il ) == Qsort( l1 ) // MkString(i) // Qsort( l2 ) ;
```

donde *pair* es un constructor para pares de *IntLists*, y *split* divide su primer argumento en una lista con enteros menores que su segundo argumento y en una lista con enteros mayores que su segundo argumento.

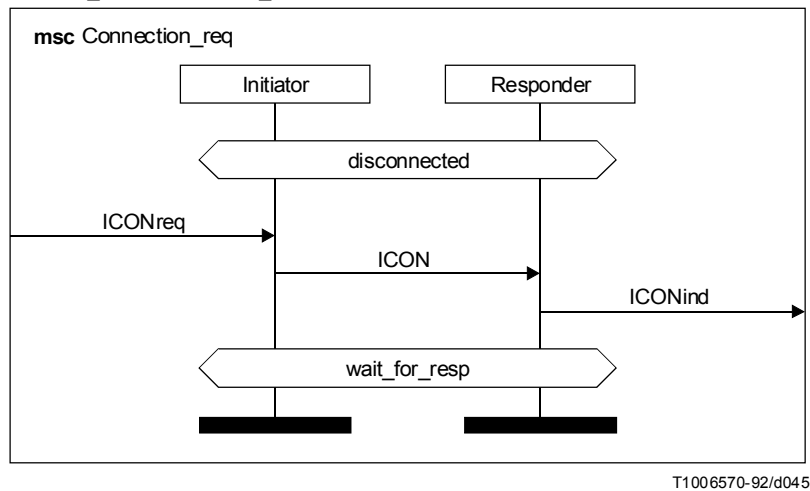
Esta definición de *Qsort* sería perfectamente correcta en la teoría de los sistemas de reescritura casi reductivos, pero con el CFM esta clase de ecuaciones «elegantes» no se producirán nunca probablemente.

## I.6 Utilización de los gráficos de secuencias de mensajes

### I.6.1 Introducción

En el ciclo de vida del sistema, se presta cada vez más atención a la etapa de especificación del sistema, ya que de ella depende la calidad de las etapas siguientes. En particular, en el campo de las telecomunicaciones, esto se ha tenido en cuenta mediante la utilización del SDL. Además de efectuar una prueba de corrección general (por ejemplo, ausencia de atascos, hay que verificar la coherencia de la especificación SDL con respecto al comportamiento requerido del sistema. Una manera conveniente de describir este comportamiento la proporcionan los trazados del sistema, adecuadamente presentados en forma de gráficos de secuencias de mensajes (MSC, *message sequence charts*). Estos gráficos son medios generalizados para describir y, en particular, visualizar gráficamente los trazados de sistema seleccionados dentro de los sistemas distribuidos, especialmente los sistemas de telecomunicación. Un MSC muestra las secuencias de mensajes intercambiados entre entidades (tales como servicios, procesos, bloques SDL) y su entorno (véase el Ejemplo I.6-1). Desde el punto de vista formal, un MSC describe la ordenación parcial de los eventos, es decir, el envío y el consumo de mensajes, véase [5].

## Reemplazada por una versión más reciente



EJEMPLO I.6-1

### Gráfico de secuencias de mensajes

Dado que cada secuencia de un MSC describe un trazado de sistema, se puede obtener un MSC de una especificación de sistema SDL existente. Sin embargo, el MSC se crea normalmente antes de la especificación de sistema, y sirve como:

- una declaración de requisitos para las especificaciones SDL;
- una base para la generación automática de esquemas de especificaciones SDL;
- una base para seleccionar y especificar los casos de prueba;
- una especificación semiformal de comunicación; o
- una especificación de interfaz.

El MSC del Ejemplo I.6-1 describe una pieza seleccionada del trazado de establecimiento de la conexión de la especificación de servicio INRES, véase [12]. Igualmente, se podría representar mediante diagramas de proceso SDL con ciertas adiciones o modificaciones (véase la Figura I.6-1), en la que las ramas o bifurcaciones no atravesadas se indican con trazo interrumpido, y el flujo de la señal, con flechas realizadas.

El diagrama de la Figura I.6-1 contiene por lo menos la misma información que el MSC del Ejemplo I.6-1. No obstante, es evidente que el MSC es más útil en este contexto, puesto que se centra en la información pertinente, es decir, las entidades (*Initiator*, *Responder*), y las señales incluidas en la parte de trazado seleccionado (*ICONreq*, *ICON*, *ICONind*). Además, un MSC puede describir la interacción de entidades que se encuentran en diferentes niveles de abstracción.

La comparación entre el Ejemplo I.6-1 y la Figura I.6-1 da una buena idea intuitiva del significado de un MSC. También demuestra que un MSC que describe un escenario posible también puede considerarse como un esquema de especificación SDL.

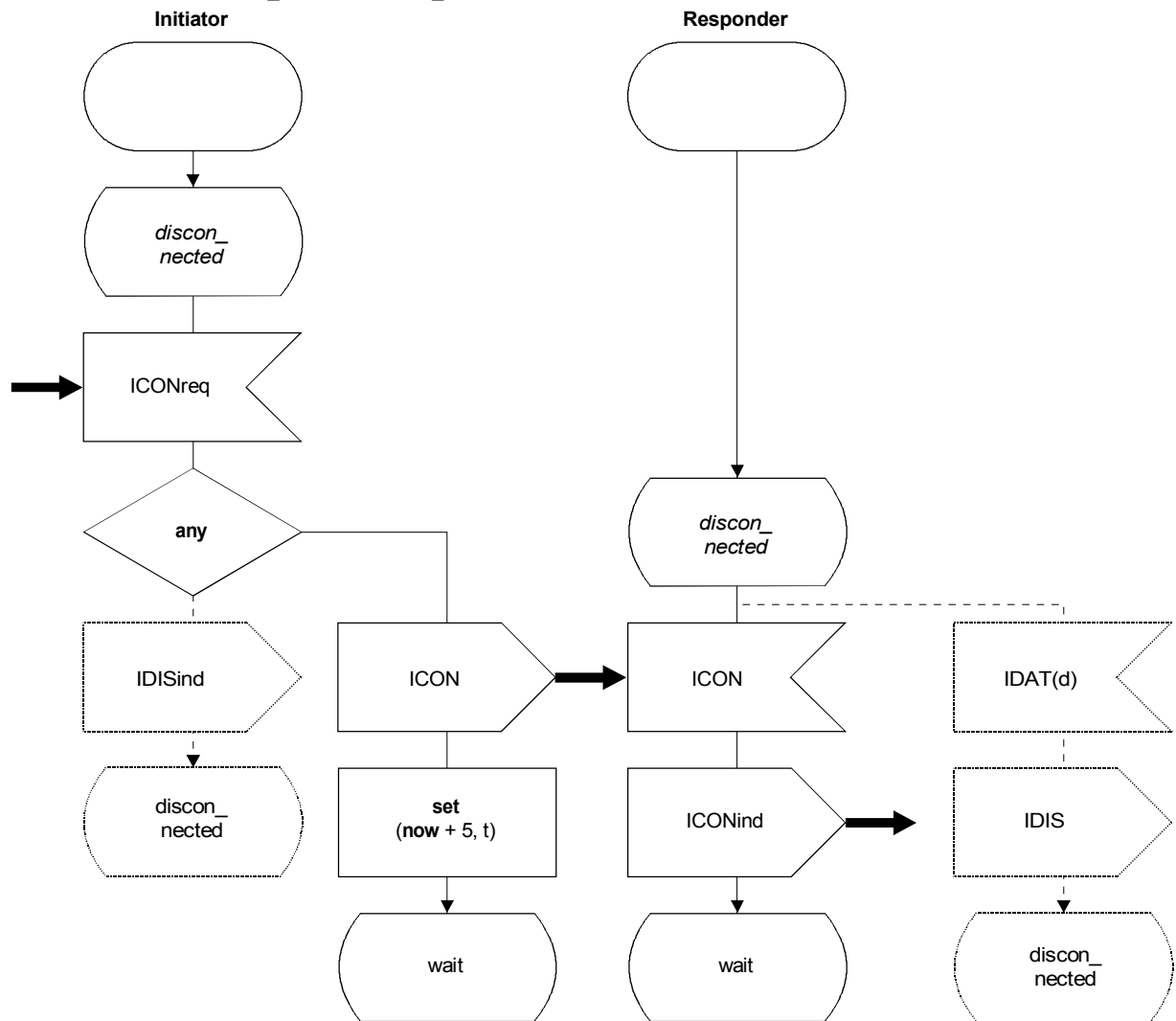
Los constructivos más básicos del lenguaje son *instancias* y *mensajes*, que se intercambian entre sí. En la forma gráfica, las instancias y los entornos están representados por líneas verticales o por columnas. Los mensajes se representan mediante líneas horizontales dirigidas, que se pueden inclinar para indicar que los mensajes se adelantan o cruzan. La flecha de mensaje indica el consumo de señal, y el extremo opuesto (origen del mensaje), el envío de señal.

En general, un MSC describe una pequeña parte del trazado completo del sistema. Por ello, es necesario caracterizar al MSC mediante la especificación de sus *condiciones* iniciales y finales y, quizás, las intermedias. Gráficamente, una condición se representa con un hexágono, que contiene el nombre de la condición (véase la condición inicial *desconectado* del Ejemplo I.6-1). Además, un MSC contiene *acciones*, provocadas por el consumo de señales y *temporizaciones*.

### I.6.2 Especificación del sistema mediante mecanismos de composición de MSC

Los MSC se utilizan principalmente como un lenguaje de requisitos, para describir el objetivo de un sistema en forma de ejemplos de trazado. A continuación, se ofrece una metodología MSC sistemática, basada en mecanismos de composición. Las condiciones introducidas a ese efecto se pueden utilizar también para obtener especificaciones SDL, esquemáticas.

# Reemplazada por una versión más reciente



T1006580-92/d046

FIGURA I.6-1/Z.100

## Diagramas SDL de flujo de señales combinado (informal)

Puesto que un MSC sólo describe un comportamiento parcial del sistema, es conveniente utilizar varios MSC sencillos, que se puedan combinar de maneras diferentes. (Esto se puede hacer efectivamente, y es lo que aquí se denomina «composición», o se puede considerar meramente como un orden de interpretación.) Los MSC se pueden componer mediante la identificación (por nombre) de condiciones iniciales y finales. A la inversa, los MSC pueden descomponerse en condiciones intermedias.

La composición y descomposición de los MSC cumplen las siguientes reglas de condiciones globales y no globales, según las cuales las condiciones globales se refieren a todas las instancias involucradas en el MSC y las condiciones no globales se asocian a un subconjunto de instancias.

### I.6.2.1 Composición de los MSC

#### Condiciones globales

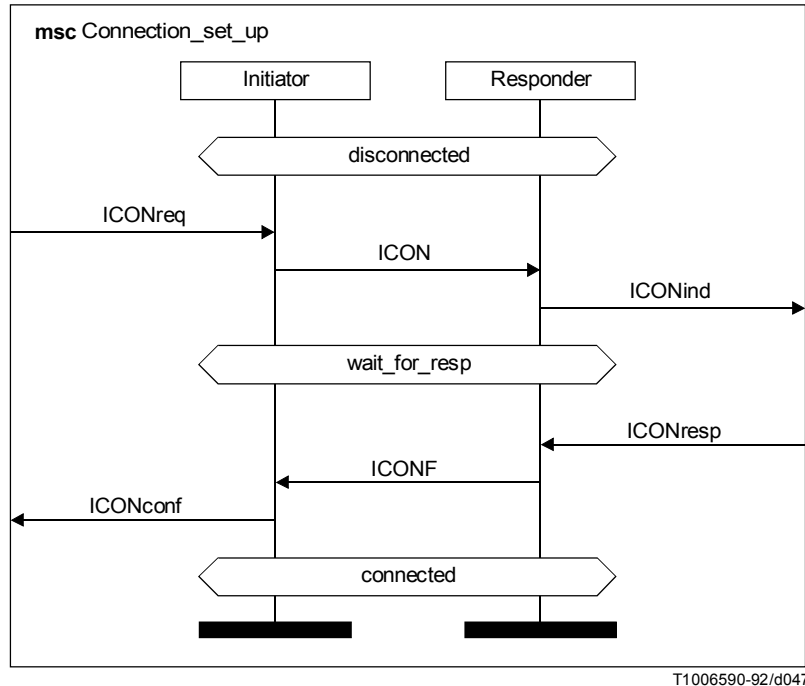
Dos gráficos de secuencias de mensajes, *MSC1* y *MSC2*, se pueden componer si ambos contienen el mismo conjunto de instancias, y si la condición inicial de *MSC2* corresponde a la condición final de *MSC1*, según la identificación por nombre. La condición final de *MSC1* y la condición inicial de *MSC2* se convierten en una condición intermedia en el MSC compuesto. Simbólicamente:

$$\begin{array}{l} \text{MSC1} = \text{MSC1}' \text{ Condition} \\ \text{MSC2} = \text{Condition MSC2}' \\ \hline \text{MSC1} * \text{MSC2} = \text{MSC1}' \text{ Condition MSC2}' \end{array}$$

## Reemplazada por una versión más reciente

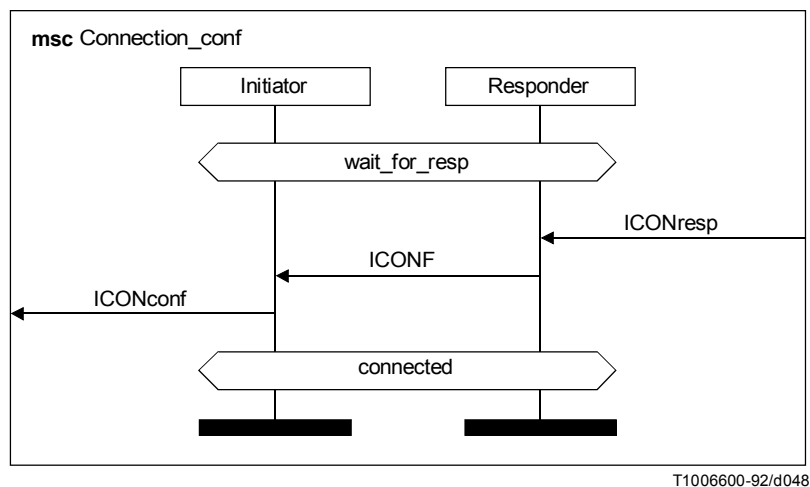
La primera ecuación significa que *MSC1* se puede escribir como una sección de *MSC*, *MSC1'*, y una condición final siguiente. La segunda ecuación dice que *MSC2* comienza con una condición inicial, seguida de la sección de *MSC* *MSC2'*. La tercera ecuación establece la composición de *MSC1* y *MSC2* (el símbolo de asterisco representa la composición). El *MSC* compuesto se puede escribir en forma de una sección de *MSC* del comienzo, *MSC1'*, una condición intermedia y la subsiguiente sección de *MSC*, *MSC2'*.

*MSC Connection\_set\_up* del Ejemplo I.6-2 es una composición de los *MSC Connection\_req* del Ejemplo I.6-1 y *Connection\_conf* del Ejemplo I.6-3.



EJEMPLO I.6-2

**MSC compuesto, basado en los Ejemplos I.6-1 y I.6-3**



EJEMPLO I.6-3

**MSC Connection\_conf**

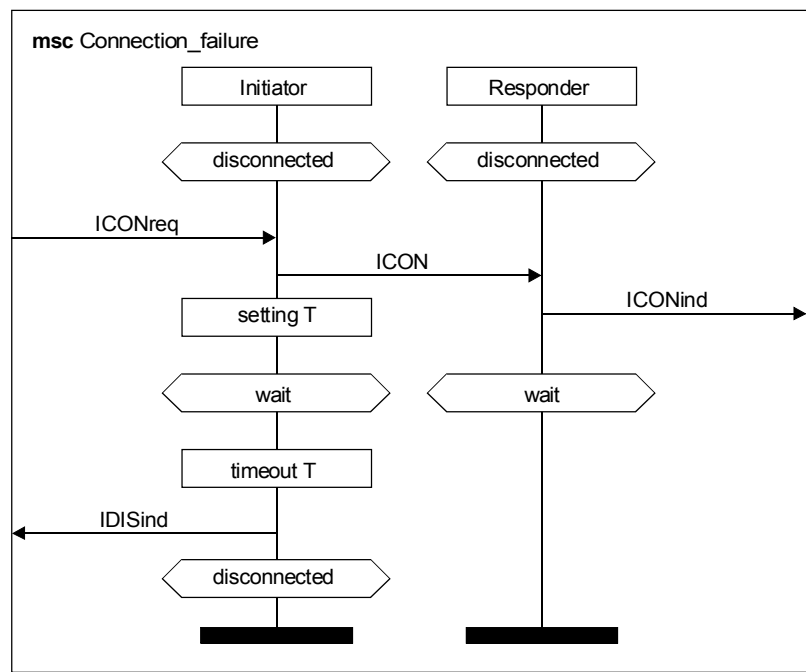
# Reemplazada por una versión más reciente

## Condiciones no globales

Dos gráficos de secuencias de mensajes, *MSC1* y *MSC2*, se pueden componer mediante condiciones no globales si, para cada instancia (*I*) que los dos MSC tienen en común, *MSC1* termina con una condición no global y *MSC2* comienza con una condición no global correspondiente. Además, cada condición no global de *MSC2* tiene que tener su correspondiente condición no global en *MSC1*. Cuando  $I(MSC_i)$  ( $i = 1, 2$ ) denota la restricción de un *MSC\_i* con respecto a los eventos de instancia *I*, esto se puede escribir simbólicamente así:

$$\begin{aligned} I(MSC1) &= I(MSC1)' \text{ Condition} \\ I(MSC2) &= \text{Condition } I(MSC2)' \\ \hline I(MSC1) * I(MSC2) &= I(MSC1)' \text{ Condition } I(MSC2)' \end{aligned}$$

Por ejemplo, el MSC *Connection\_failure* (véase el Ejemplo I.6-4) es una composición de MSC *Connection-request* (véase el Ejemplo I.6-5a) y *Timeout* (véase el Ejemplo I.6-5c) mediante la condición local *wait*. El MSC *Connection\_request* contiene dos instancias: *Initiator* y *Responder*. A cada una de las instancias se conecta una condición local (final) *wait*. Hay que tener presente que las dos condiciones locales *wait* con nombres idénticos son diferentes, y son discriminadas por las instancias a las que están asociadas. El MSC *Timeout* contiene sólo una instancia: *Initiator*, a la que se une la condición local inicial. La composición de MSC *Connection\_request* con el MSC *Timeout* sólo se refiere a la instancia *Initiator*, es decir, MSC *Connection\_request* se continúa a lo largo de la instancia *Initiator* mediante MSC *Timeout*. Esto también muestra la utilidad de las condiciones no globales, que permiten hacer una composición con respecto a un subconjunto de las instancias involucradas en el MSC. Obsérvese, también, que en el Ejemplo I.6-4, la fijación del temporizador *T* y la temporización (timeout) se indican como acciones, a efectos de compatibilidad con los Ejemplos I.6-5a y I.6-5c.



EJEMPLO I.6-4  
MSC *Connection\_failure*

# Reemplazada por una versión más reciente

## I.6.2.2 Descomposición de los MSC

### Condiciones globales

Una condición intermedia permite descomponer un MSC,  $MSC3'$ , dividiéndolo en la condición intermedia en  $MSC1$  y  $MSC2$ ; la condición intermedia se convierte en una condición final para  $MSC1$  y una condición inicial para  $MSC2$ :

$$\underline{MSC3 = MSC1' \text{ Condition } MSC2'}$$

$$MSC1 = MSC1' \text{ Condition}$$

$$MSC2 = \text{Condition } MSC2'$$

### Condiciones no globales

Un subconjunto de condiciones intermedias no globales permite descomponer un MSC,  $MSC3'$ , en  $MSC1$  y  $MSC2$ , si todas las condiciones no globales de ese subconjunto se refieren a instancias diferentes, y si no se corta ningún mensaje en parte por la descomposición, es decir, que tanto la entrada de mensaje y su salida correspondiente pertenecen a  $MSC1$  o a  $MSC2$ :

$$\underline{I(MSC3) = I(MSC1)' \text{ Condition } I(MSC2)'}$$

$$I(MSC1) = I(MSC1)' \text{ Condition}$$

$$I(MSC2) = \text{Condition } I(MSC2)'$$

Por ejemplo, *MSC Connection\_failure* del Ejemplo I.6-4 se puede descomponer en *MSC Connection\_request* del Ejemplo I.6-5a, y *Timeout* del Ejemplo I.6-5c en la condición local *wait*.

## I.6.2.3 Normalización de gráficos de secuencias de mensajes

Por lo común, un conjunto de MSC elaborado en una etapa preliminar del diseño está poco estructurado. Resulta difícil tener una visión general adecuada y estimar el alcance de todos los trazados posibles del sistema. Por otra parte, los MSC grandes tienen, con frecuencia, muchos subtrazados cíclicos en común, situación que se podría evitar mediante una regla de descomposición apropiada.

Para solucionar estas deficiencias, se pueden construir los denominados *MSC normalizados*, a partir de un conjunto determinado de MSC, aplicando las reglas de descomposición y composición definidas más adelante. Esos MSC normalizados representan bloques de construcción «normalizados». Describen el mismo comportamiento del sistema que el conjunto original de MSC del que se obtienen, es decir, que de ellos se pueden derivar los mismos trazados de sistema, teniendo en cuenta las reglas de composición. En este sentido, el conjunto derivado de MSC normalizados es equivalente al conjunto original de MSC.

Los bloques de construcción normalizados y el método correspondiente de composición estructurada ya se han propuesto para los procesos de red Petri, mediante los denominados *periodos de proceso*.

Básicamente, los MSC normalizados son gráficos de secuencias de mensajes máximos que son inseparables, o sea, siempre aparecen como un todo, sin inserciones cíclicas. «Máximo» en este contexto significa que no existe un MSC más grande con las propiedades enumeradas que contenga al MSC como un subtrazado. Los MSC normalizados describen trazados secuenciales o trazados cíclicos, que no contienen subtrazados cíclicos procedentes de estados intermedios.

Los MSC normalizados se definen de tal forma que resulta posible generarlos automáticamente a partir de un conjunto determinado de MSC en cada etapa de la especificación del sistema. Así, los MSC normalizados representan bloques de construcción, que se pueden utilizar por separado y en paralelo con el conjunto de MSC originalmente especificado. Se recomienda especificar los MSC desde el comienzo mismo, en forma estructurada y ajustándose, en la medida de lo posible, a las reglas que rigen los MSC normalizados. Es posible que, en la práctica, este método imponga al usuario restricciones innecesarias e inconvenientes, puesto que los MSC normalizados se refieren al sistema como un todo, mientras que en las etapas preliminares sólo suelen especificarse partes del sistema.

La composición de MSC normalizados mediante computador permite efectuar una simulación inmediata del comportamiento especificado. Además, se pueden utilizar los MSC normalizados para generar y seleccionar casos de prueba. Considerar los MSC normalizados como las unidades más pequeñas, permite hacer abstracción de los detalles de comunicación y analizar el comportamiento esencial del sistema. Los MSC normalizados también son de interés para el análisis del sistema puesto que, al representar los aspectos concurrentes del sistema de manera directa, proporcionan una alternativa al gráfico de alcanzabilidad.

A continuación se ofrecen, esquemáticamente, los pasos del procedimiento para normalizar MSC, suponiendo un estado inicial distinguido del sistema.

# Reemplazada por una versión más reciente

## Paso 1 – Conjunto inicial de MSC

- Los MSC seleccionados para un sistema se especifican en la etapa de definición de los requisitos. Las relaciones entre estos MSC se especifican mediante condiciones, que permiten las composiciones y descomposiciones correspondientes.

Para ilustrar los pasos, se utiliza la especificación de requisitos de servicio INRES, véase [12], que muestra un establecimiento (simplificado) de conexión y la consiguiente transferencia de datos entre *Initiator* y *Responder*. Un medio no fiable puede hacer fallar la transferencia de señales. Por razones prácticas, el conjunto inicial de MSC elegido es idéntico al conjunto de MSC atómicos que se define en el **paso 2**.

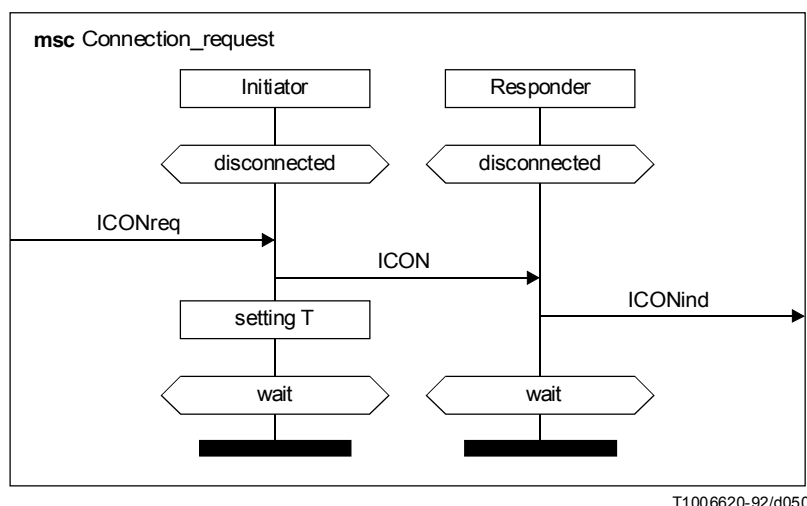
## Paso 2 – Descomposición en MSC atómicos

- Descomponer los MSC dados en MSC «atómicos», es decir, MSC que no contienen condiciones intermedias.

Los MSC atómicos de los Ejemplos I.6-5a a I.6-5g muestran únicamente los casos sin fallos con respecto al medio, o sea, casos en los que no se han perdido señales. La temporización del Ejemplo I.6-5c obedece al hecho de que el respondedor no responde a tiempo, pero no a una pérdida de señales.

Los MSC están estrechamente relacionados con los cronogramas utilizados en el modelo de referencia básico OSI. Sin embargo, los MSC contienen más información: además de acciones y temporizaciones, también contienen *condiciones*, que proporcionan la base de los mecanismos de composición.

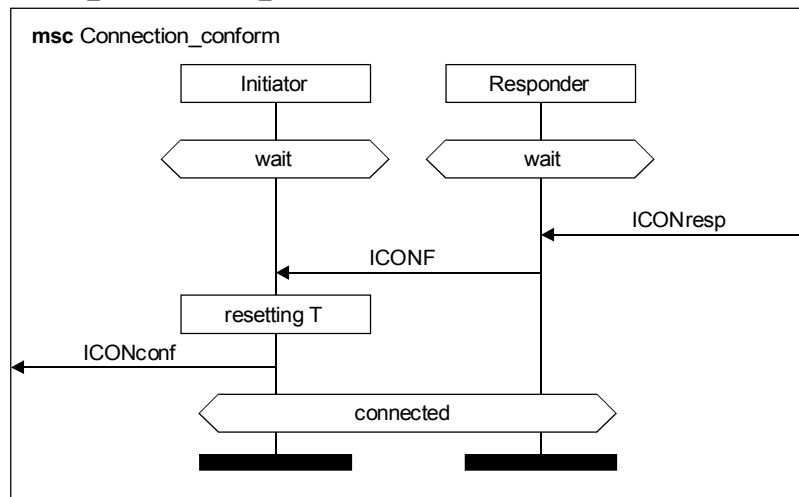
En los ejemplos se utilizan condiciones globales y locales. Las condiciones locales se emplean para facilitar la continuación de un MSC por otro MSC con respecto a un subconjunto de instancias. Por ejemplo, MSC *Connection\_request* del Ejemplo I.6-5a puede ser continuado por MSC *Timeout* del Ejemplo I.6-5c con respecto a la instancia *Initiator*. Las condiciones globales, por ejemplo, *connected*, se utilizan para una composición global, es decir, una composición con respecto a todas las instancias contenidas. Obsérvese que la fijación del temporizador *T* y la reiniciación y temporización correspondientes se encuentran en diferentes MSC, por lo que se deben indicar como acciones.



EJEMPLO I.6-5a  
MSC atómico



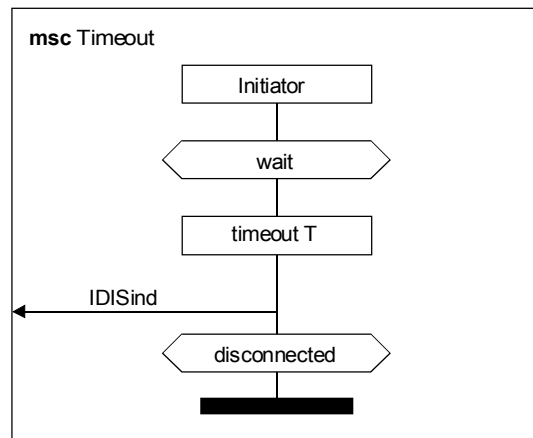
# Reemplazada por una versión más reciente



T1006630-92/d051

EJEMPLO I.6-5b

MSC atómico

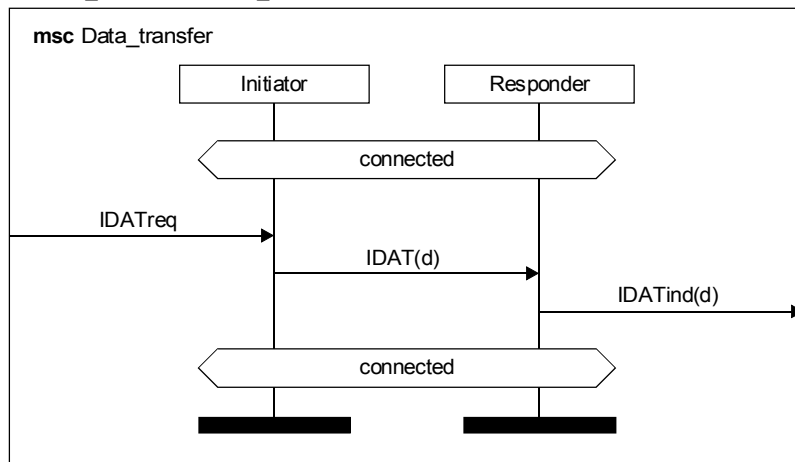


T1006640-92/d052

EJEMPLO I.6-5c

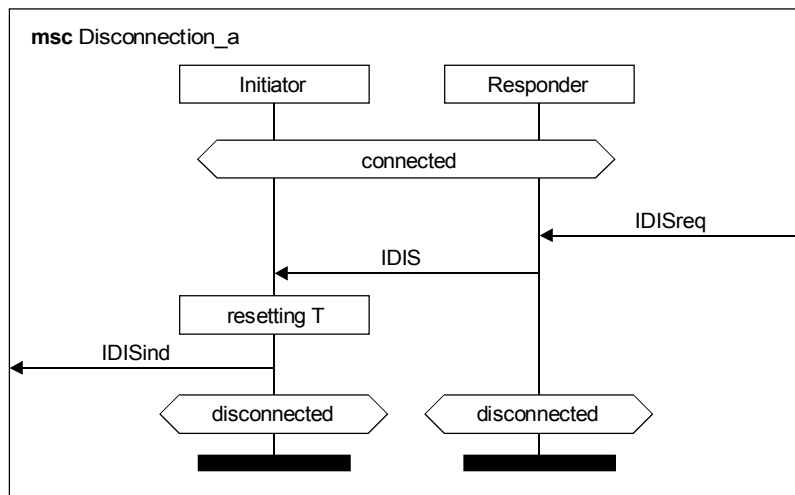
MSC atómico

# Reemplazada por una versión más reciente



T1006650-92/d053

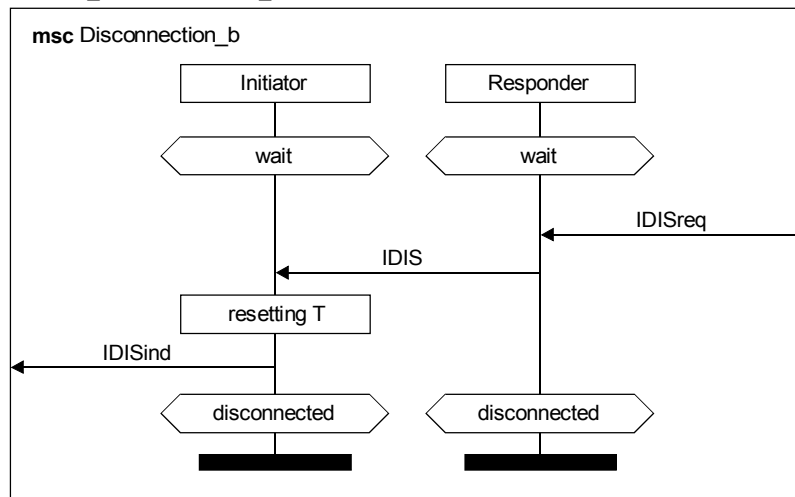
EJEMPLO I.6-5d  
MSC atómico



T1006660-92/d054

EJEMPLO I.6-5e  
MSC atómico

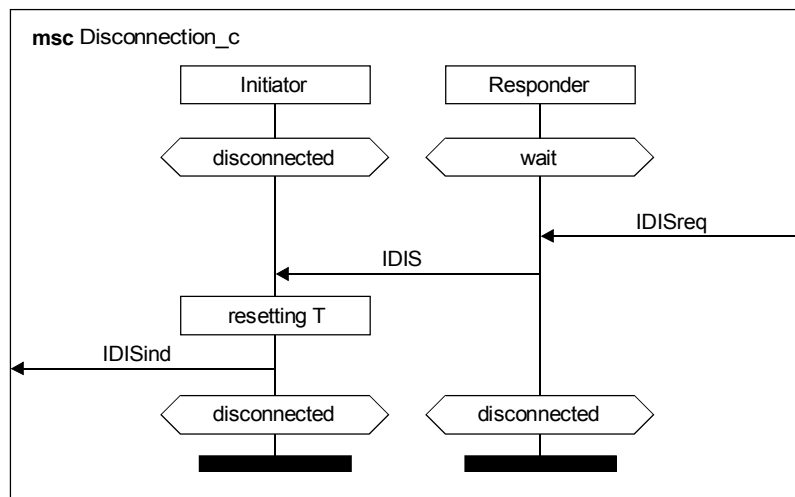
# Reemplazada por una versión más reciente



T1006670-92/d055

EJEMPLO I.6-5f

MSC atómico



T1006680-92/d056

EJEMPLO I.6-5g

MSC atómico

# Reemplazada por una versión más reciente

## Paso 3 – Creación de un diagrama general de MSC

- Comenzando por el estado inicial y teniendo en cuenta las reglas de composición de MSC, se pueden reagrupar los MSC atómicos. El conjunto de secuencias obtenido de MSC se puede presentar en forma de *diagrama general MSC* en el que se identifican los estados de sistema alcanzados, suministrados por las condiciones del MSC.

En la Figura I.6-2 aparece el diagrama general de MSC obtenido a partir del conjunto de MSC atómicos de los Ejemplos I.6-5a a I.6-5g. Los nodos representan estados globales del sistema. El estado inicial *disconnected* está marcado. En general, este tipo de diagrama puede resultar útil para mostrar la relación/conexión entre varios MSC. Un diagrama general de MSC puede considerarse como diagrama auxiliar de los MSC, y corresponde al diagrama general de estados del SDL.

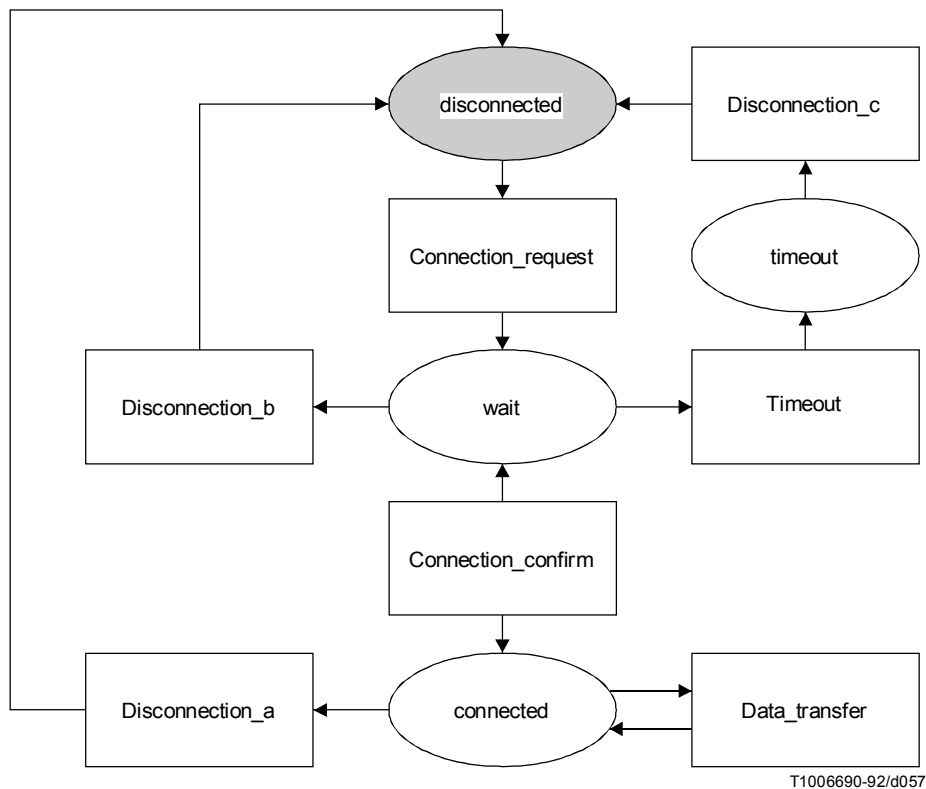


FIGURA I.6-2/Z.100  
Diagrama general de MSC

## Paso 4 – División del diagrama general de MSC

- El diagrama general de MSC se divide en el nodo asignado al estado inicial. A partir de ese nodo, el diagrama se sigue dividiendo en los nodos de los que arrancan subtrazados cíclicos. A cada borde conectado a uno de esos nodos se une la copia correspondiente de ese nodo después de la descomposición.

## Paso 5

- Los trazados máximos dentro de los fragmentos del diagrama general de MSC forman los MSC normalizados.

De la Figura I.6-3 se obtienen cinco MSC normalizados, de los que *Data\_transfer* y *Disconnection\_a* son los mismos del Ejemplo I.6-5. Los nuevos MSC normalizados se presentan en los Ejemplos I.6-6a a I.6-6c.

## Reemplazada por una versión más reciente

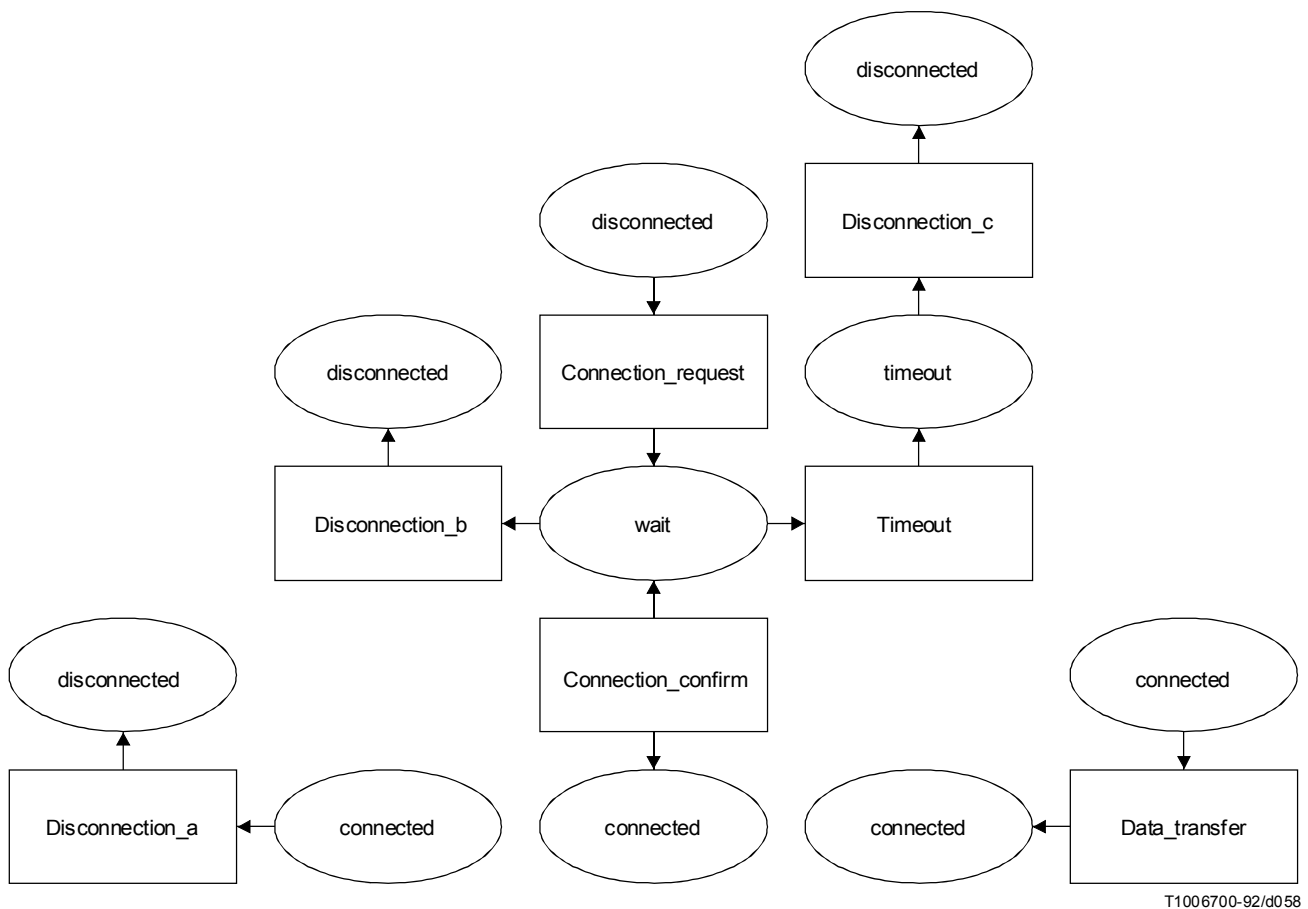
Para conseguir una visión general de todos los posibles trazados del sistema, hay que mostrar la relación/conexión entre los MSC normalizados derivados. Esto se hace en la Figura I.6 4, en la que, una vez más, los MSC normalizados se representan como transiciones dentro de un diagrama general de MSC.

Una comparación entre las Figuras I.6-2 e I.6-4 demuestra la capacidad estructuradora de los MSC normalizados. La utilidad de los MSC normalizados es aún más evidente cuando se trata de ejemplos más reales, que contienen grandes trazados de sistema.

### I.6.3 Método para desarrollar sistemas basado en MSC, SDL y descomposición funcional

Puesto que los MSC son una herramienta importante en la etapa de definición de requisitos, y aunque en la práctica industrial se utiliza el SDL sobre todo en el diseño de sistemas, parece obvio que hay que obtener la especificación SDL a partir de un conjunto de MSC. Sin embargo, ese enfoque se puede ejecutar en la práctica automáticamente sólo hasta cierto nivel. Después, habrá que refinar manualmente la especificación SDL. Además, esa cronología estricta (ciclo de vida «cascada») presenta una visión simplificada del diseño de sistemas. El SDL y el MSC se deben utilizar simultáneamente en diferentes etapas del desarrollo de sistemas de manera que se complementen entre sí y se correlacionen de muchas maneras. A continuación, se ofrece una metodología más perfeccionada que sustenta el diseño de arriba a abajo, basado en conceptos de refinamiento.

Se describe un sistema de telecomunicación desde tres puntos de vista diferentes pero relacionados entre sí: *funcional*, *de comportamiento* y *arquitectural*. El aspecto funcional se puede modelar mediante una jerarquía de funciones; el aspecto de comportamiento, mediante gráficos de secuencias de mensajes; y el aspecto arquitectural, mediante el SDL. Cabe señalar que en esta tarea esquemática sólo se manifiesta el énfasis principal, es decir, en cada clase de descripción se presenta un aspecto con toda claridad.

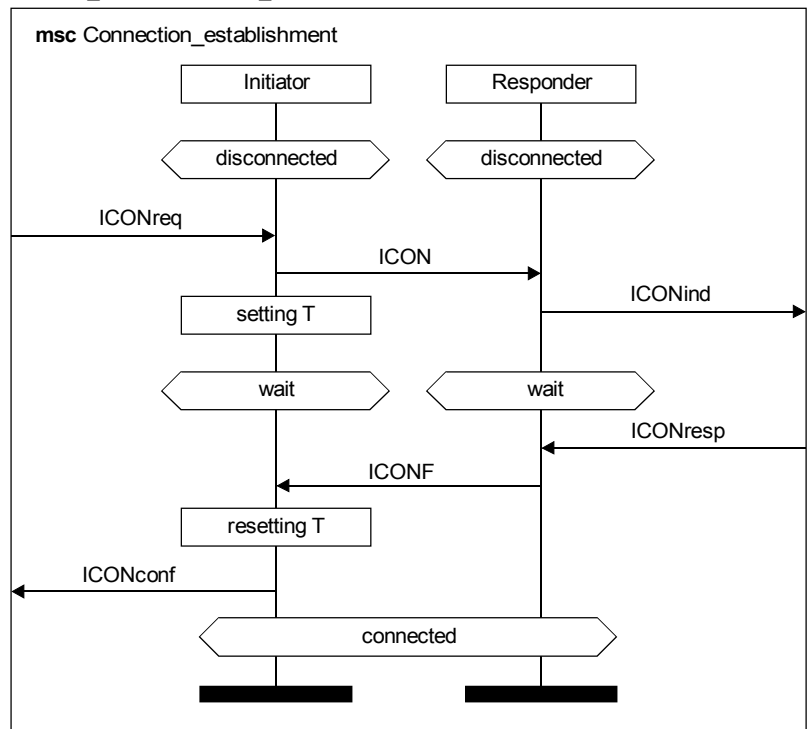


T1006700-92/d058

FIGURA I.6-3/Z.100

División del diagrama general de MSC

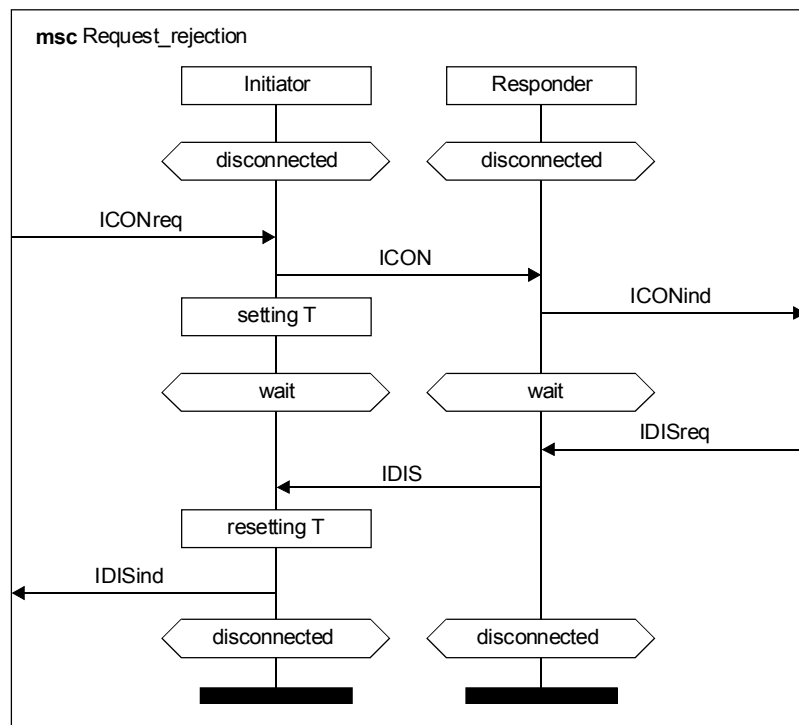
# Reemplazada por una versión más reciente



T1006710-92/d59

EJEMPLO I.6-6a

MSC normalizado

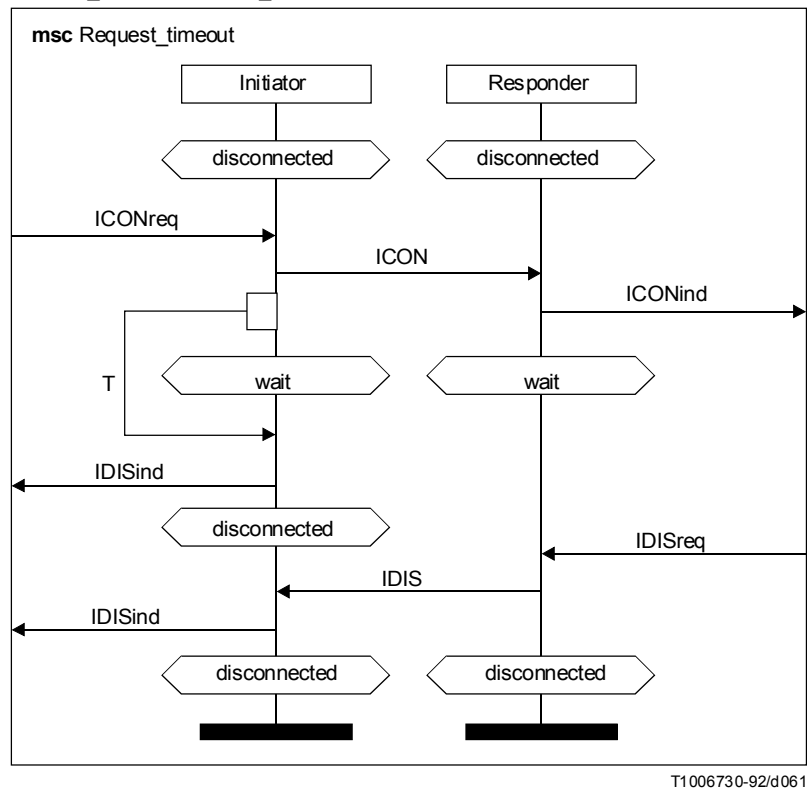


T1006720-92/d060

EJEMPLO I.6-6b

MSC normalizado

# Reemplazada por una versión más reciente



EJEMPLO I.6-6c  
MSC normalizado

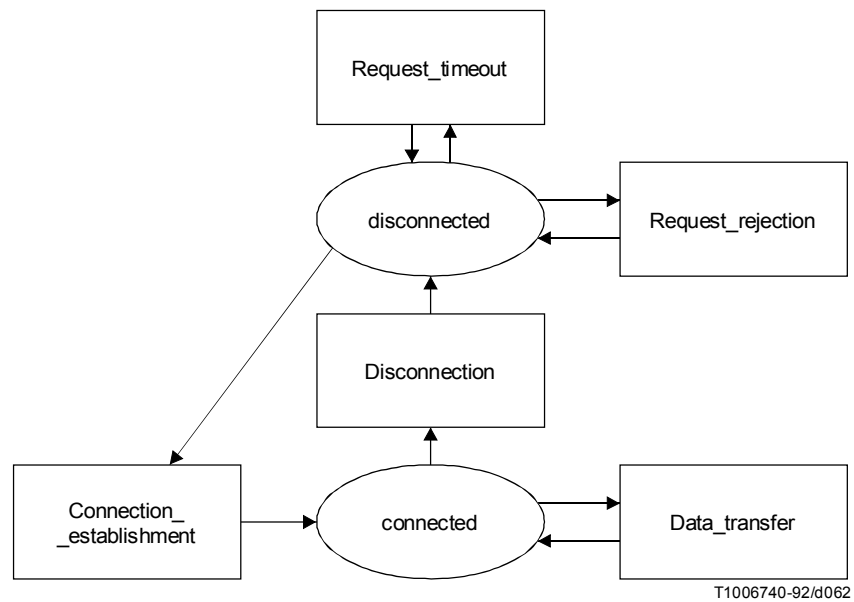


FIGURA I.6-4/Z.100  
Diagrama general para MSC normalizados

# Reemplazada por una versión más reciente

## I.6.3.1 Descomposición funcional

Las funciones se utilizan para la especificación funcional del sistema, y se puede distinguir entre:

- *funciones globales*, cada una de las cuales es descompuesta en una jerarquía de funciones de nivel más bajo;
- *funciones abstractas*, que representan nodos intermedios en diferentes niveles de la jerarquía de funciones;
- *funciones terminales*, que conforman las hojas de la jerarquía de funciones y, por ello, no pueden descomponerse más. Representan funciones muy precisas, y pueden ser descritas por una jerarquía de gráficos de secuencias de mensajes.

Para expresar la relación temporal entre funciones, se asocia un operador de composición a cada función global y abstracta, que indica cómo se ordenan entre sí temporalmente las subfunciones. Hay seis operadores:

- **es (is)** – La función resultante es una subfunción.
- **y (and)** – La función resultante es la secuencia de subfunciones temporalmente ordenadas de izquierda a derecha.
- **paralelo (parallel)** – No existe ordenación temporal entre las subfunciones.
- **o (or)** – La función resultante es una de las subfunciones.
- **repetición (repeat)** – La función resultante es la repetición de su subfunción.
- **excepción (exception)** – La función resultante es su subfunción cuando se encuentra un error.

En la Figura I.6-5 aparece un ejemplo de descomposición funcional. La función *llamada normal* corresponde a la secuencia *marcación, conversación y desconexión*. Téngase en cuenta que los símbolos utilizados para los operadores no son parte del SDL.

El formalismo se utiliza para expresar lo que debe ejecutar el sistema. El SDL describirá cómo debe ejecutarlo. Por consiguiente, las funciones definidas son realizadas por la descripción SDL.

## I.6.3.2 Especificación SDL

Por medio del SDL se especifica el sistema completo. Un aspecto importante del SDL es su capacidad para describir una descomposición jerárquica, que representa la arquitectura del sistema. Cada nodo SDL de esta jerarquía (sistema, bloque, proceso) se denomina una *entidad*.

## I.6.3.3 Gráficos de secuencias de mensajes

A cada función terminal se asocia un conjunto de gráficos de secuencias de mensajes (MSC). Este conjunto se estructura como una jerarquía de MSC. La jerarquía MSC debe corresponderse con la jerarquía de SDL. Cada MSC describe cómo se suministra un servicio terminal en un nivel de descomposición determinado de la arquitectura del sistema SDL. Se puede considerar que un MSC es la proyección de un servicio terminal en un nivel de descomposición de la jerarquía SDL.

## I.6.3.4 Reglas de coherencia entre descripciones

La metodología presentada exige que las tres clases de descripción sean coherentes durante todas las etapas de diseño del sistema.

En la Figura I.6-6 se muestra, por medio de un ejemplo sencillo, una visión global de las relaciones entre las tres clases de descripción, y se ilustran las reglas de coherencia. Deben cumplirse las reglas siguientes:

- La descomposición MSC debe ser coherente en la descomposición SDL.
- Todas las instancias y mensajes de un MSC deben ser visibles en la entidad correspondiente de la descripción SDL.
- Un MSC deberá ser coherente con la descripción SDL en el nivel de descripción correspondiente.

Una instancia de MSC muestra un conjunto de eventos de entrada y salida con ordenación totalmente lineal. Por consiguiente, una instancia de MSC puede interpretarse como una secuencia de eventos. A esa secuencia se la denomina un trazado MSC.

Una especificación de proceso SDL se puede representar como un grafo de alcanzabilidad (reachability graph): los nodos son estados de proceso, y los bordes etiquetados representan eventos de entrada. Un trazado SDL de un proceso es una secuencia de eventos en el grafo de alcanzabilidad.



# Reemplazada por una versión más reciente

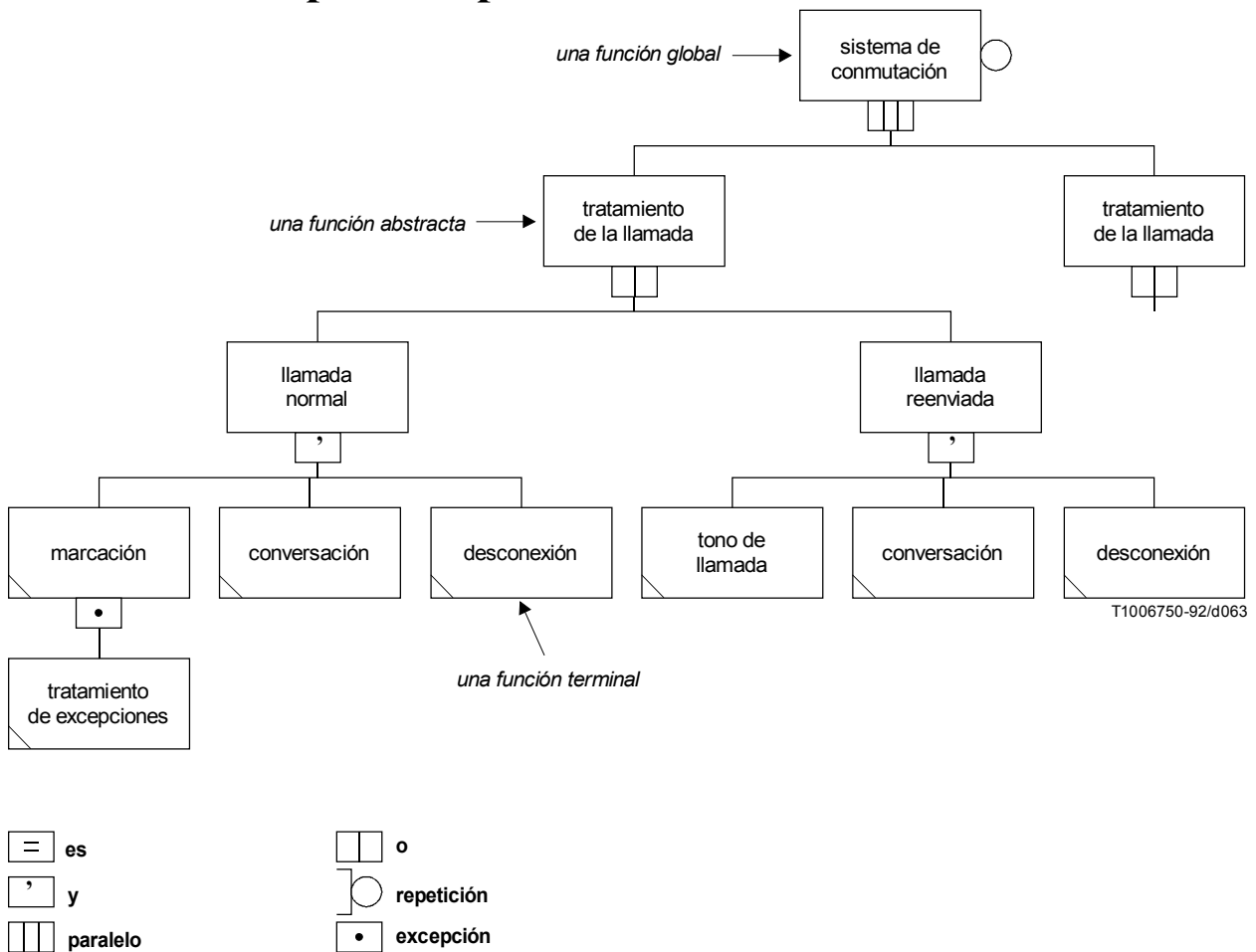


FIGURA I.6-5/Z.100

## Ejemplo de descomposición funcional

Ahora bien, la definición de coherencia se reduce a un problema teórico de grafo. Una instancia MSC es coherente con una especificación de proceso en SDL si existe un trazado SDL que contiene el trazado MSC y no muestra ningún otro evento de entrada y de salida (véase la Figura I.6-7).

Esta regla de coherencia no es suficiente para todas las clases de MSC. Se podría suprimir información sobre el flujo de señal entre procesos, por ejemplo, en el caso de adelantamiento de señales. Se puede generalizar la definición de los trazados y su correspondiente regla de coherencia para abarcar todas las clases de MSC. En la práctica, no obstante, se puede efectuar una verificación automática de la coherencia, únicamente para instancias MSC restringidas, debido a la explosión de estados.

## I.7 Obtención de realizaciones a partir de especificaciones SDL

El problema que se trata en esta cláusula es cómo establecer la correspondencia de sistemas abstractos especificados en SDL con sistemas reales, compuestos de soporte físico y soporte lógico. En general, los componentes del mundo real difieren de los componentes abstractos tanto en estructura como en comportamiento. Por ello, es preciso efectuar adaptaciones, tanto en el nivel abstracto como en el concreto, para garantizar que la especificación SDL define fielmente la funcionalidad del sistema real. Además del SDL propiamente dicho, hace falta documentación para definir el sistema concreto y sus relaciones con el sistema SDL abstracto. En esta cláusula se bosquejan los pasos principales para llegar a las realizaciones, y la forma en que esas realizaciones se pueden documentar en el nivel arquitectural. Las directrices paso a paso se resumen al final de la cláusula.

# Reemplazada por una versión más reciente

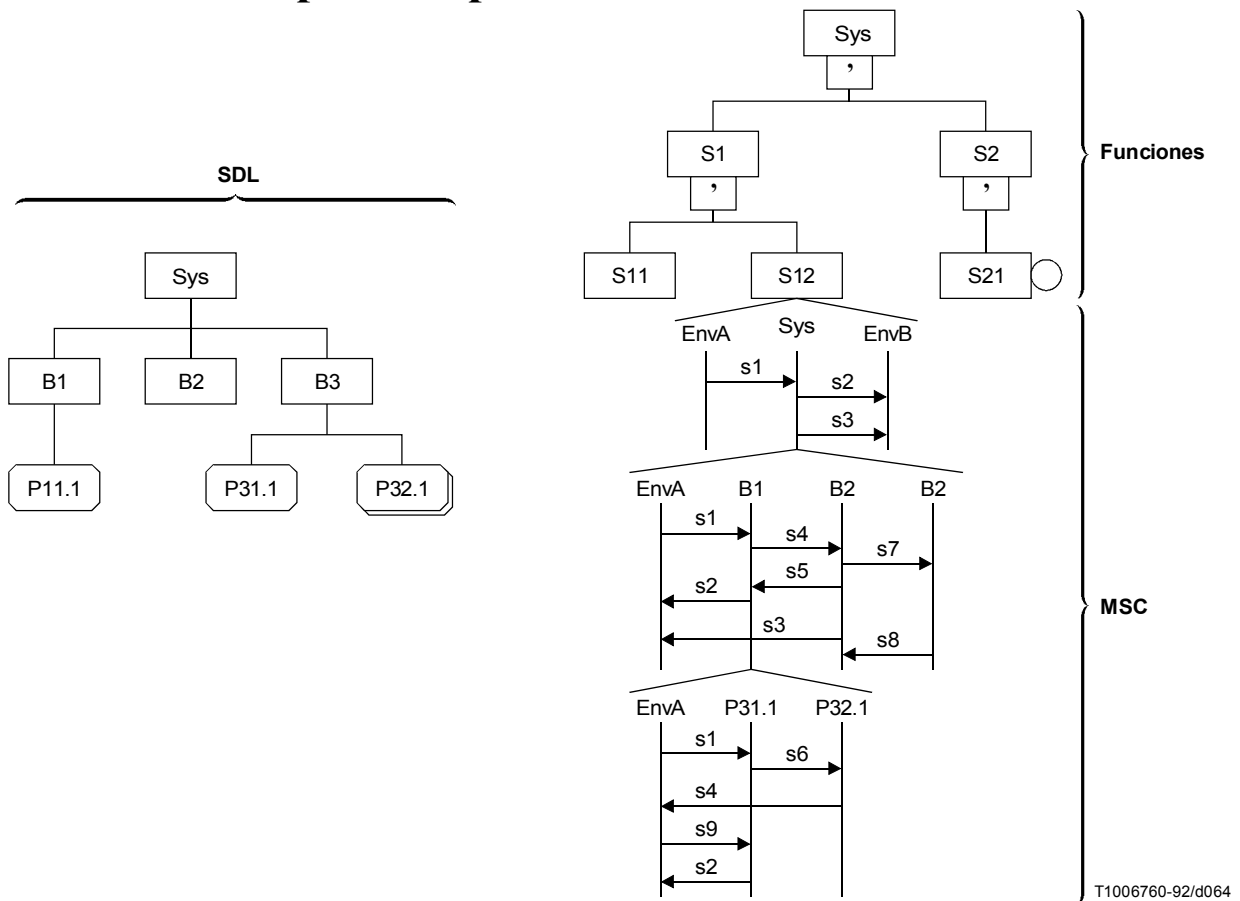


FIGURA I.6-6/Z.100  
Las tres perspectivas de un sistema

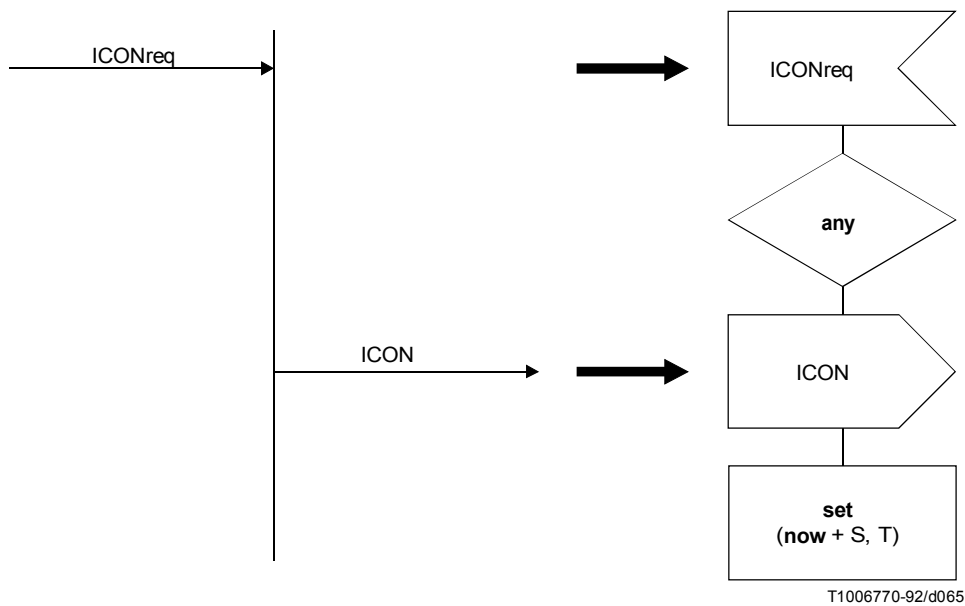


FIGURA I.6-7/Z.100  
Coherencia entre el trazado SDL y el trazado MSC

# Reemplazada por una versión más reciente

## I.7.1 Introducción

Durante el ciclo de vida de un sistema, las especificaciones SDL se utilizarán, al menos, para tres fines:

- En una etapa preliminar para especificar y validar la funcionalidad (comportamiento) requerida por el entorno de usuario (especificación de alto nivel).
- Luego, para proporcionar una base firme al diseño de la realización, es decir, para diseñar la realización óptima (especificación de bajo nivel).
- Después del diseño de la realización, para describir (documentar) las propiedades funcionales completas del sistema realizado (especificación orientada a la realización).

El SDL se basa en conceptos bien adaptados al primer objetivo mencionado: especificar el comportamiento observable de los sistemas con claridad y sin ambigüedades. Para conseguirlo, es preciso destacar el comportamiento externo, y descartar los detalles internos del diseño que no son pertinentes.

El SDL también es adecuado para el segundo objetivo: servir de base para la realización del diseño. El SDL tiene la propiedad conveniente de combinar la independencia con respecto a la realización y la posibilidad de llevarla a la práctica (salvo en el caso de tipos de datos abstractos infinitos). Para ello no deben insertarse decisiones de diseño prematuras en las especificaciones SDL, pero se podrá ofrecer, a manera de orientación adicional, los llamados requisitos no funcionales, es decir, las propiedades que tendrá la realización, además de las expresadas en la especificación.

Si el sistema real es funcionalmente equivalente al sistema SDL, entonces se logra también el tercer objetivo, lo que vale la pena, no sólo porque ahorra esfuerzos de documentación, sino también, porque las especificaciones SDL serán útiles en la prueba, funcionamiento y mantenimiento del sistema. Si los diseñadores y programadores consideran que las especificaciones SDL resultan útiles para su labor, estarán suficientemente motivados para mantenerlas actualizadas, y evitarán cambios en el nivel de la realización que degeneren la documentación. Para ello, será preciso adaptar las especificaciones SDL de manera que reflejen las propiedades funcionales específicas de la realización subyacente.

El diseño de la realización de un sistema SDL tiene dos aspectos importantes:

- Primero: la selección entre realizaciones alternativas que son funcionalmente equivalentes al sistema SDL (aspecto de «proyección»). Normalmente, el diseñador dispondrá de varias posibilidades, entre las que debe seleccionar una realización óptima con respecto a los requisitos no funcionales.
- Segundo: la adaptación de la especificación SDL cuando la realización seleccionada no es funcionalmente equivalente (aspecto de «regeneración»). Entre el mundo abstracto del SDL y el mundo real hay diferencias considerables, que se manifestarán con frecuencia en el comportamiento observable del sistema. En esos casos, habrá que adaptar toda la especificación SDL del sistema para mantener la equivalencia funcional.

En la Figura I.7-1 se ilustra el primer aspecto. Se utiliza la especificación SDL junto con los requisitos no funcionales para seleccionar y especificar una realización. La especificación de realización resultante define la correspondencia entre la especificación SDL y la ejecución del sistema real. Este es ortogonal a la especificación SDL y se muestra en una casilla diferente en esa figura. La separación no es el punto clave en este caso, sino la índole ortogonal de la información. En la práctica, la especificación de la realización podría incluirse como anotaciones en la especificación SDL. Cabe hacer algunas observaciones con respecto a la Figura I.7-1.

- La misma especificación SDL cumple en este caso los tres objetivos de las especificaciones SDL antes mencionados.
- De la misma especificación se pueden obtener realizaciones alternativas, mediante otras posibles especificaciones de realización.
- Si la actividad de realización está automatizada, la realización se mantendrá coherente con la especificación en todo momento (incluida la corrección de la traducción).

En muchas situaciones reales, la visión algo idealizada que muestra la Figura I.7-1 será modificada por el segundo aspecto. Por ejemplo, si el sistema ha de ser aplicado por una red informatizada distribuida, algunos canales SDL se realizarán mediante protocolos de red. Estos protocolos son necesarios en una realización distribuida, pero no en una solución centralizada. Dado que partes de un sistema distribuido pueden no ser operacionales mientras que otras siguen en estado operativo, el tratamiento de los errores es diferente en un sistema distribuido que en un sistema centralizado. Por tanto, la especificación SDL para un sistema distribuido puede ser diferente de la de un sistema centralizado a efectos de la definición de la funcionalidad completa que se realiza efectivamente.

## Reemplazada por una versión más reciente

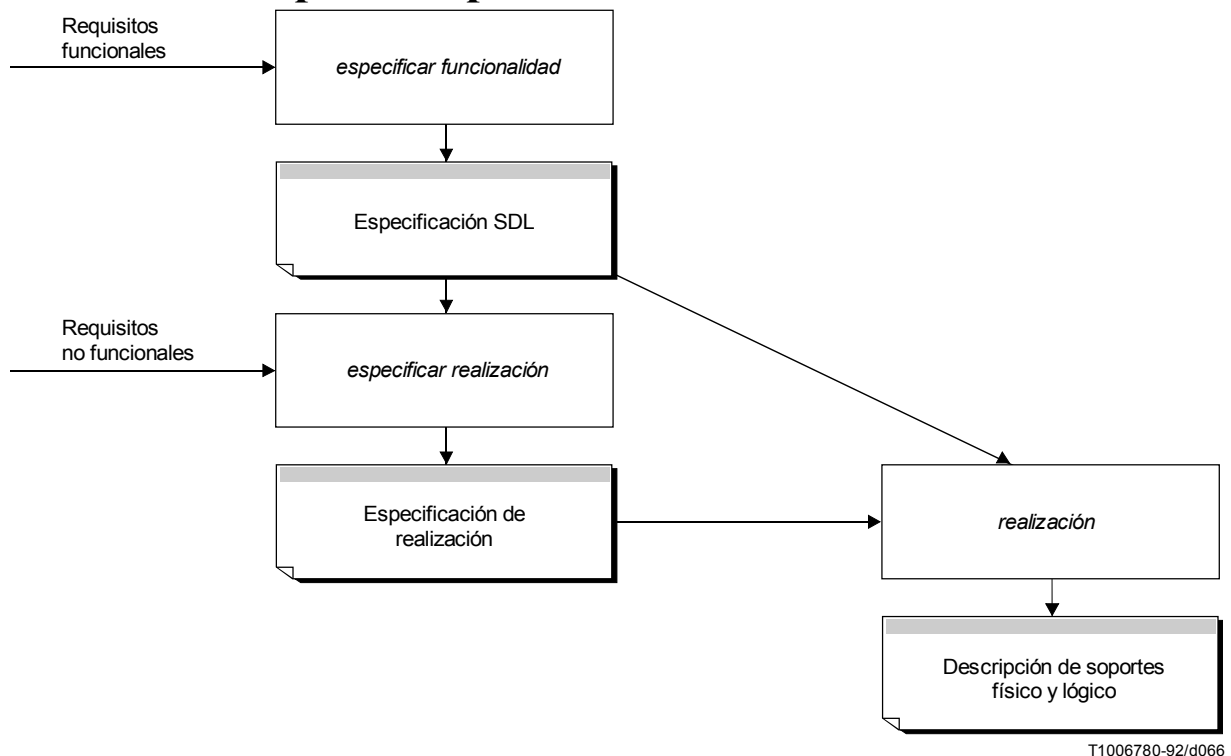


FIGURA I.7-1/Z.100

### Requisitos no funcionales utilizados para seleccionar una realización funcionalmente equivalente (primer aspecto)

La Figura I.7-2 ilustra la interacción entre los dos aspectos (de «proyección» y de «regeneración»). Al igual que en la Figura I.7-1, las propiedades no funcionales se utilizan para elaborar una especificación de realización. El conocimiento de la realización se utiliza para refinar y reestructurar la especificación SDL hasta obtener una especificación SDL de bajo nivel refinado completo. Sin embargo, incluso esta última especificación SDL permitirá otras realizaciones funcionalmente equivalentes. Por consiguiente, se necesita aún la especificación de realización para dirigir los pasos de la realización como en la Figura I.7-1.

Se hacen las siguientes observaciones con respecto a la Figura I.7-2:

- Se necesitan dos especificaciones SDL, una de alto nivel y otra de bajo nivel, para cumplir los tres objetivos del SDL.
- De una especificación SDL de alto nivel dada se pueden obtener varias especificaciones SDL de nivel bajo, según la especificación de la realización.
- Los dos aspectos se pueden separar y combinar de forma estructurada.

Cuando se construye un sistema nuevo a partir de cero, es deseable que la especificación SDL sea suficientemente independiente de la realización para cumplir el segundo objetivo (mencionado al comienzo de este punto), lo que puede significar una regeneración considerable al elegir la realización<sup>4)</sup>. Sin embargo, cuando se amplía o mejora un sistema existente, se tienen más conocimientos sobre la realización, por lo que se puede elaborar más directamente la especificación SDL de bajo nivel y evitar dicha regeneración.

<sup>4)</sup> Pero esto no significa que cambie todo en la especificación. Con frecuencia, se puede confinar el refinamiento y la reestructuración necesarias a partes limitadas del sistema.

## Reemplazada por una versión más reciente

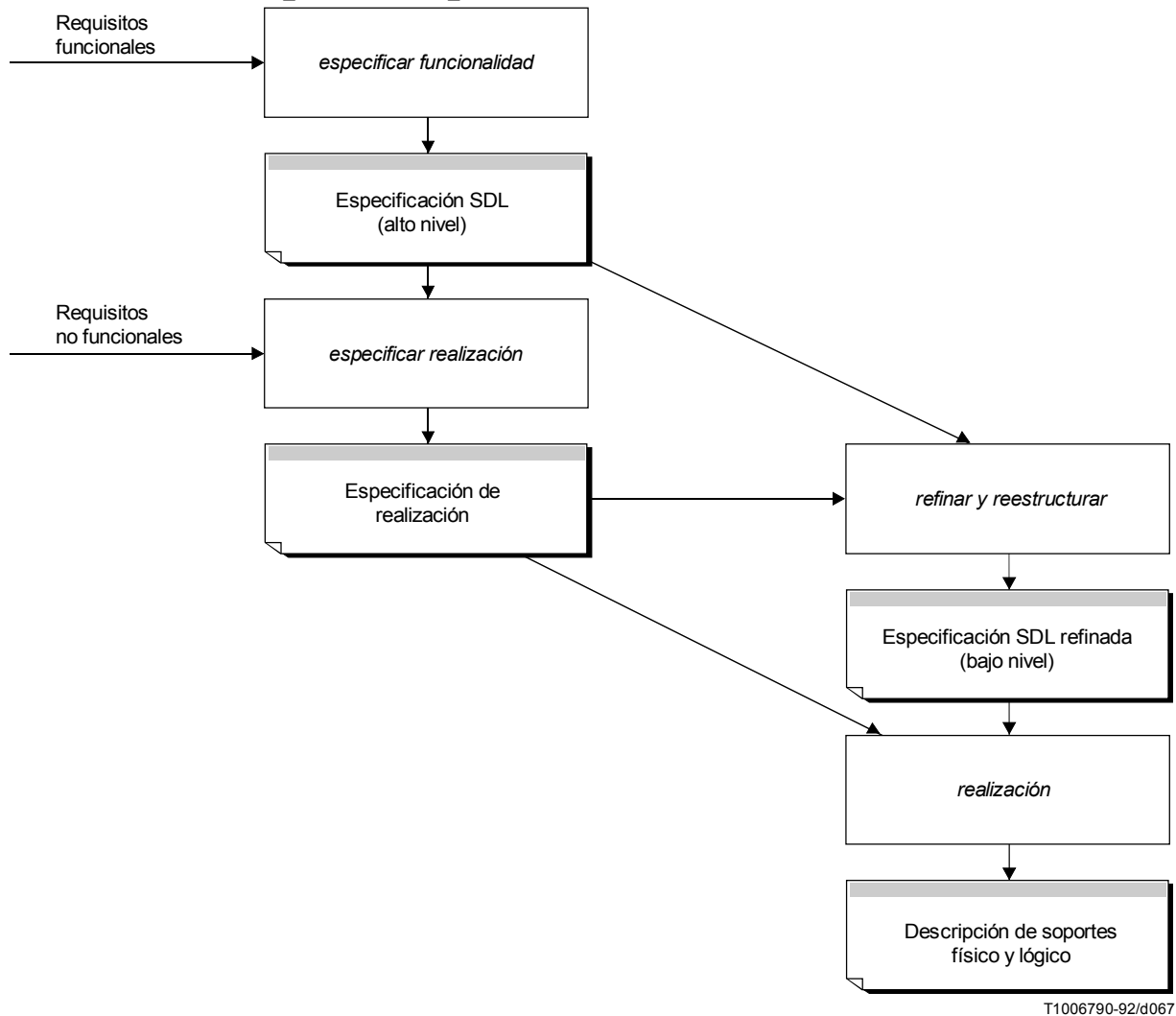


FIGURA I.7-2/Z.100

### Aspectos de proyección y de regeneración del diseño de realización

Cabe ahora plantear tres preguntas:

- ¿Cuáles son las diferencias fundamentales entre los sistemas SDL y los sistemas reales que deben considerarse?
- ¿Cómo se debe expresar la especificación de realización: como anotaciones a la especificación SDL de nivel bajo o separadamente de la especificación SDL?
- ¿Cuáles son las principales alternativas de realización entre las que se puede elegir?

A continuación se analizarán esas cuestiones.

#### I.7.2 Diferencias entre sistemas reales y sistemas SDL

Un buen diseñador debe estar consciente de las diferencias que existen entre el mundo real y el abstracto. Hay dos categorías principales de diferencia:

- Diferencias fundamentales en la naturaleza de los componentes. Los componentes físicos son más bien imperfectos en comparación con las propiedades más ideales de los componentes del SDL, pues generan errores al correr del tiempo, están sometidos a ruido y necesitan tiempo para ejecutar sus tareas de procesamiento.
- Diferencias conceptuales en el funcionamiento de los componentes. En ambos mundos existen conceptos relativos a la concurrencia, comunicación, comportamiento secuencial y datos, pero no son necesariamente los mismos.

# Reemplazada por una versión más reciente

## I.7.2.1 Diferencias fundamentales

### Tiempo de procesamiento

Un sistema SDL no está limitado por los recursos de procesamiento. Por consiguiente, no es necesario tener en cuenta el equilibrio entre la carga de tráfico ofrecida al sistema y su capacidad de procesamiento. Simplemente, se supone que el sistema es suficientemente rápido para procesar la carga que se le ofrece.

A este respecto, el mundo real es muy diferente. Cada transferencia de señal, y cada transición de un proceso, necesitarán un cierto tiempo y ciertos recursos de procesamiento. Por ello, una de las cuestiones más importantes para el diseño de la realización es equilibrar la capacidad de procesamiento de la realización y la carga de tráfico ofrecida.

Una cuestión conexas es la rapidez con que el sistema deberá responder a ciertas entradas. Una vez más, al sistema SDL no se le plantean problemas, pero se ejercerá considerable presión sobre la realización para que satisfaga los requisitos de tiempo de respuesta.

Debido a estas diferencias, el punto focal del diseño de la realización es bastante diferente del de la especificación funcional, ya que se trata de conseguir realizaciones de los conceptos SDL que sean suficientemente rápidas para satisfacer la carga de tráfico y los requisitos de tiempo de respuesta sin menguar la validez de las especificaciones SDL.

Puesto que las especificaciones SDL especifican claramente las interacciones internas y externas que se precisan para ejecutar determinadas funciones, proporcionan una base excelente para evaluar la capacidad de procesamiento necesaria para cumplir los requisitos relativos a la carga y al tiempo de respuesta. Esto se examina en I.7.4.3.

### Errores y ruido

Los sistemas SDL están expuestos a errores de especificación, pero el mundo abstracto del SDL no es afectado por errores físicos. Se supone, simplemente, que los procesos y canales funcionan siempre conforme a sus especificaciones. No se supone que los procesos se detendrán de tiempo en tiempo, o que los canales distorsionarán el contenido de las señales. Sin embargo, esas situaciones se producen en el mundo real. A veces, los errores se manifiestan como fallos en el funcionamiento de los canales y procesos.

Los errores de soporte físico, el ruido y los daños físicos son inevitables, ya que obedecen a procesos físicos que están totalmente fuera del dominio lógico. Además es normal que aparezcan en la realización errores lógicos que no estaban en la especificación SDL.

Los mecanismos causantes de errores y ruido no están dentro del alcance de la especificación SDL, pero con frecuencia, habrá que tratar explícitamente en la especificación los efectos de errores y ruido. Es preciso considerar lo que deberá hacer un proceso si no recibe respuesta de otro proceso, o si recibe una respuesta errónea. ¿Cabe la posibilidad de que un canal deje de funcionar, o que el ruido distorsione las señales? ¿Cuál será la reacción ante un canal que deja de funcionar? ¿Qué pasará si un proceso produce, de repente, señales «incontroladas»? Estos son los tipos de preguntas que plantean y que hay que responder.

Hasta cierto punto, las respuestas dependen de la distribución física de los procesos SDL en el sistema real y de las distancias físicas que los canales SDL deben cubrir. Los procesos y canales separados físicamente pueden fallar independientemente. Los canales que cubren grandes distancias físicas están sujetos a más errores y ruido que los canales en el soporte lógico de un computador.

Un efecto positivo de la separación física es que los errores están aislados. Los errores que se producen en una unidad no afectan a las restantes unidades del sistema, siempre y cuando éstas no reciban la información errónea o puedan protegerse contra ella. Por consiguiente, la separación física puede mejorar el tratamiento de los errores, pero en general, ello no se conseguirá sin tener que pagar un precio. Es necesario detectar los errores y aislarlos para que las partes operacionales sigan funcionando. El tratamiento adecuado de este aspecto puede ser bastante complejo y normalmente requiere funcionalidad adicional en la especificación SDL.

Una especificación SDL no dice nada sobre la distancia física de un canal, aunque en realidad, puede tratarse (o no) de distancias grandes. Esto significa que se necesitan equipos y protocolos de transmisión para establecer un canal fiable. Por consiguiente, la distancia física puede exigir nuevas funciones para sustentar el establecimiento de canales.

### Recursos finitos

Todos los recursos de un sistema real son finitos. Es posible que exista un número máximo de procesos que el sistema operativo pueda tratar, o un número máximo de memorias tampón para el envío de mensajes. La longitud de la palabra está limitada, como también lo está el espacio de memoria. Hasta los datos primitivos, como los enteros, son finitos.

## Reemplazada por una versión más reciente

Por otra parte, el SDL tiene una cola ilimitada en el puerto de entrada de cada proceso, y permite especificar datos infinitos. Por ello, el diseñador deberá tratar de encontrar las maneras de llevar a la práctica sistemas SDL potencialmente infinitos mediante recursos finitos. Una manera consiste en restringir la utilización del SDL, de forma que se asegure de que todos los valores están limitados. Otra es tratar las limitaciones de recursos en la realización, preferentemente de modo que resulte transparente para el nivel SDL. Cuando no se puede conseguir esa transparencia, se debe aceptar apartarse de la semántica SDL o tratar las limitaciones explícitamente en la especificación SDL.

### I.7.2.2 Diferencias conceptuales

#### Concurrencia

En el modelo de concurrencia utilizado en el SDL se supone que los procesos se comportan de forma independiente y asíncrona. No hay una ordenación relativa de operaciones de diferentes procesos, excepto la ordenación implícita de envío y recepción de señales.

Esto hace que los procesos SDL se puedan realizar verdaderamente en paralelo en diferentes unidades de soporte físico, o casi en paralelo en soportes físicos compartidos.

Los objetos físicos del mundo real se comportan realmente en paralelo, lo que significa que las operaciones en objetos diferentes se efectúan en paralelo entre sí a la velocidad del soporte físico que las ejecuta. Hay dos tipos de funcionamiento:

- *Síncrono*, cuando las operaciones de los objetos paralelos se ejecutan al mismo tiempo. Se utiliza principalmente en el nivel de circuito electrónico donde esas operaciones pueden ser controladas por señales de reloj comunes. Desde el punto de vista de la semántica SDL, podría decirse que éste es un caso especial.
- *Asíncrono*, cuando las operaciones de los objetos paralelos se ejecutan independientemente y quizás en tiempos diferentes. A menos que se conozcan con precisión las velocidades de procesamiento, no se podrá saber la ordenación exacta de las operaciones. Esto se adecúa bien a la semántica SDL.

El casi paralelismo significa que sólo un proceso estará activo a la vez y que ese proceso activo bloqueará el funcionamiento de los demás procesos mientras se le permita permanecer activo. Esto afectará los tiempos de respuesta de los procesos bloqueados. Se necesitará apoyo adicional para llevar a cabo la organización y multiplexación de los procesos casi paralelos en la parte superior de la máquina secuencial. Normalmente, esto es tratado por un sistema operativo, que puede considerarse como una capa que realiza una máquina virtual casi paralela por encima de la máquina física.

#### Comunicación

Básicamente, hay dos clases diferentes de necesidades de comunicación:

- comunicar una secuencia de símbolos o valores, en un orden determinado;
- comunicar un símbolo o valor, continuamente mientras es válido.

En el primer caso, el ordenamiento secuencial es importante. En el segundo, la secuencia no importa, lo único que interesa es el valor vigente en cada instante de tiempo.

Al leer un libro, la secuencia de letras, palabras y oraciones es fundamental para la comprensión. Cuando se llega a un semáforo, el color del disco determinará si hay que detenerse o no. Los cambios de color anteriores no cuentan. Los símbolos continuos se pueden leer una y otra vez, mientras que los que forman una secuencia se leen, generalmente una sola vez.

Las señales en el SDL pertenecen a la categoría de secuencia de símbolos. Se leen sólo una vez y se consumen en el extremo receptor. Son muy adecuadas para la representación de secuencias de eventos.

La construcción visión/revelado del SDL pertenece a la categoría de valor continuo. La construcción exportación/importación parece semejante, pero es una representación abreviada de un protocolo subyacente de señales que permite al receptor seguir la trayectoria de un valor continuo.

## Reemplazada por una versión más reciente

Las dos formas de comunicación son duales en el sentido de que se puede utilizar una forma para realizar la otra. Se analizará primero la necesidad de comunicar un valor continuo. La manera más directa es utilizar un medio de comunicación que transmita el valor continuamente, por ejemplo una variable compartida en el soporte lógico o una conexión eléctrica en el soporte físico. También se puede utilizar un medio secuencial, como una cola de mensajes, para transmitir una secuencia de símbolos que representan la secuencia de cambios (eventos) del valor continuo. Para que este esquema funcione, es preciso que el emisor produzca eventos y que el receptor integre eventos para regenerar el valor continuo. Este principio se aplica en el mecanismo de exportación/importación del SDL.

¿Qué se puede decir de la necesidad de comunicar una secuencia de símbolos? La manera más directa de hacerlo es utilizar un medio de comunicación secuencial, por ejemplo, una cola de mensajes, pero también se puede emplear un medio de valor continuo. En este caso, hay que representar la secuencia mediante los cambios (eventos) sufridos del valor continuo. Para que este esquema funcione, el emisor debe integrar una secuencia de eventos para conformar el valor continuo, y el receptor debe producir los eventos a partir de cambios del valor continuo.

Aunque se puede realizar una forma de comunicación mediante la otra, el cambio de paradigma conlleva una penalización.

El medio de realización «natural» de las señales SDL será el orientado a la secuencia de símbolos, pero ésta no será siempre la mejor forma en el sistema real. A veces, hay que realizar las señales SDL por medio de valores continuos.

La entrada procedente de un teclado puede servir de ejemplo. La señal de salida de una tecla es, básicamente, un «1» continuo cuando se pulsa, y un «0» continuo cuando no se pulsa, pero el sistema necesita conocer la secuencia de teclas pulsadas, y no los valores instantáneos. Por consiguiente, tiene que detectar los cambios de valores (eventos) y convertirlos en símbolos, que representan las pulsaciones de teclas completas. Con frecuencia, se necesita este tipo de detección de eventos en las interfaces de un sistema real, y se puede efectuar en el soporte lógico o en el físico.

Las señales visuales en una pantalla son otro ejemplo. El usuario necesita que la información sea presentada como valores continuos, y no como mensajes que oscilan en la pantalla. Por ello, hay que convertir la señal SDL en valor continuo en la pantalla.

Por otra parte, cuando se utiliza la exportación/importación en el SDL, puede ser mejor emplear valores continuos en la realización en vez del protocolo de secuencias de símbolos supuesto en el SDL<sup>5)</sup>.

Los canales que cruzan la frontera soportes físico-lógico requieren especial atención. Un canal atómico, representado por una línea en la representación gráfica, puede resultar una mezcla de líneas físicas, equipo electrónico y soporte lógico en el sistema real. Las primitivas de comunicación y sincronización utilizadas en el lado de soporte físico diferirán, con frecuencia, de las usadas en el lado de soporte lógico de la frontera, véanse los puntos siguientes. Será preciso efectuar conversiones de una forma u otra, lo que a menudo es una labor crítica desde el punto de vista del tiempo que necesita una optimización cuidadosa.

### Sincronización

Considérense dos procesos SDL que comunican. El proceso emisor puede enviar una señal en cualquier momento porque será almacenada en la memoria tampón del puerto de entrada del proceso receptor, que puede consumirla en un momento posterior.

Este es un esquema de comunicación almacenada en memoria tampón, y el emisor puede enviar una cantidad infinita de señales sin esperar a que el receptor las consuma. Con frecuencia, esto se denomina *comunicación asíncrona*.

La comunicación asíncrona se puede comparar con la denominada *comunicación síncrona*, en la cual la operación de envío y la operación de consumo se realizan al mismo tiempo. Esto es necesario cuando no hay un tampón entre ambos procesos.

Al acto de alinear entre sí las operaciones de diferentes procesos concurrentes se llama generalmente *sincronización*. La sincronización es necesaria no sólo para lograr comunicaciones correctas, sino también para controlar el acceso a los recursos compartidos del sistema físico.

---

<sup>5)</sup> Esto modifica ligeramente la semántica del SDL, por lo que debe tenerse cuidado, aunque normalmente sea aceptable por el usuario y mucho más eficaz.



# Reemplazada por una versión más reciente

La sincronización de las interacciones se puede clasificar en las categorías siguientes:

- a) *Interacción síncrona*, cuando los procesos ejecutan operaciones de interacción al mismo tiempo. Tiene dos subcategorías:
  - 1) *Dependiente del tiempo* – En este caso, el medio de transmisión propiamente dicho es síncrono, por ejemplo, los canales de un sistema PCM. El emisor y el receptor tienen que mantener la temporización común del medio de transmisión. Esto impone limitaciones de tiempo real tanto al emisor como al receptor.
  - 2) *Independiente del tiempo*, o sincronizada explícitamente – En este caso, se realiza una sincronización explícita, independiente del tiempo, entre los procesos, por la que quedan enganchados durante la interacción. Los procesos pueden tener que esperar que la interacción esté habilitada. Una vez habilitada, la interacción se efectúa mediante operaciones, que son ejecutadas simultáneamente por ambos procesos. La comunicación en LOTOS pertenece a esta categoría. Este es un esquema secuencial.
- b) *Interacción asíncrona*, cuando los procesos ejecutan operaciones de interacción no necesariamente al mismo tiempo. Aun en este caso hay que alinear las operaciones según un orden relativo para garantizar la interacción correcta. En la comunicación de valores continuos, el requisito principal es que las operaciones sean *mutuamente exclusivas*. La comunicación de secuencias de valores tiene dos subcategorías:
  - 1) *Dependiente del tiempo* – El medio de transmisión propiamente dicho es asíncrono y no tiene un mecanismo explícito de sincronización. Este esquema depende de la velocidad del receptor con respecto a la del emisor. El emisor debe producir salidas a una velocidad que el receptor pueda seguir. Un ejemplo típico es la marcación por impulsos decádicos de las antiguas líneas de abonado. Este esquema se utiliza ampliamente en los enlaces físicos de comunicación (y es una causa evidente de problemas en tiempo real).
  - 2) *Independiente del tiempo*, o explícitamente sincronizada – En este caso, hay un tampón y una sincronización explícitamente independiente del tiempo entre los procesos que interactúan. En general, el tampón es una cola en la que el primero en llegar es el primero en salir. El emisor pone (una secuencia de) valores (señales) en el tampón y el receptor saca (la secuencia de) valores en algún momento posterior. La capacidad del tampón determina cuántos valores puede producir el emisor con respecto al receptor. La comunicación de señales SDL pertenece a esta categoría. Se utiliza ampliamente en los sistemas de soporte lógico y es menos común en el soporte físico. La interacción síncrona puede considerarse un caso especial cuando la capacidad del tampón es nula.

En la semántica SDL se supone la interacción asíncrona con capacidad de memoria tampón infinita, pero en la práctica, la capacidad tampón debe limitarse de alguna manera.

A veces, la índole de la aplicación determina que el número de señales que el productor puede generar con respecto al consumidor esté siempre limitado. En otros casos, la limitación depende de que el consumidor sea suficientemente rápido para evitar que las colas se alarguen demasiado. Por lo común, los problemas que plantea el desbordamiento de la memoria tampón se solucionan imponiendo retardos al productor cuando el tampón alcanza su capacidad máxima.

Esto significa que quizás sea necesario demorar las salidas de un proceso SDL hasta que el tampón de recepción esté listo. En una realización finita, es casi inevitable apartarse de la semántica SDL. Se necesita un diseño cuidadoso para minimizar los problemas prácticos que esto pueda causar.

Puede suceder con frecuencia que en las interfaces físicas del sistema hay mecanismos que difieren de los mecanismos SDL. Por ejemplo, la interacción dependiente del tiempo es bastante típica en los canales físicos, lo que entraña que se necesitará la supervisión y la generación críticas de eventos en el tiempo.

El diseñador se verá confrontado, por una parte, con las primitivas de sincronización disponibles en el sistema real y, por otra, con la sincronización implícita en la especificación SDL. Con frecuencia se necesitará funcionalidad adicional para reunir las diversas formas.

## Datos

Los datos SDL se basan en la noción de tipos de datos abstractos, cuyas operaciones se pueden definir mediante ecuaciones. Por lo general, una realización exigirá tipos de datos concretos, cuyas operaciones se definirán operacionalmente. En consecuencia, es probable que el diseñador tenga que transformar los tipos de datos abstractos del SDL en tipos de datos más concretos, adecuados a la realización.

# Reemplazada por una versión más reciente

## I.7.3 Especificaciones de realización

De la independencia con respecto a la realización se deduce que es posible reutilizar la misma especificación SDL en varias realizaciones diferentes. Asimismo, se deduce que la estructura física de esas realizaciones puede variar considerablemente. En un caso, cada proceso SDL se puede realizar en una microplaqueta física separada, mientras que en otro, los procesos pueden ser parte del soporte lógico instalado en el mismo computador. La manera en que esto se hace necesita documentación.

### I.7.3.1 Fundamentos

Las especificaciones de realización definen la correspondencia entre un sistema SDL abstracto y un sistema concreto constituido por componentes de soporte físico y soporte lógico. Es obvio que un sistema SDL se puede refinar y reestructurar hasta que refleje gran parte del diseño, pero aun así, el sistema SDL seguirá siendo abstracto y podrá realizarse de maneras diferentes. Por tanto, para definir la realización se necesita algo más que el SDL puro.

¿Cómo se expresará la especificación de realización?

Una posibilidad consiste en añadir especificaciones de realización como anotaciones a las especificaciones SDL. La ventaja de esta solución estriba en que toda la información que describe el sistema se encuentra en un solo lugar. La desventaja es que las especificaciones SDL quedan vinculadas a determinados diseños de realización. Si se describe la realización por separado, será más fácil reutilizar las especificaciones SDL en sistemas que tienen realizaciones diferentes. También será más fácil centrar las actividades de diseño en los aspectos particulares de las realizaciones.

Hay que considerar los aspectos siguientes:

- La estructura general del sistema real, en términos de los componentes de soportes físico y lógico.
- Las propiedades no funcionales de los componentes del sistema real y el sistema resultante. En particular, su funcionamiento en términos de capacidad de tratamiento del tráfico, tiempos de respuesta y tratamiento de errores.
- La correspondencia entre el sistema SDL abstracto y los componentes del sistema real.

Puede ser provechoso especificar y analizar estos aspectos apartándose un poco de la estructura funcional de los sistemas SDL. Los principales criterios de estructuración de los sistemas reales están vinculados con la calidad de funcionamiento, el costo y las propiedades físicas, mientras que los criterios relativos a las especificaciones SDL son la claridad y la integridad del comportamiento. La naturaleza de esos criterios es tan diferente que no siempre resultarán en sistemas semejantes. Además, ciertos aspectos de los sistemas reales son suficientemente complejos por sí mismos para justificar una especificación separada.

Para analizar los aspectos de diseño de las realizaciones independientemente de las especificaciones SDL sin entrar en detalles sobre realizaciones específicas, a continuación se presenta, con carácter informal, una notación para especificaciones de realización.

### I.7.3.2 Notación para especificaciones de realización

La notación para las especificaciones de realización se centra en las estructuras del soporte físico y del soporte lógico, y en las correspondencias establecidas entre especificaciones en distintos niveles de abstracción. Obsérvese que esta notación no es normativa.

Se trata de una notación bastante general para diagramas de bloques. En muchos aspectos es sintácticamente similar a los diagramas de interacción de bloques SDL, pero existen diferencias importantes de significado.

Las casillas y las flechas de los diagramas SDL de interacción de bloques representan bloques y canales abstractos, que tienen una semántica bien definida. Se pueden comprender y analizar por sí mismos, independientemente de la realización.

El objetivo de las especificaciones de realización es definir la correspondencia entre la realización y las especificaciones SDL. A tal efecto, no precisan una semántica propia, en el sentido de que el SDL tiene una semántica. Su significado está dado por lo que representan en el mundo real. Para el mundo real propiamente dicho existen otros formalismos, por ejemplo, los lenguajes de programación y los lenguajes de descripción de soporte físico con semánticas bien definidas. Así, el alcance de las especificaciones de realización se puede limitar a una correspondencia sintáctica.

Al representar la estructura de la realización mediante una notación gráfica, se gana en visión general y comprensión de la estructura interna del sistema físico. En la Figura I.7-3 se ofrece un ejemplo de estructura de soporte físico.

# Reemplazada por una versión más reciente

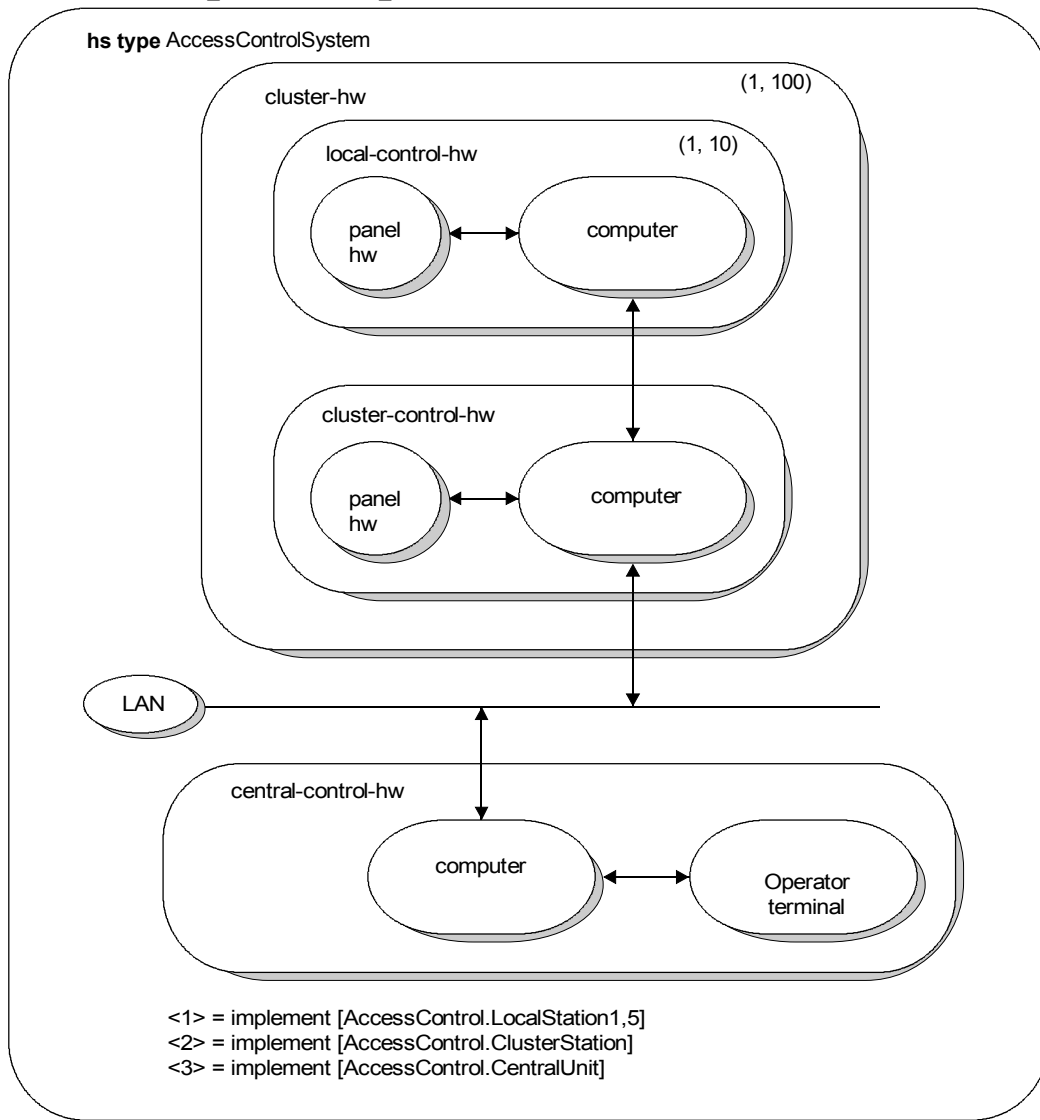


FIGURA I.7-3/Z.100

## Ejemplo de estructura de soporte físico

Las casillas representan unidades concretas de soporte físico, tales como computadoras, circuitos impresos, etc. Las flechas representan conexiones físicas, tales como cables. Con esta notación, se puede descomponer la especificación de soporte físico para llegar paso a paso hasta los detalles. La idea es utilizar esta notación para definir la estructura general, y utilizar después notaciones especiales de soporte físico, tales como diagramas de circuitos, para los detalles. De esta manera se puede elaborar una especificación de soporte físico bien estructurada. Véase al final del punto la tabla de símbolos para los diagramas de soporte físico.

Las puntas de flecha indican el sentido de las señales que fluyen por conexión. Puede haber conexiones bidireccionales.

La notación para la estructuras de soporte lógico se basan en los mismos principios que la notación para la estructura de soporte físico, solo que las casillas tienen formas diferentes, que permiten diferenciar los tipos característicamente diferentes de unidades de soporte lógico (véase la Figura I.7-4).

Las casillas con borde triple representan el *proceso de soporte lógico*, una unidad de soporte lógico que contiene al menos un programa de no terminación. Este programa se ejecutará como procesos casi paralelos en un sistema operativo. Así, estas casillas de borde triple representan unidades concurrentes. Las unidades que solo contienen programas de terminación, tales como los procedimientos, se representan mediante casillas de borde doble, y los datos puros, mediante casillas de un solo borde.

## Reemplazada por una versión más reciente

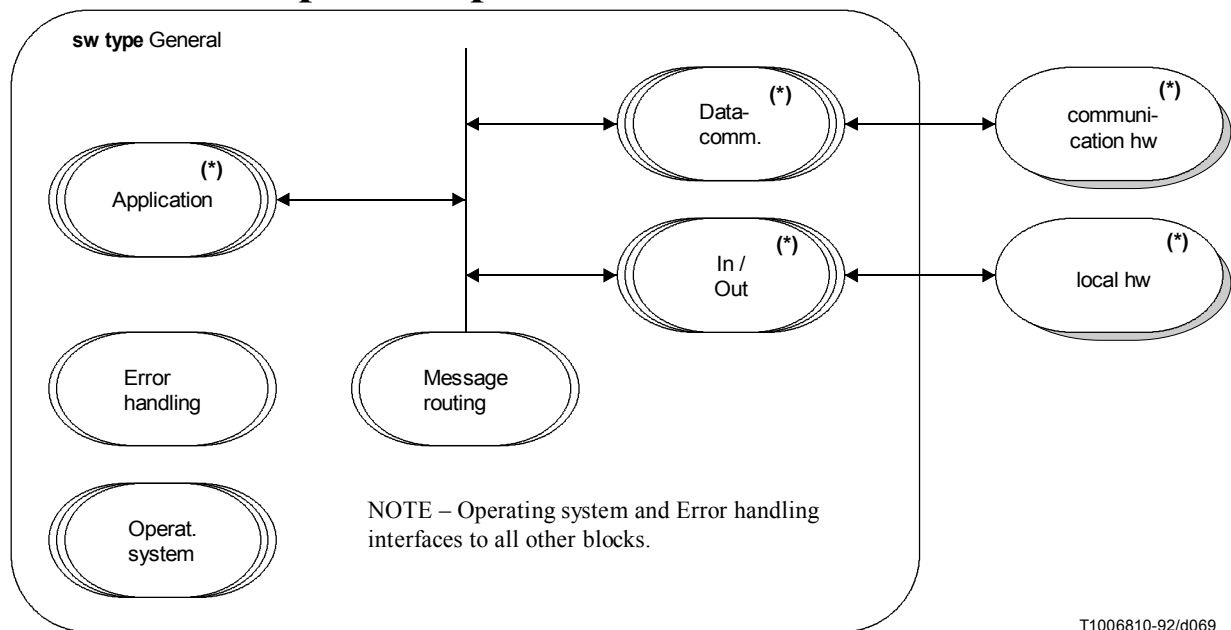


FIGURA I.7-4/Z.100

### Ejemplo de estructura de soporte lógico

La estructura de soporte lógico de la Figura I.7-4 es bastante general. Contiene varios procesos de soporte lógico, algunos de los cuales ejecutan funciones de aplicación, mientras que otros producen entradas y salidas y algunos se destinan a la comunicación entre computadores. Esos procesos están organizados por un sistema operativo y comunican intercambiando mensajes mediante un procedimiento de encaminamiento de mensajes.

La notación de estructura de soporte lógico se utiliza para proporcionar una visión general y un enfoque gradual del análisis de los detalles. Esta notación combina estructuras de datos y estructuras de programas en una notación unificada. Permite la representación de las relaciones estructuradas de red y puede expresar la concurrencia.

Para definir el sistema de soporte lógico con todo detalle se pueden utilizar otras notaciones, tales como las de pseudocódigo o código fuente.

Puesto que resulta difícil ver las relaciones en un texto de programa lineal, es provechoso representarlas gráficamente. Como la activación o flujo de control se ve claramente en el texto de programa, se considera como la relación menos importante en la representación de la estructura del soporte lógico. Se destacarán los flujos de datos y las referencias.

Las flechas que representan flujos de datos, referencias y activaciones entre unidades de soporte lógico tienen formas diferentes, véase la tabla de símbolos al final de la cláusula.

Los diagramas de estructura de soporte lógico y los de estructura de soporte físico se pueden descomponer jerárquicamente. Tienen conceptos semejantes para tipos e instancias. La forma de presentación de ambos diagramas es similar a la representación gráfica del SDL: un símbolo de recuadro representa la frontera entre la entidad especificada y su entorno.

Dentro del símbolo de recuadro, en el ángulo superior izquierdo, se colocará el tipo de diagrama y el nombre de entidad especificada:

<b>hs type</b> <name>	especificación de un tipo de estructura de soporte físico
<b>ss type</b> <name>	especificación de un tipo de estructura de soporte lógico
<b>hs</b> <name>	especificación de una instancia de estructura de soporte físico
<b>ss</b> <name>	especificación de una instancia de estructura de soporte lógico.

# Reemplazada por una versión más reciente

Se pueden agrupar las casillas y las flechas en conjuntos o matrices. El tamaño de un conjunto o matriz se indica mediante un número mínimo y un número máximo de elementos encerrados entre paréntesis:

- (<min>,<max>) al menos <min>, como máximo <max>
- (<min>,) al menos <min>, ningún límite superior
- (+) al menos uno, ningún límite superior
- (\*) cualquier número

La casilla se identifica mediante un nombre de tipo y, facultativamente, un nombre de instancia:

- lu:LocalUnit instancia *lu*, tipo *LocalUnit*
- CentralUnit tipo *CentralUnit*

Esta sintaxis es similar a la del SDL.

Los vínculos con el entorno se indican mediante flechas, que llegan hasta el símbolo de recuadro o lo rebasan. Las entidades y vínculos en el entorno pueden representarse fuera del símbolo de recuadro y estar ligadas a entidades que se encuentran dentro del recuadro.

Las especificaciones de realización especifican el sistema real, que es una realización del sistema SDL (véase la Figura I.7-5).

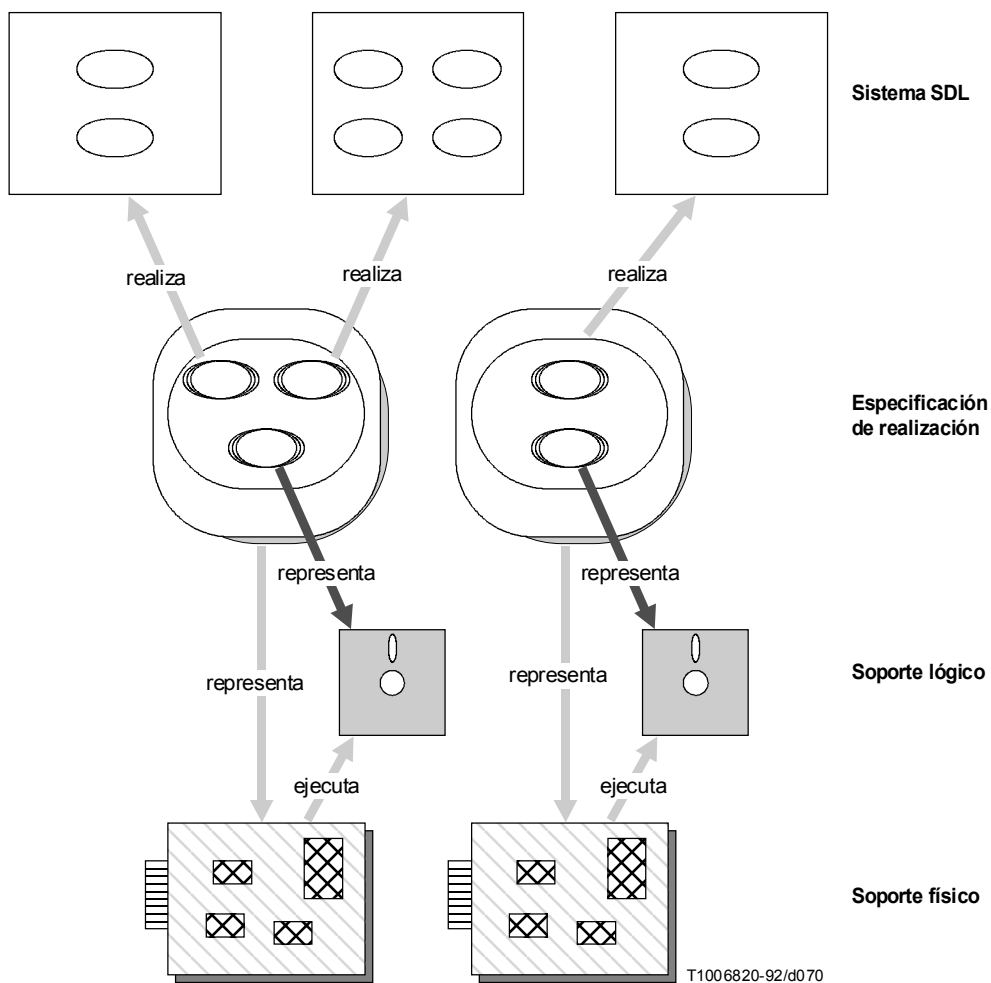


FIGURA I.7-5/Z.100  
Especificaciones de realización que especifican el sistema real

## Reemplazada por una versión más reciente

Para especificar la correspondencia entre el sistema SDL y el sistema real es necesario añadir información de correspondencia a los diagramas de estructura de soporte físico y soporte lógico. Hay que tener en cuenta varias correspondencias:

- código fuente es especificación de fuente *traducida*,
- código objeto es código de fuente *compilado*,
- código ejecutable es código de objeto *cargado*,
- el soporte físico *ejecuta* el código ejecutable,
- el soporte físico + el código ejecutable *realizan* la especificación fuente.

Hay que considerar todos los niveles para controlar y documentar plenamente el diseño y la realización. Para una documentación más práctica pueden omitirse algunos niveles.

Las casillas y las flechas de los diagramas de bloques de realización están relacionadas, por una parte, con las expresiones SDL y, por otra, con los objetos del mundo real que representan. Esto se especifica mediante *expresiones de correspondencia* en el diagrama, como se ilustra en la Figura I.7-3.

Para ahorrar espacio en casillas y flechas, se puede hacer referencia a las expresiones de correspondencia así:

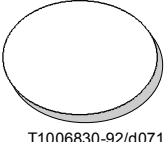
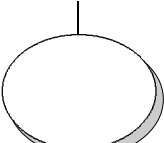
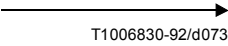
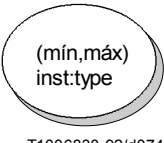
<1>	referencia de correspondencia
<1> = implement	expresión de correspondencia
[system AccessControl.LocalStation]	referenciada

Se utilizan los tipos de correspondencia siguientes [*node identifier* (identificador de nodo) identifica a la entidad conexas]:

- Realiza [node identifier]
- Realizado por [node identifier]
- Ejecuta [node identifier] (implica que el soporte lógico está cargado)
- Ejecutado por [node identifier]


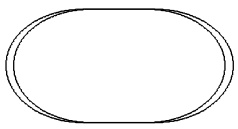
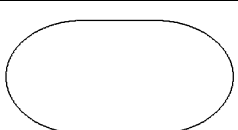
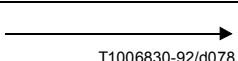
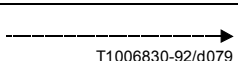
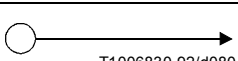
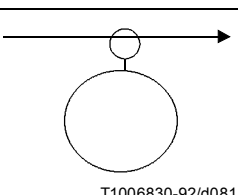
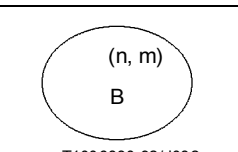
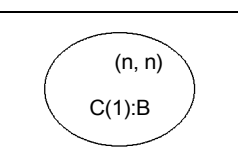
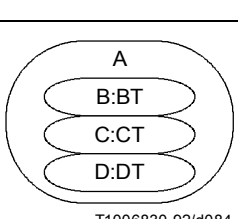
### I.7.3.3 Resumen de símbolos

#### Símbolos del diagrama de estructura de soporte físico

	Bloque de soporte físico	Símbolo general de bloque de soporte físico. Se puede descomponer recursivamente.
	Conmutador	Símbolo general de bloque de soporte físico que efectúa conmutación y encaminamiento. Se puede inclinar el símbolo en cualquier dirección, y alargar la línea para asociarlo a muchos enlaces de soporte físico.
	Conexión (flechas facultativas)	La flecha de la línea de conexión es facultativa. Las flechas indican el sentido de las señales. Puede haber flechas en ambos extremos. En el nivel más bajo, las conexiones son enlaces físicos.
	Conjunto de bloques	Se pueden agrupar los bloques de soporte físico en un conjunto de bloques del mismo tipo. Los bloques tendrán un tipo y pueden tener un identificador de instancia.

# Reemplazada por una versión más reciente

## Símbolos del diagrama de estructura de soporte lógico

 <p>T1006830-92/d075</p>	<p>Bloque de proceso de soporte lógico</p>	<p>Unidad de soporte lógico que contiene al menos un programa de no terminación. Puede contener procedimientos y datos.</p>
 <p>T1006830-92/d076</p>	<p>Bloque de procedimiento</p>	<p>Unidad de soporte lógico que contiene un programa de terminación, pero ningún programa de no terminación. Puede contener datos.</p>
 <p>T1006830-92/d077</p>	<p>Bloque de datos</p>	<p>Un elemento de datos puros o un grupo de elementos de datos. Puede no contener programas.</p>
 <p>T1006830-92/d078</p>	<p>Flujo de datos</p>	<p>La flecha indica el sentido del flujo.</p>
 <p>T1006830-92/d079</p>	<p>Activación</p>	<p>Activación o llamada. No puede conectar con bloques de datos.</p>
 <p>T1006830-92/d080</p>	<p>Referencia</p>	<p>Referencia o puntero.</p>
 <p>T1006830-92/d081</p>	<p>Flujo de datos con referencia a memoria tampón de mensajes</p>	<p>La línea denota el flujo de datos, y el cuadrado referenciado, la memoria tampón de mensajes.</p>
 <p>T1006830-92/d082</p>	<p>Conjunto de elementos de datos de tipo B</p>	<p>Un conjunto de mín, <math>n</math>, máx <math>m</math> elementos de tipo B.</p>
 <p>T1006830-92/d083</p>	<p>Matriz con índice de elementos de datos</p>	<p>Matriz de <math>n</math> elementos denominada C(1) - C(n) de tipo B.</p>
 <p>T1006830-92/d084</p>	<p>Estructura de elementos de datos; registro</p>	<p>Estructura de datos compuesta (A) que consiste en elementos de tipos diferentes. Cada elemento de datos tiene un tipo y un identificador de instancia.</p>

# Reemplazada por una versión más reciente

## I.7.4 Transacciones entre el soporte físico y el lógico

### I.7.4.1 Introducción

En estas directrices se destaca el diseño del soporte lógico, pero el soporte lógico y el soporte físico no se pueden diseñar independientemente el uno del otro. Por ejemplo, la capacidad de cumplir los requisitos temporales depende tanto de la velocidad del computador como del volumen de soporte lógico necesario para ejecutar una función determinada. En realidad, el soporte físico y el soporte lógico están estrechamente relacionados, y habrá que hacer transacciones entre ambos.

Una vez elegida una estructura adecuada de soporte físico, se consideran sus repercusiones en el diseño del soporte lógico. Por ejemplo, los periféricos afectan al soporte lógico de entrada-salida. Si las funciones están distribuidas entre muchos computadores, se necesitará soporte lógico adicional para la comunicación entre computadores y el tratamiento de errores distribuido. En consecuencia, habrá que conocer toda la estructura del soporte físico para poder diseñar el soporte lógico.

Tampoco se diseñará el soporte físico sin tener en cuenta la estructura del soporte lógico. Por ello, al diseñar ambos soportes será necesario la coordinación y efectuar transacciones. El primer paso del diseño de la realización consiste en efectuar estas transacciones y diseñar la arquitectura general del soporte físico y del soporte lógico teniendo en cuenta los requisitos no funcionales. Las consideraciones principales son:

- la distribución física y las interfaces físicas;
- las limitaciones temporales en función de la capacidad de procesamiento;
- los requisitos del tratamiento de errores, es decir, detección, aislamiento y recuperación tras error;
- seguridad con respecto al acceso no autorizado a la información;
- posibilidades de ampliación, funcionamiento y mantenimiento del soporte físico;
- costos de desarrollo, producción y mantenimiento;
- utilización de los componentes existentes.

### I.7.4.2 Distribución física e interfaces

La especificación SDL no deberá suponer prematuramente una división física interna del sistema, aunque a veces, la ubicación física de las interfaces entraña una distribución física del sistema. Por ejemplo, los abonados a un sistema de telecomunicación están distribuidos físicamente, lo que significa que por lo menos las interfaces de usuario estarán físicamente distribuidas.

Las señales SDL se definen con independencia de las distancias físicas. Por tanto, se puede ubicar procesos en unidades físicas separadas. Sin embargo, la transferencia de señales a lo largo de distancias siempre supondrá retardos de transmisión y gastos conexos. En consecuencia, se tratará de encontrar canales que transporten poco tráfico de señales sin limitaciones temporales estrictas. A veces, esos canales se pueden encontrar en las interfaces externas del sistema, pero con mucha más frecuencia estarán dentro del sistema.

Esto se puede generalizar en una regla que estipule que la distribución se efectuará por canales con pocas interacciones y limitaciones temporales mitigadas (pequeña anchura de banda). Los procesos fuertemente acoplados se mantendrán juntos, lo que con frecuencia significará que una buena cantidad de procesamiento se efectuará físicamente cerca de las interfaces externas.

En consecuencia, la realización puede dividir el sistema SDL en bloques y canales físicamente diferentes. Es probable que se necesite funcionalidad adicional para apoyar la distribución, por ejemplo, protocolos de transferencia de señales, tratamiento de errores.

### I.7.4.3 Carga de tráfico y tiempo de respuesta

El objetivo es encontrar soportes físicos y lógicos que puedan cumplir los requisitos relativos a carga de tráfico y tiempo de respuesta con un costo mínimo. Para ello, habrá que estimar los tiempos de proceso, y calcular luego la carga de procesamiento representada por la aplicación SDL y sus tiempos de respuesta.

Se puede calcular valores medios sencillos como sigue:

- a) para cada proceso SDL,  $P$ , calcular un tiempo de transición medio,  $t_p$ . Este valor depende del tamaño del soporte lógico y de la velocidad de ejecución del soporte físico, y deberá ser estimado. Por ejemplo, se puede calcular el número medio de operaciones  $o_p$  por transición (envío de señales, operaciones de temporizador, operaciones de datos), y luego estimar un número medio de instrucciones  $i_p$  por operación. Si la velocidad de ejecución del soporte físico es  $S$  instrucciones por segundo entonces  $t_p = i_p * o_p * S$ ;



## Reemplazada por una versión más reciente

- b) calcular el promedio de transiciones  $n_p$  que ejecutará cada proceso SDL por segundo con carga máxima. Este valor se puede obtener contando el número de transiciones necesarias para efectuar una función determinada, por ejemplo, tratar una llamada telefónica, multiplicado por el número de esas funciones que el proceso ejecutará por segundo.

- c) calcular la carga máxima media normalizada para cada proceso:

$$l_p = n_p * t_p$$

Esta es una medida del tiempo de procesamiento que necesitará este proceso por segundo o sea, la fracción de capacidad de procesamiento que necesita este proceso (unidad de medida: Erlang);

- d) calcular un valor de carga correspondiente para cada canal  $C$  y ruta de señales  $R$ :

$$l_c = n_c * t_c$$

$$l_r = n_r * t_r$$

donde  $n_c$  y  $n_r$  son el número de señales por segundo, y  $t_c$  y  $t_r$ , los tiempos de procesamiento por transferencia de señal;

- e) calcular la carga media del sistema como la suma de las cargas de canal, de ruta de señal y de proceso. Si este valor es mayor que uno, la carga media es superior a la capacidad de procesamiento de un solo computador. En ese caso, se deberá aumentar la capacidad, ya sea optimizando el soporte lógico, aumentando la velocidad del soporte físico o distribuyendo el sistema. Por regla general, la carga media de un computador no excederá de 0,3, para dejar un margen a las cargas máximas estadísticas.

Obsérvese que la carga real variará estadísticamente, y tendrá valores máximos considerablemente superiores a los valores medios considerados. Por tanto, el sistema podrá estar sobrecargado durante periodos de tiempo, aunque se haya dejado márgenes suficientes para el tratamiento de la carga media. Por esta razón, el diseño de la realización debe incluir una estrategia de control de carga.

Si el sistema puede funcionar en un solo computador, esto será preferible (a menos que otras consideraciones exijan otras cosas). Si por razones de calidad de funcionamiento, no se basta un solo computador, el sistema lo distribuirá. Esto añadirá taras de comunicación, que se deben incluir en los cálculos de carga revisados.

Además de calcular la carga, es necesario comprobar que el sistema cumplirá las limitaciones de tiempo real. De los diagramas SDL se obtendrá el número de transiciones y transferencias de señal por transacción dependiente del tiempo, y se calculará los tiempos de procesamiento correspondientes. Una vez más, hay que dejar márgenes para las variaciones estadísticas.

Las interfaces de soporte físico/soporte lógico requieren atención especial. Es normal que la mayor parte de la capacidad de un computador se utilice para entradas y salidas. Por ello, resulta muy provechoso diseñar cuidadosamente la interfaz de entrada-salida. Los canales tienen sincronización dependiente del tiempo, originan una clase especial de restricciones de temporización, véase I.7.2.2.

### I.7.4.4 Fiabilidad, protección y seguridad

Los requisitos relativos a la fiabilidad pueden influir en el soporte físico de varias maneras:

- la tolerancia de averías significa redundancia. Se necesita, por lo menos, dos unidades de soporte físico y dispositivos de detección de errores, diagnóstico y conmutación para la tolerancia de averías;
- el seccionamiento de averías significa distribuir las funciones entre unidades de soporte físico diferentes para limitar el número de procesos SDL que pueden ser bloqueados por un solo error de soporte físico;
- la protección contra fallos significa que el sistema debe fallar hasta un estado de salida de protección, donde no cause daño a su entorno. Normalmente, se necesitará algún tipo de dispositivo de supervisión.

La protección en el caso de acceso no autorizado a la información o modificación de ésta también exige algunas medidas especiales en el soporte físico.

# Reemplazada por una versión más reciente

## I.7.4.5 Modularidad y reutilización

En el diseño del soporte físico deberán tomarse en consideración los gastos de producción del soporte físico más que en el diseño del soporte lógico. Los gastos dependen tanto de la complejidad del soporte físico, como del volumen de producción. Un gran volumen de producción significa menor costo por unidad. Por ello, habrá que minimizar la cantidad de diseños diferentes para aprovechar esa ventaja. Con frecuencia, se podrá encontrar un soporte físico genérico para realizar una amplia gama de sistemas SDL.

## I.7.4.6 Arquitectura del soporte físico

De lo antedicho se desprende que es necesario definir la arquitectura general del soporte físico necesario para realizar el sistema SDL. Esto se debe documentar mediante diagramas de estructura de soporte físico, como se muestra en la Figura I.7-3. En esta etapa, el acento recae en la estructura global del soporte físico, es decir, computadores, periféricos y canales de comunicación.

Se definirán los protocolos, los formatos de señal y los esquemas de sincronización de todas las interconexiones físicas, pues ésta es una contribución importante al diseño del soporte lógico.

Por último, se documentará la asignación de procesos SDL a unidades físicas. Una vez hecho esto, se conocerá la funcionalidad y el entorno físico del soporte lógico en el sistema.

## I.7.5 Diseño de la arquitectura del soporte lógico

### I.7.5.1 Principios de diseño

Existen diferencias semánticas considerables entre el SDL y la mayoría (si no la totalidad) de los lenguajes de programación:

- a) **Concurrencia** – Los lenguajes de programación secuenciales, como C y PASCAL, no admiten la concurrencia del SDL. Otros, como CHILL y ADA, si la admiten, pero de manera diferente.
- b) **Tiempo** – Muy pocos lenguajes de programación sustentan tiempo. Ninguno sustenta directamente tiempo del tipo SDL, salvo, quizás, el CHILL.
- c) **Comunicación** – Ningún lenguaje admite comunicación de señales de tipo SDL.
- d) **Comportamiento secuencial** – Un gráfico de procesos SDL especifica el comportamiento de transiciones de estados como una máquina de estados finitos ampliada. Los lenguajes de programación especifican secuencias de acciones.
- e) **Datos** – Los datos SDL son abstractos y, posiblemente, infinitos. En un lenguaje de programación, la realización tiene que ser operacional y finita.

Una manera común de superar esas diferencias es adaptar la máquina subyacente y el lenguaje de programación a la semántica SDL con soporte lógico de apoyo. En general, se utiliza tres niveles de apoyo (véase la Figura I.7-6):

- a) Ninguna – Los conceptos SDL se hacen corresponder directamente con los conceptos de un lenguaje de programación secuencial.
- b) Facilidades básicas de concurrencia, tiempo y comunicación suministradas por un sistema operativo en tiempo real – Esto también incluye el apoyo del tiempo de ejecución previsto para lenguajes concurrentes, como CHILL y ADA.
- c) Apoyo adicional para los conceptos SDL además de los indicados en b).

Si bien la utilización de un sistema operativo es el método más común en la actualidad, hay casos en los que la tara que conlleva es inaceptable, ya sea por limitaciones de velocidad, capacidad de memoria o costo.

# Reemplazada por una versión más reciente

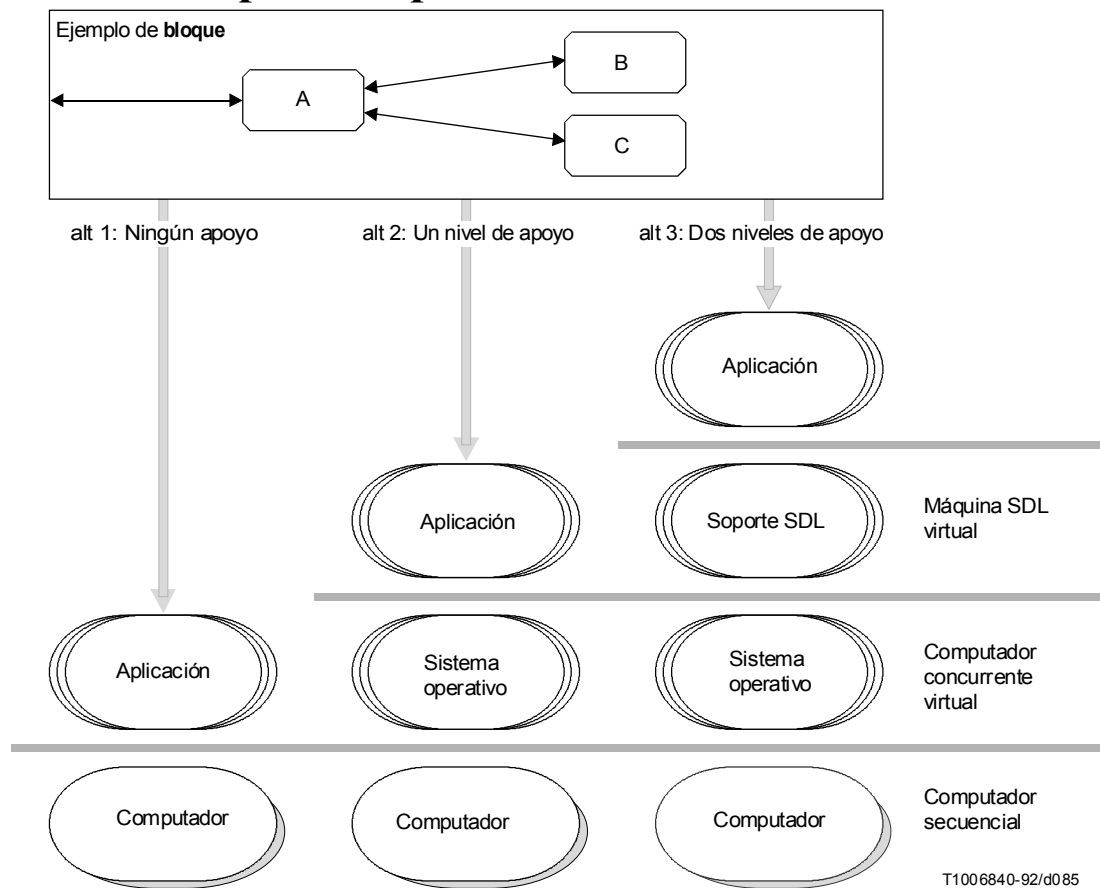


FIGURA I.7-6/Z.100  
Niveles alternativos de soporte lógico de apoyo

## I.7.5.2 Concurrencia y tiempo

### Niveles de apoyo

Cuando se aplican varios procesos SDL en el mismo computador con un lenguaje de programación secuencial, por ejemplo, el C, la concurrencia del SDL se debe aproximar mediante un comportamiento secuencial. En principio, esto se puede lograr transformando una especificación SDL, que contiene varios procesos, en una especificación equivalente, que contiene sólo uno, y después aplicar este proceso como un programa secuencial. Debido al problema que plantea la «explosión de estados», y la falta de modularidad de este método sólo es práctico en muy pocos casos. Por ello, normalmente se buscará una realización que mantenga la estructura de proceso original de la especificación SDL. Esto significa algún tipo de apoyo para la organización de los procesos y su comunicación.

En un lenguaje puramente secuencial no se apoya en general la concurrencia, pero las llamadas de procedimiento proporcionan un mecanismo combinado de comunicación y organización, que se puede utilizar para ciertos casos especiales de comunicación y concurrencia SDL. (Entonces, la comunicación asíncrona del SDL se realiza mediante comunicación síncrona.) En consecuencia, el nivel más bajo de apoyo consiste en utilizar llamadas de procedimiento como mecanismo básico de organización y comunicación. Este método se expone en I.7.5.3.

Para realizar sistemas SDL más generales, hace falta introducir un esquema de comunicación asíncrona con almacenamiento en memoria tampón. Esto se consigue dentro del marco de un sistema de programación puramente secuencial mediante, por ejemplo, un programa principal encargado de organizar la activación de los procedimientos que aplican procesos SDL que comunican por medio de memorias tampón de mensajes. La limitación de este método radica en su capacidad para tratar las restricciones de tiempo y tiempo real.

# Reemplazada por una versión más reciente

Un sistema operativo para fines generales, que organiza los procesos concurrentes conforme a prioridades y apoya la comunicación y temporización de tipo SDL, proporcionará la plataforma más fácil y general para realizar la concurrencia de sistemas SDL. Las facilidades básicas necesarias de un sistema operativo en tiempo real son:

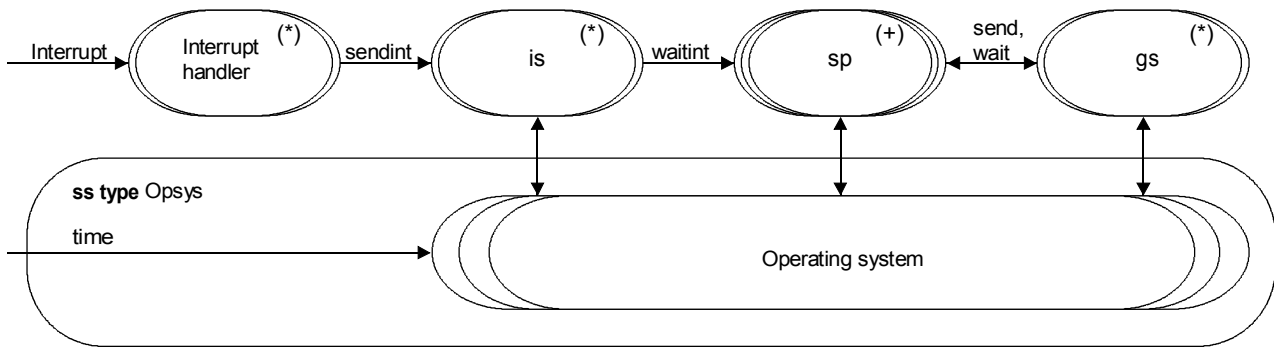
- Multiplexación y organización de procesos, es decir
  - 1) cambios de contexto entre procesos;
  - 2) organización por prioridades con y sin apropiación para satisfacer los requisitos de tiempo de respuesta;
  - 3) interrupción del tratamiento para facilitar la espera pasiva de eventos externos.
- Sincronización de interacciones, es decir
  - 1) comunicación;
  - 2) acceso a recursos compartidos.
- Medición de tiempo.

Muchos sistemas operativos comercializados ofrecen estas facilidades de una u otra manera. También están sustentados por lenguajes de programación concurrentes, como CHILL y ADA. A continuación, se describe un pequeño sistema operativo que apoya esas funciones, que servirá de ejemplo y marco de referencia.

Además del sistema operativo básico, se necesitarán facilidades para la comunicación y temporización de tipo SDL para la sustentación general del SDL.

## Ejemplo de sistema operativo

El sistema operativo ve al sistema de soporte lógico como una colección de *procesos de soporte lógico* y *semáforos* generales. Ejecuta la multiplexación y organización de los procesos a partir de los eventos externos e internos. Esos eventos consisten en interrupciones externas (incluidas las interrupciones temporales), u operaciones internas en los semáforos (véase la Figura I.7-7).



sp Proceso de soporte lógico  
gs Semáforo general  
is Semáforo de interrupción

FIGURA I.7-7/Z.100

## Ejemplo de sistema operativo

Los semáforos administran memorias tampón y procesos de espera, y se utilizan en la comunicación de mensajes y para representar los recursos compartidos asignados por un semáforo. Las memorias tampón también son, en sí mismas, recursos compartidos. Antes de enviar un mensaje, se debe asignar una memoria tampón libre de un conjunto de memorias libres. El semáforo general es un mecanismo que se puede utilizar para asignar memorias tampón libres, que quizás representan otros recursos, y para proporcionar comunicación asíncrona.

# Reemplazada por una versión más reciente

Las operaciones que se efectúan en las semáforos generales son:

- **send**(semaphore, buffer);
- **wait**(semaphore, max-time) → (buffer, time).

La operación *wait* (*espera*) puede especificar un tiempo máximo de espera. Si no se pone a disposición una memoria tampón antes de que expire el temporizador, la operación será devuelta con una indicación de temporización. Mediante la espera en un semáforo especial, *suspend* (*suspensión*) que nunca devuelve una memoria tampón, el proceso puede suspenderse a sí mismo durante un periodo especificado.

Se utiliza un tipo especial de semáforo para indicar las interrupciones de señales y esperar interrupciones:

- **sendint**(semaphore);
- **waitint**(semaphore, max-time) → (time).

Se puede considerar el semáforo general como un tipo de dato abstracto con dos operaciones, *send* y *wait*, realizadas mediante llamadas de procedimiento. Utilizará una estructura de datos, por ejemplo, una lista enlazada, para mantener una cola de memorias tampón. También mantendrá una lista de referencias para esperar procesos de soporte lógico cuando no haya memorias tampón en la cola de éstas (véase la Figura I.7-8).

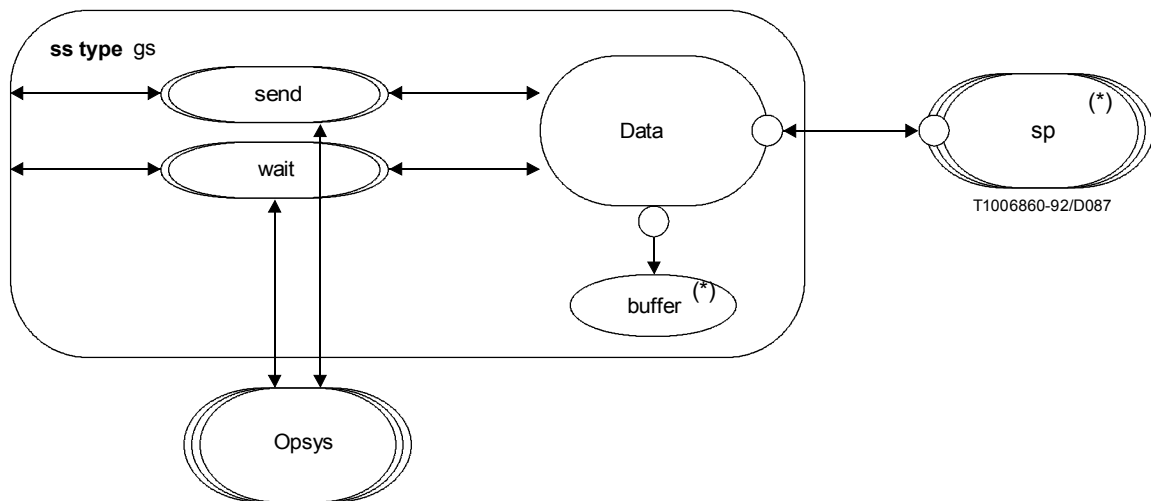


FIGURA I.7-8/Z.100  
Semáforo general

Esta clase de semáforo es una generalización de los semáforos clásicos descritos en las publicaciones [17]. Los mensajes/memorias tampón libres se ponen en la cola mediante una operación *send* (*envío*), y se sacan mediante la operación *wait*.

Para realizar la comunicación de tipo SDL se puede codificar las memorias tampón con señales SDL.

La temporización en SDL difiere de la suspensión de procesos, puesto que los procesos SDL pueden estar activos durante las transiciones, mientras está funcionando un temporizador. Para esto, el sistema operativo proporciona una facilidad de temporizador de tipo SDL:

- **starttimer**(time, timer-id, PId) → message(PId, timer-id);
- **stoptimer**(timer-id, PId);
- **now** → time.

Básicamente, la facilidad de temporizador opera sobre una lista de temporizadores, mientras los procesos correspondientes esperan recibir un mensaje de temporización. Mediante *starttimer*, un proceso ordena a la facilidad de temporizador que haga una anotación en su lista de temporizadores y comience el cómputo y la prueba contra el tiempo. Cuando se alcanza el tiempo especificado, se envía el mensaje de temporización y se suprime la anotación de la lista.

## Reemplazada por una versión más reciente

De forma semejante *stoptimer* ordena suprimir la anotación de la lista. ¿Qué sucederá si se emite *stoptimer* después de enviar la temporización al proceso? El proceso se encontrará en un estado nuevo, en el que no esperará la temporización y no habrá un contador para suprimir de la lista de temporizadores. En este caso, la mejor solución es dejar que *stoptimer* suprima el mensaje de temporización donde quiera que se encuentre.

Cada proceso de soporte lógico tendrá una descripción de proceso, que el sistema operativo utiliza durante la organización y cambio de contexto. En la Figura I.7-9 se muestra cierta información de la descripción de proceso.

El organizador llevará una lista de procesos preparados para ejecución, y siempre arrancará el de prioridad más alta (en el caso de prioridades iguales, puede alternarlos). Cuando se ejecuta una operación *send* o *wait*, puede suceder que algunos procesos estén preparados para ejecución y/o que el proceso en ejecución tenga que esperar. En consecuencia, cada operación de semáforo puede hacer que se active un proceso nuevo o se detenga el proceso en curso. Los interruptores pueden tener el mismo efecto mediante el uso de las operaciones *sendint* y *waitint*. Por añadidura, las interrupciones pueden causar temporizaciones.

Además del conjunto básico de operaciones descrito, algunas aplicaciones pueden necesitar operaciones para crear y borrar dinámicamente procesos y semáforos de soporte lógico.

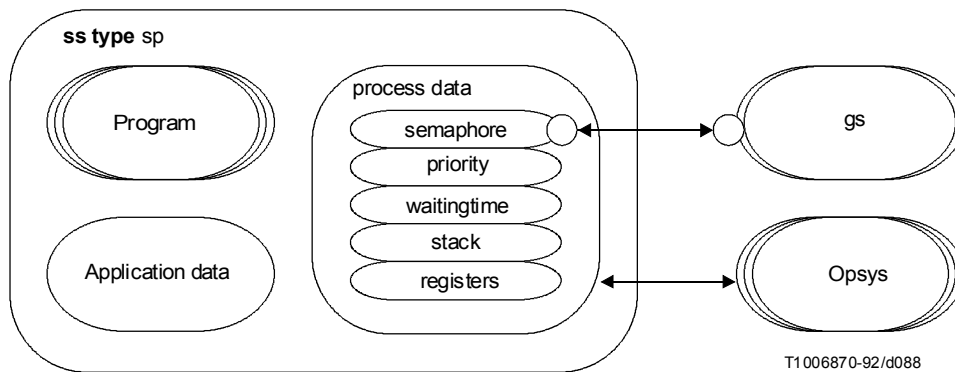


FIGURA I.7-9/Z.100

### Un proceso de soporte lógico

#### Prioridad

En el SDL no hay prioridad entre procesos. Esto no quiere decir que no se pueda utilizar la prioridad en la realización. La prioridad es necesaria por varios motivos:

- hay restricciones externas de tiempo real que sólo se pueden cumplir otorgando prioridad alta a ciertos procesos;
- hay situaciones internas de error que deberán ser tratadas con la prioridad más alta;
- hay situaciones en las que la prioridad ayuda a simplificar y acelerar los cálculos.

Normalmente, las restricciones de tiempo real se deben cumplir asignando la prioridad más alta a los procesos críticos con respecto al tiempo. Por ello, habrá que tratar las entradas/salidas críticas con respecto al tiempo con la prioridad más alta y dejar que el procesamiento interno prosiga con prioridad más baja. Esta es una de las razones por las que las entradas/salidas se deben separar de los procesos de aplicación.

En las situaciones de error, habrá que reaccionar rápidamente para aislar el error, reducir el daño y dar la alarma. Esto significa que hay que efectuar algún procesamiento inmediatamente, y comunicar los errores lo más pronto posible. Para conseguir esto último, no basta con que los procesos tengan prioridad alta; también se necesita prioridad en la transferencia de mensajes. El sistema operativo puede lograrlo asignando un atributo de prioridad a cada mensaje, o enviando mensajes de alta prioridad a través de semáforos que son atendidos con prioridad alta.

## Reemplazada por una versión más reciente

Los servicios SDL funcionan casi en paralelo y comunican mediante señales de prioridad. Esto se puede realizar otorgando a los servicios prioridades iguales y dejando que sus señales internas tengan prioridad sobre las señales externas.

Si los mensajes internos de un sistema de soporte lógico son prioritarios con respecto a los mensajes externos procedentes del entorno, el orden interno de procesamiento será más determinístico. Esto ayuda a reducir el número de maneras en que los procesos se entrelazarán efectivamente. A su vez, disminuye la probabilidad de ocurrencia de ciertos errores de interacción. Más importante aún, a veces ayuda a reducir la tara y permite aumentar la velocidad de las comunicaciones internas. Por último, se mejora el control de carga cuando se da prioridad a las peticiones de servicio ya aceptadas con respecto a las nuevas.

### I.7.5.3 Comunicación

En las transacciones entre soporte físico y soporte lógico (véase I.7.4), se consideró primero las interfaces físicas del sistema para llegar gradualmente hasta la estructura interna. En el diseño del soporte lógico se hará lo mismo comenzando por las interfaces de soportes físico-lógico hasta llegar a la estructura interna del soporte lógico.

El esquema de comunicación utilizado en las interfaces físicas influye decisivamente en la estructura global del soporte lógico. Las necesidades de comunicación interna y externa también afectan la forma en que se podrá realizar la comunicación en un sistema de soporte lógico. Serán igualmente afectadas las necesidades de organización, apropiación y sincronización.

#### Entrada/salida

Con frecuencia, hay que hacer conversiones en la interfaz de soporte físico-lógico, no sólo para convertir datos de un formato a otro sino, también, para la conversión entre valores continuos y valores secuenciales. Por ejemplo, puede ser necesario explorar las variables externas a intervalos para detectar eventos y generar mensajes de tipo SDL para la comunicación interna.

Las restricciones efectivas de concurrencia y temporización presentes en la interfaz soporte lógico-soporte físico tienen que ser concordadas por el casi paralelismo y las prioridades del sistema de soporte lógico. En general, las interacciones de entrada/salida necesitan prioridad alta para

- garantizar que todos los eventos de entrada son detectados, en el caso de la interacción dependiente de la velocidad,
- utilizar eficazmente los canales de entrada/salida lentos, y
- reducir los tiempos de respuesta.

Las interrupciones son los medios básicos para suministrar prioridad de apropiación y habilitar la espera pasiva en los eventos externos. La espera pasiva economiza recursos de computador, pero las interrupciones introducen un orden de ejecución no determinístico en el sistema de soporte lógico, que hace que las condiciones se asemejen al procesamiento verdaderamente paralelo. Por consiguiente, se necesita la sincronización en las acciones internas que se efectúan entre unidades de soporte lógico interrumpidas e interruptoras (de ahí el uso de semáforos de interrupción).

En razón de las prioridades y la velocidad que se necesita para satisfacer los requisitos de tiempo real y calidad de funcionamiento, la entrada/salida se realizará, con frecuencia, mediante procesos separados de soporte lógico, como se muestra en la Figura I.7-10.

Esto sirve para ocultar las particularidades de la interfaz entrada/salida de la parte de aplicación que realiza el proceso SDL. La realización de los procesos SDL se analiza en I.7.5.4.

En la Figura I.7-10 se analiza la comunicación de tipo SDL realizada mediante la transferencia de mensajes a través de semáforos, aunque ésta no es la única manera posible.

#### Llamadas de procedimiento

Se puede utilizar llamadas de procedimiento para realizar señales SDL directamente. Cada tipo de señal se puede representar como un procedimiento que pertenece al proceso receptor. Por ejemplo, es posible realizar la señal «Open\_door(here)» mediante el procedimiento OPEN\_DOOR(HERE). Esto entraña que el proceso SDL receptor es realizado por un conjunto de procedimientos y datos.

A manera de ejemplo, considérense tres procesos SDL, *P1*, *P2* y *P3*, representados en la Figura I.7-11.

## Reemplazada por una versión más reciente

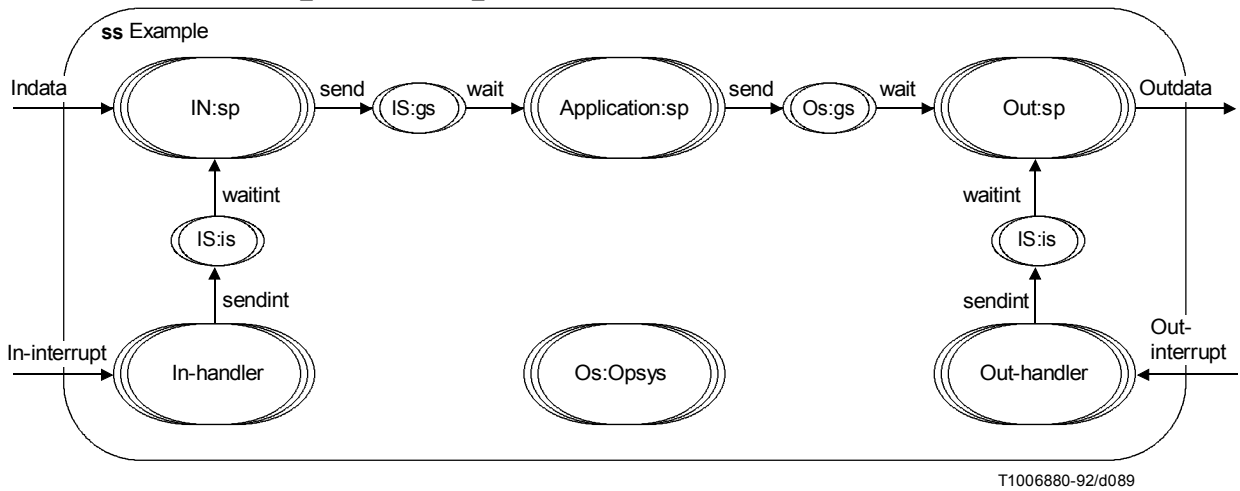


FIGURA I.7-10/Z.100

Realización de entrada/salida mediante procesos de soporte lógico separados

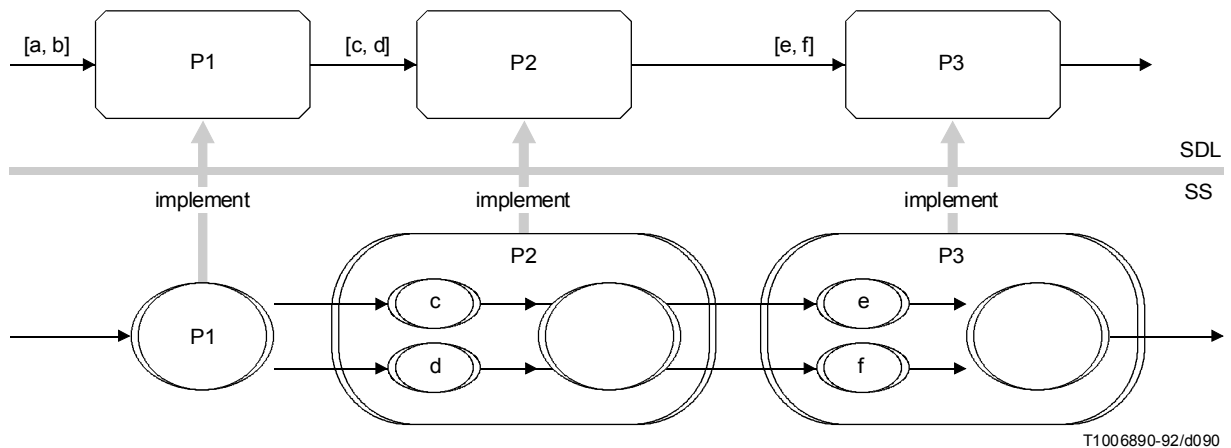


FIGURA I.7-11/Z.100

Señales realizadas como procedimientos

Para cada proceso, puede existir un procedimiento para cada señal de entrada, o un procedimiento común con el tipo de señal codificado como parámetro. En cualquiera de los dos casos, la activación de los procesos se hará después de las llamadas de procedimiento. Cuando se envía una señal, el receptor tendrá la prioridad de apropiación con respecto al emisor, y terminará sus transiciones antes de que el control sea devuelto al emisor.

Para que esta solución funcione, el sistema SDL debe estar estructurado y comportarse como un árbol de llamadas de procedimiento. Es preciso contar con el tiempo suficiente entre cada señal de entrada o temporización para efectuar todo el procesamiento y la espera que debe ser realizada en respuesta. No debe ser necesario otorgar prioridad de apropiación a los procesos distinta de la que indica implícitamente el árbol de llamadas. Para cada señal enviada hacia abajo en el árbol, no se devolverá más de una señal de respuesta (realizada como valor de devolución de procedimiento). En la práctica, toda espera no determinística de «nuevas» entradas y temporizaciones será ejecutada por un proceso. La función de los demás procesos es efectuar el procesamiento secundario y la salida que ocasiona una nueva entrada.



# Reemplazada por una versión más reciente

Las llamadas de procedimiento pueden considerarse un caso especial de la comunicación síncrona cuya organización es tal que el receptor tiene la prioridad de apropiación con respecto al emisor. Esto impone una restricción bastante severa a la utilización del SDL. Por ello, esta solución sencilla y rápida no funcionará en todos los casos. La comunicación SDL es más general y flexible que las llamadas de procedimiento.

## Comunicación con memoria tampón

La sincronización SDL es infinitamente elástica, lo que significa que el emisor puede aventajar al receptor en infinitamente muchas señales. Como ya se ha dicho, esta situación no puede ocurrir en la realidad. En la práctica, la cola debe estar limitada, y en el caso de una cola completa, el emisor tendrá que esperar o las señales se perderán. Hace falta un diseño cuidadoso en este punto para garantizar un control de carga uniforme sin degradaciones considerables de la calidad en situaciones de desbordamiento.

En el esquema de comunicación de la Figura I.7-12 se utilizan los semáforos generales presentados en I.7.5.2. Las señales SDL son codificadas en las memorias tampón y pasadas como mensajes a través del semáforo *S*. Las memorias tampón libres son administradas por otro semáforo *F*. El esquema es bastante general, y se ha utilizado para realizar muchos sistemas en tiempo real especificados en SDL. Se considerarán dos aspectos:

- *El control de la carga* – Las memorias tampón en circulación ofrecen la oportunidad de efectuar una forma sencilla de control de carga. En situaciones de sobrecarga, la cola de mensajes entre el emisor y el receptor aumentará hasta utilizar todas las memorias tampón libres. Esto impide al emisor generar más mensajes antes de que el receptor haya podido procesar algunos de los mensajes que ya están en la cola. Mediante el dimensionado cuidadoso de los conjuntos de memorias tampón libres, se podrá controlar la carga interna en un sistema de soporte lógico.
- *Las posibilidades de atasco* – Cuando varios procesos compiten por los mismos recursos, pueden producirse atascos. Sea una situación de sobrecarga, en la que el conjunto de memorias tampón libres de la Figura I.7-12 está vacío. Si el receptor necesita otra memoria tampón del conjunto, el sistema puede bloquearse, al estar ambos procesos en espera de una memoria tampón libre. Por consiguiente, es preferible utilizar conjuntos de memorias tampón libres separados para evitar este problema.

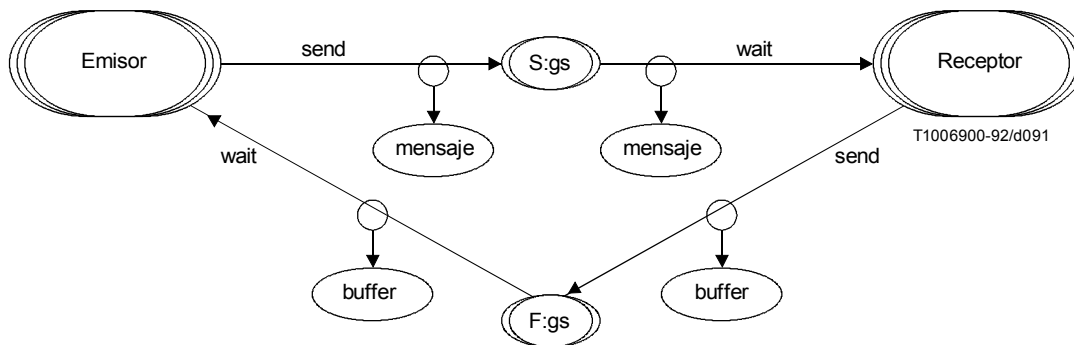


FIGURA I.7-12/Z.100

## Comunicación mediante memorias tampón de mensajes

En la Figura I.7-12, los mensajes son desplazados mediante punteros de transferencia, y el tamaño del tampón está determinado por el conjunto de tampones libres. Otra posibilidad consiste en mantener los tampones dentro del semáforo de comunicación y copiar los mensajes. Cuando se produce la sobrecarga, la operación de envío debe demorar al emisor.

La desventaja de la comunicación con memorias tampón estriba en que, en muchos casos, es más lenta que las llamadas de procedimiento directas, y en que no es sustentada por muchos lenguajes de programación (el CHILL es una excepción). Por consiguiente, se necesita soporte lógico adicional para la comunicación con memoria tampón, y recursos informatizados para el procesamiento adicional. Las primitivas de comunicación asíncrona que proporcionan muchos sistemas operativos en tiempo real serán, con frecuencia, suficientes.

# Reemplazada por una versión más reciente

Como se ha explicado en I.7.2.2, hay dos clases de información que se quiere comunicar: valores continuos y secuencias de valores.

La comunicación mediante una sola memoria tampón, o sea, una variable compartida, es la manera más directa de transmitir valores continuos. El productor puede fijar el valor cuando debe ser cambiado, y los usuarios pueden leerlos cuando haga falta (véase la Figura I.7-13). La lectura de un valor vigente sólo se puede efectuar cuando el usuario está activo y ejecuta una operación de lectura. No habrá espera, porque el usuario quiere conocer el valor en el instante de la lectura.

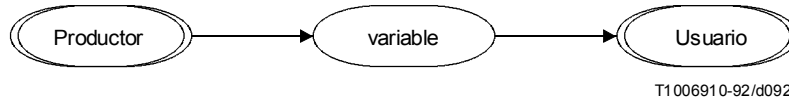


FIGURA I.7-13/Z.100

## Comunicación mediante una variable compartida

Cuando el productor y el usuario son procesos concurrentes, las operaciones de escritura y lectura deben ser mutuamente exclusivas. Esto se puede conseguir de diversas maneras:

- a) Garantizando que las operaciones de escritura y lectura son atómicas entre sí:
  - 1) utilizando instrucciones de primitivas,
  - 2) desactivando las interrupciones,
  - 3) mediante una organización tal que el emisor y el receptor nunca se interrumpan mutuamente.
- b) Controlando acceso a la variable compartida mediante un asignador de recursos. El productor y el usuario deberán solicitar al asignador el derecho de acceso antes de comenzar una operación en la variable compartida, y devolverlo una vez terminada la operación. Esto puede conseguirse con operaciones *wait* y *send*, como se muestra en la Figura I.7-14.

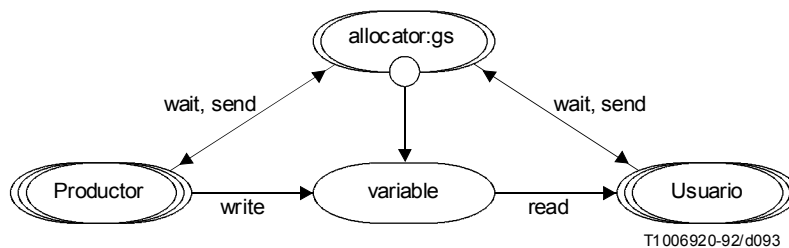


FIGURA I.7-14/Z.100

## Comunicación a través de una variable compartida, controlada por un asignador de recursos

### Encaminamiento

Cuando un proceso SDL envía una señal, ésta debe ser encaminada de alguna manera hacia el receptor. Las señales SDL identifican a su emisor receptor y mediante valores PID, y son encaminadas por los bloques según *quién* es el receptor y no *dónde* está, salvo, quizás, cuando la ruta se especifica con la cláusula *via*. Basta con esto en el nivel abstracto, pero en la realización, hay que conocer la ubicación física del receptor para encaminar la señal correctamente desde el emisor hasta el proceso receptor.

# Reemplazada por una versión más reciente

La ubicación física del proceso la determinan el computador físico, el proceso de soporte lógico dentro del computador y, posiblemente, la dirección local dentro del proceso de soporte lógico.

No obstante, debe ser un objetivo mantener el soporte lógico lo más independiente posible de la ubicación física de los procesos. Un proceso tendrá los conocimientos mínimos sobre el trayecto por el que se encamina una señal para llegar a su destino. Idealmente, sólo debe saber *quién* es el receptor, y no *dónde* está.

Hasta cierto punto, esto se consigue mediante el concepto de variables PID. La estructura de datos de un proceso contendrá variables PID, que representan los procesos a los que puede enviar señales. Esas variables son vinculadas en el momento de la creación de procesos o dinámicamente durante el comportamiento del proceso. Gracias al concepto de tipos de datos abstractos, la representación real de los valores PID se puede ocultar al código de proceso.

Sin embargo, la realización de los tipos de datos PID dependerá de la manera en que se representen los valores PID. El SDL no define cómo se generan o representan los valores PID. Ello se deja al diseñador. En consecuencia, una de las principales cuestiones del diseño consiste en determinar cómo se representarán y asignarán los valores PID.

Una solución común consiste en representar los valores PID (direcciones de señal) mediante un identificador de tipo de proceso y otro identificador de instancia de proceso. El resultado será una dirección lógica, no una dirección física. Por consiguiente, se necesitará más apoyo para establecer la correspondencia entre direcciones lógicas y ubicaciones físicas.

En la Figura I.7-4 hay un bloque de soporte lógico separado denominado *Message routing (Encaminamiento de mensajes)*, que ejecuta el encaminamiento en el nivel de proceso de soporte lógico. Su objetivo es ocultar las direcciones físicas a los procesos de aplicación, lo que permite encaminar los mensajes según las direcciones de destino lógicas. Por ello, no es necesario que un proceso SDL sepa dónde están ubicados los demás procesos. Basta con que conozca sus identificadores lógicos. Además, la realización de los tipos de dato PID son independientes de la estructura de dirección física. El sistema de encaminamiento utilizará el identificador lógico y un mapa de direcciones para elegir un semáforo a través del cual enviar el mensaje (véase la Figura I.7-15). Así pues, los conocimientos sobre el encaminamiento físico están centralizados con respecto al mapa de direcciones.

Si no hace falta ocultar las direcciones físicas, no se necesita un sistema de encaminamiento separado.

Una ventaja de este método consiste en que resulta sencillo reencaminar los mensajes en el caso de extensiones de sistema en línea.

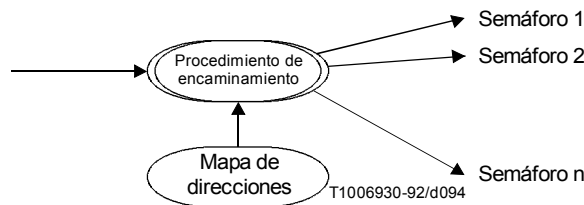


FIGURA I.7-15/Z.100

## Procedimiento de encaminamiento

### I.7.5.4 Comportamiento secuencial

Se examinará primero la realización de las máquinas de estados finitos ampliadas (FSM, *finite state machines*) en general, y luego se analizarán las características especiales del SDL, o sea, servicios, procedimientos, decisiones y la creación dinámica de procesos.

El comportamiento de las FSM de los procesos SDL se puede determinar de muchas maneras. Una consiste en codificar la parte de control directamente en el lenguaje de programación mediante expresiones **if-then-else** o **case**. Otra, en codificar la parte de control en tablas de transiciones de estados. Entre estas dos soluciones existe toda una gama de soluciones intermedias.

# Reemplazada por una versión más reciente

## Realización de las máquinas de estados finitos mediante código directo

En este enfoque, hay dos maneras de representar el estado:

- por una posición en el texto del programa;
- por un valor almacenado en una variable.

Analicemos primero el caso de la posición en el programa. El principio aplicado es la codificación directa del diagrama de transición de estados en el lenguaje de programación. Puesto que el programa es una buena definición de la lógica, se deberá buscar una correspondencia directa. Por consiguiente, no se tratará de que el diagrama sea un programa sin **goto**, sino de utilizar **goto** para pasar al estado siguiente.

Una realización típica es la siguiente:

```
begin
  state-1: wait(input);
  case input of
    signal-a:      call Action-1; goto state-3;
    signal-b:      call Action-2; goto state-5;
    else:          call Action-3; goto state-1;
  end case;
  state-2: wait(input);
  case input of
    signal-c: ...
  ...
end
```

Debido a la velocidad de ejecución exigida, ésta puede ser la solución preferida para los procesos entrada/salida. Cuando las decisiones pueden afectar al estado siguiente, se deberá añadir los enunciados «goto state» necesarios a los procedimientos «Action». Se puede apoyar directamente los procedimientos SDL si se utilizan procedimientos en el lenguaje de programación.

Dado que el estado está representado por la posición en el programa, esta técnica entraña que cada proceso SDL se realiza como un proceso de soporte lógico separado, por ejemplo, un proceso CHILL. Esto puede necesitar demasiado espacio si se trata de un número considerable de procesos.

La otra posibilidad es almacenar el estado en una variable. Esto resultará casi siempre en un programa que espera la entrada en un solo lugar:

```
repeat forever
begin
  wait (input);
  case state of
    state-1:
      case input of
        signal-a:      call Action-1; state:= state-3;
        signal-b:      call Action-2; state:= state-5;
        else:          call Action-3; state:= state-1;
      end case;
    state-2:
      case input of
        signal-c: ...
      ...
    end case;
  end;
end;
```

Si las decisiones pueden modificar el estado siguiente, la asignación de estado siguiente se colocará dentro de los procedimientos «Action» mencionados más arriba. También resulta fácil apoyar los procedimientos SDL en este caso, siempre y cuando se acepte esperar la entrada dentro de los procedimientos.

Puede ampliarse este enfoque para tratar muchos procesos SDL dentro de un proceso de soporte lógico si se introduce una matriz de estados indizada por el identificador de proceso. La multiplexación de muchos procesos SDL en un proceso de soporte lógico puede ayudar a reducir la tara de espacio y tiempo asociada con el sistema operativo. La comunicación interna entre procesos SDL que funcionan en el mismo proceso de soporte lógico puede ser considerablemente más rápida que la comunicación con sincronización y cambio de contexto entre procesos de soporte lógico.

Una desventaja estriba en que la realización de los procedimientos SDL generales se torna un poco más difícil. Cuando los procedimientos SDL contienen estados, se plantea la necesidad de seguir la pista de las direcciones de retorno. Si el programa es compartido entre muchas instancias de proceso, esto se hará explícitamente mediante una pila de direcciones de retorno.

# Reemplazada por una versión más reciente

En el caso de comunicación mediante llamadas de procedimiento, hay que almacenar el estado en una variable. En esta técnica, habrá un procedimiento que corresponde a cada señal de entrada al proceso. Cada procedimiento efectuará primero una prueba de estado, y luego efectuará la transición correspondiente. También en este enfoque hay que prestar especial atención a los procedimientos SDL generales.

## Realización de las máquinas de estados finitos mediante tablas

Puesto que el modelo de procesos del SDL se basa en el modelo de máquina de estados finitos ampliada (EFSM, *extended finite state machine*), todos los procesos SDL comparten las características comunes de las máquinas de estados finitos ampliadas.

Esas máquinas se adecúan bien a la realización dirigida por tablas. Su comportamiento se puede definir convenientemente mediante una tabla de transición de estados, que equivale a la forma gráfica utilizada en los diagramas de transición de estados. Ese tipo de tabla se puede realizar con una matriz bidimensional indizada por el estado vigente y la señal de entrada, en la que cada elemento especifica un estado siguiente y una acción que ha de ejecutarse para cada combinación de estado vigente y entrada (véase la Figura I.7-16).

State	Signal	
	Signal 1	Signal 2
State 1	State 2 / Action 1	State 3 / Action 3
State 2	State 3 / Action 5	State 1 / Action 2
State 3	State 2 / Action 3	State 1 / Action 2

FIGURA I.7-16/Z.100

### Tabla de transición de estados

Si se tiene un estado vigente y una señal de entrada, es fácil diseñar un programa general para determinar el estado siguiente y la acción. Esta es la noción general que sustenta la realización de las FSM mediante tablas.

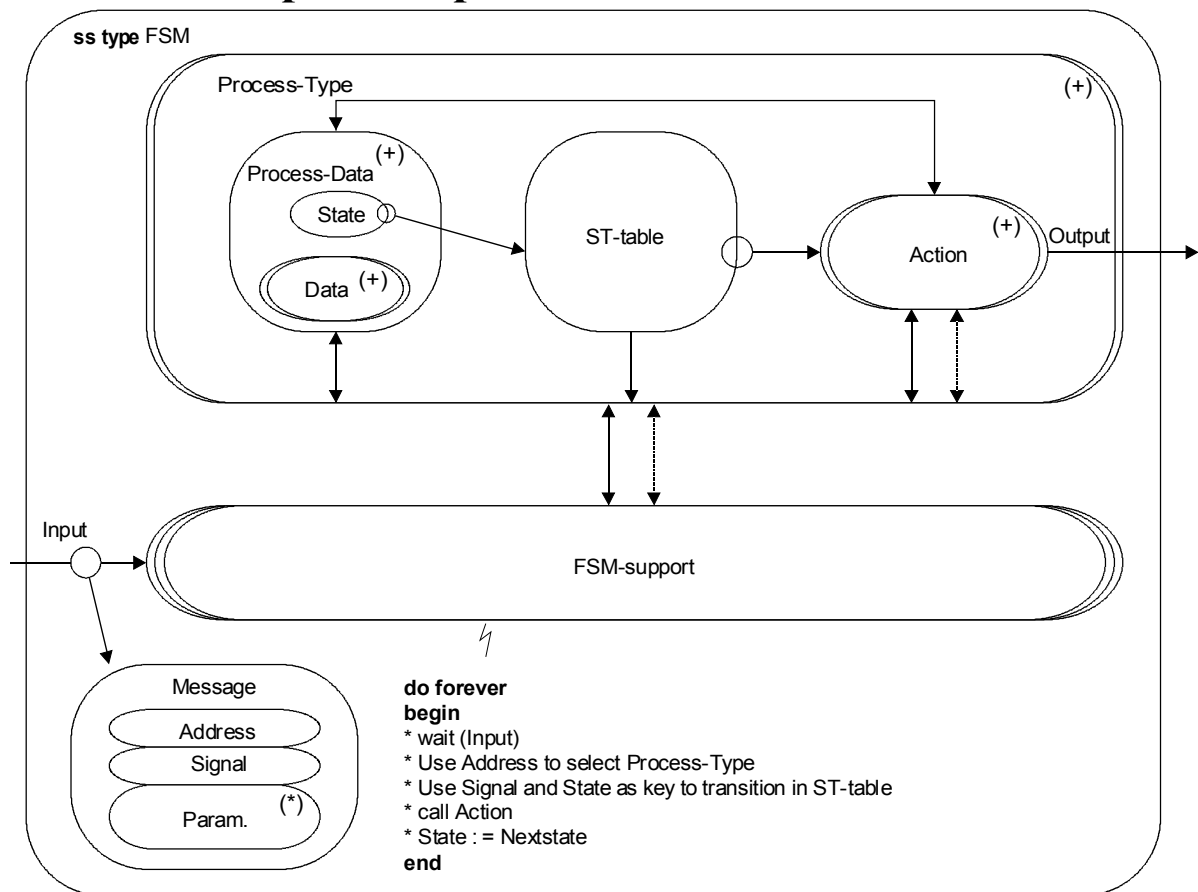
Por lo común, habrá muchas casillas abiertas, pues no se prevé que todas las señales de entrada lleguen a todos los estados, así que es mejor realizar la tabla sin desperdiciar espacio.

En el diagrama de estructura de soporte lógico de la Figura I.7-17 se bosqueja una manera de diseñar la realización de máquinas de estados finitos ampliadas mediante tablas. Hay un programa general, llamado *FSM-support*, que interpreta una estructura de datos *ST-table*, que representa la tabla de transición de estados de la máquina de estados finitos.

El soporte lógico descrito en la Figura I.7-17 proporciona una plataforma de realización concurrente para los procesos descritos como máquinas de estados finitos ampliadas. La organización se efectúa sobre la base de mensajes de entrada. Cada mensaje contiene una dirección, que identifica al proceso receptor. El programa *FSM-Support* utilizará la dirección para activar el proceso, mediante la selección de *Process-Type* y *Process-Data* apropiados. *Process-Data* contiene el estado vigente y los objetos de datos facultativos que puede tener una máquina de estados finitos ampliada además del estado (o sea, la *extension* de la FSM pura). El mensaje de entrada transporta un nombre de tipo de señal, que se emplea para elegir una transición. El proceso direccionado está autorizado a ejecutar una transición para cada mensaje de entrada. Por ello, es una realización casi concurrente, en la que la organización de los procesos está determinada por la dirección del mensaje de entrada.

Resumiendo, la *FSM* aprovecha la circunstancia de que el programa de apoyo puede ser reentrante con respecto a todos los procesos FSM, y que *ST-table* es reentrante con respecto a todos los procesos del mismo tipo. Si hay muchas instancias del mismo tipo de proceso deberá haber un ítem *Process-Data* para cada una, pero *ST-table* y *Actions* pueden compartirse. Si hay muchos tipos de procesos, cada uno debe tener su propia *ST-table* y sus *Actions* correspondientes.

# Reemplazada por una versión más reciente



T1006940-92/d095

Process type      Soporte lógico para un processtype SDL  
 Address            Identificación del proceso receptor (PID)

FIGURA I.7-17/Z.100

## Soporte lógico para máquinas de estados finitos ampliadas

En *ST-table*, cada estado está representado por un registro, que contiene un número (variable) de anotaciones de transición, uno para cada transición del estado (véase la Figura I.7-18). Cada anotación de transición especifica las señales de entrada que pueden causar la transición, la acción correspondiente y el estado siguiente. Véase el ejemplo de la Figura I.7-19.

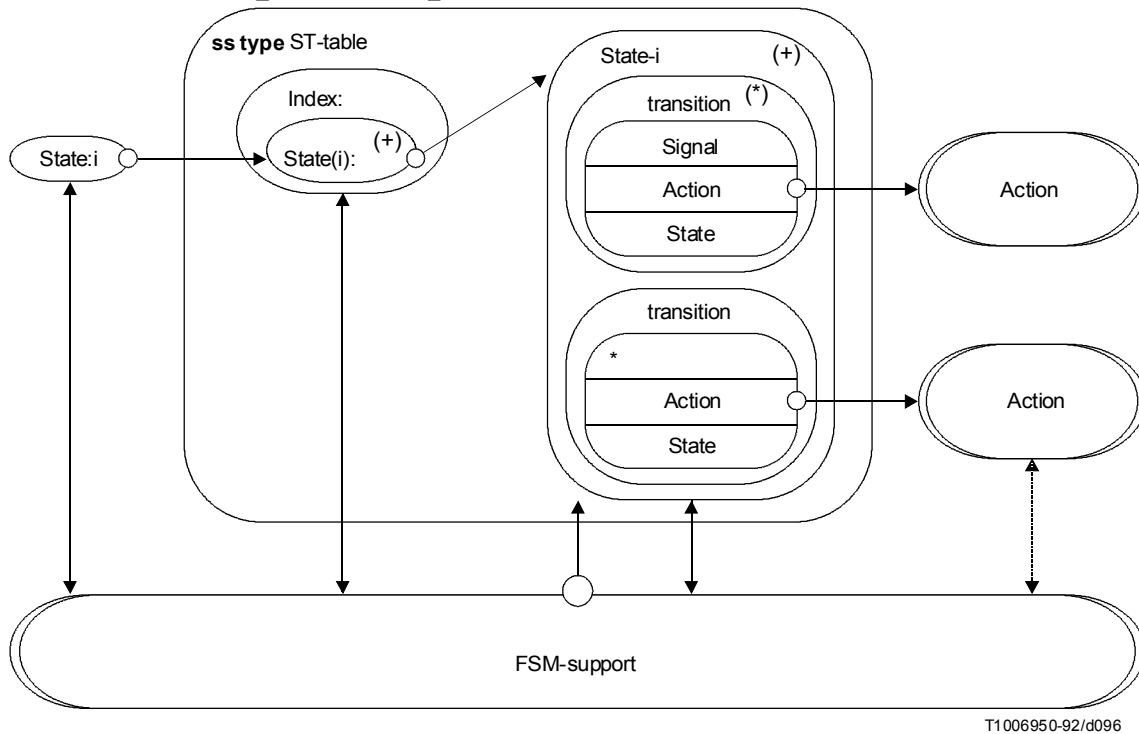
El programa *FSM-support* esperará los mensajes de entrada, y responderá a los mensajes según su orden de llegada. Para cada mensaje, accederá a la anotación de estados de *ST-table* indicado por el estado vigente en *Process-Data* y buscará la anotación de transición que corresponde al mensaje de entrada. Si encuentra una anotación de transición que tiene el mismo nombre de señal que el mensaje de entrada, seleccionará esa transición.

Si no lo encuentra, seleccionará la transición de recepción no especificada, marcada con «\*» (véase la Figura I.7-18). Seguidamente, *FSM-support* ejecutará el procedimiento *Action* indicado en la anotación de transición seleccionada y, por último, asignará el valor de Estado siguiente al Estado vigente. Esto finaliza la transición, y el programa *FSM-support* esperará la llegada de un nuevo mensaje de entrada.

Cuando llega un nuevo mensaje de entrada, la transición siguiente se efectuará de la misma manera.

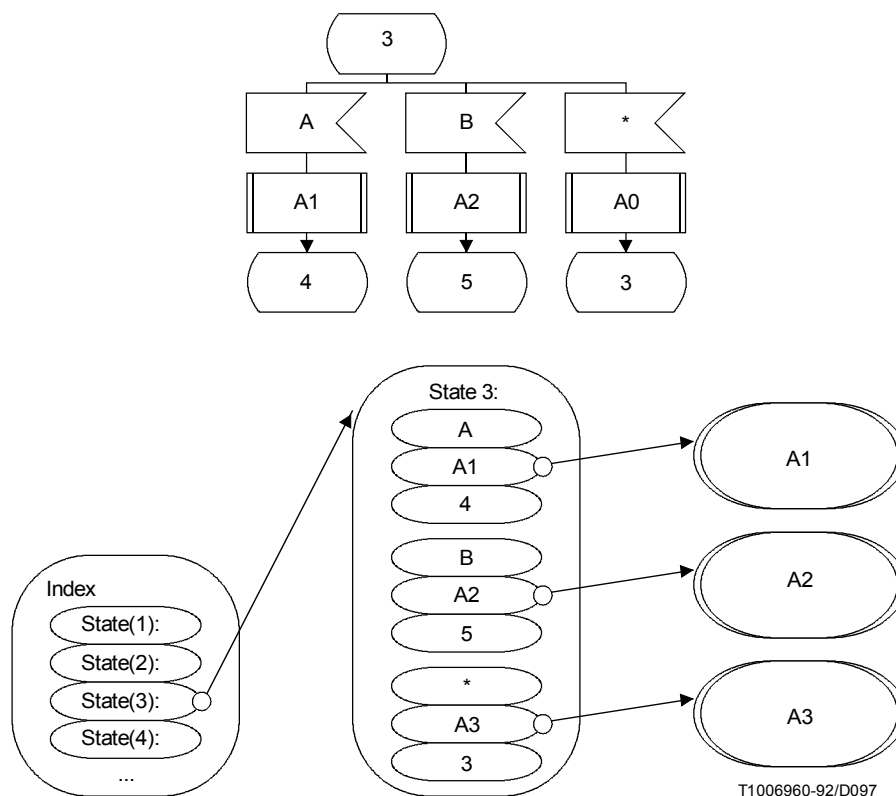
En principio, las acciones se pueden codificar como código interpretado o código directamente ejecutable en el lenguaje de programación. En este trabajo se utilizará el segundo método. Por ello, la interpretación está limitada a la tabla de transición de estados, que representa la parte de control. La parte de acción se ejecuta mediante llamadas al procedimiento de acción indicado en el registro de transiciones.

# Reemplazada por una versión más reciente



T1006950-92/d096

FIGURA I.7-18/Z.100  
Definición de ST-table



T1006960-92/D097

FIGURA I.7-19/Z.100  
Ejemplo de codificación de un estado en ST-table

## Reemplazada por una versión más reciente

En general, durante una transición se efectuará un número de operaciones diferentes, por ejemplo, enviar señales de salida, arrancar y parar temporizadores, asignar valores a objetos de datos. Los procedimientos *Action* ejecutan todas estas operaciones. Por lo común, cada operación será realizada como procedimiento. De allí que el procedimiento *Action* consista, principalmente, en llamadas de procedimiento a procedimientos de operaciones. Además, puede haber asignaciones y expresiones simples que corresponden a las operaciones de datos especificadas, por ejemplo:

```
procedure action-1
begin
  send-signal(open_door, here);
  send-signal(door_bell, here);
  starttimer(now+waiting_time, floor_delay, self);
  table:= cancel_res(table, moving, here);
end;
```

Puede verse que los procedimientos de operaciones pueden diseñarse de forma que constituyan un nivel de lenguaje que se asemeja bastante a la especificación SDL<sup>6)</sup>. Esto simplifica considerablemente el esfuerzo necesario para generar y leer el código de acciones. En general, los procedimientos corresponderán a las operaciones que puede ejecutar una máquina de estados finitos ampliada:

- enviar señales a otros procesos o equipo externo,
- operaciones en datos locales, y
- fijación y reiniciación de temporizadores.

Hay que normalizar el formato de las señales para cada realización específica. No basta con que las interfaces de señal sean correctas en el nivel de diseño funcional; deben ser compatibles también en el nivel concreto. Esto significa que los procesos que comunican deben codificar las señales de la misma manera.

Se puede dar prioridad a los mensajes cursados entre dos procesos que funcionan en la misma instancia de *FSM-support* con respecto a los mensajes externos enviando esos mensajes a través de una cola interna. Esto acelerará las comunicaciones internas, pues se evita la sincronización externa y las llamadas al sistema operativo (operaciones **wait**, **send**). Los mensajes dirigidos a procesos externos se pasan a semáforos externos de comunicación, quizás mediante un procedimiento de encaminamiento.

La técnica de tabla tiene varias ventajas:

- a) **Facilita la realización y la documentación** – La codificación de las estructuras de control, normalmente difícil y expuesta a errores, se efectúa fácil y fiablemente con *ST-table*. En general, las acciones de transición se estructuran en procedimientos pequeños y bien definidos. Un tamaño típico es 10 renglones de código, en su mayor parte llamadas a rutinas comunes de envío de señales, operaciones de datos y temporización. Es fácil ver en las *ST-table* la relación entre el diseño funcional y su realización de soporte lógico. Tan fácil es, que hasta quien no es programador puede hacer la codificación. Se puede preparar un manual de «recetas» al alcance de todos.

El nombre del procedimiento *Action* puede introducirse como comentario en el gráfico de procesos SDL para facilitar la referencia de la especificación SDL al código real.

Si el código está razonablemente bien estructurado y comentado, el gráfico de procesos SDL y la lista de códigos fuente bastan como documentación. Normalmente, los usuarios emplean los diagramas SDL como documentación principal, y sólo utilizan los códigos cuando hay que suprimir errores. Así, esta técnica estimula a los usuarios a mantener actualizado el diseño funcional.

- b) **Mejora la modularidad y facilita el mantenimiento** – Se puede utilizar el programa *FSM-support* en muchos sistemas con aplicaciones diferentes. Por ello, es un módulo muy reutilizable. También resulta más fácil reutilizar los procesos en nuevas aplicaciones, cuando éstas se realizan de manera normalizada.

---

<sup>6)</sup> Esto no es exclusivo del método por tablas. Las realizaciones mediante codificación directa pueden emplear un método semejante al código de acción.



## Reemplazada por una versión más reciente

Además, las modificaciones son fáciles, y se puede controlar sus consecuencias. La práctica común es modificar primero el gráfico de proceso (*ST-table*), y luego añadir los procedimientos *Action* necesarios. La cuestión consiste en evitar los efectos secundarios cuando se llama a un procedimiento *Action* desde varias transiciones. Esto se puede conseguir manteniendo una referencia «hacia atrás» de los procedimientos *Action* a las transiciones, o utilizando un procedimiento *Action* separado para cada transición.

- c) **Es fiable** – El programa *FMS-support* se probará minuciosamente en muchas aplicaciones, y puede ser muy fiable. También es probable que la aplicación sea fiable, puesto que se ha derivado directamente de un diseño funcional supuestamente correcto.
- d) **Es robusta** – Su robustez obedece, por una parte, al mecanismo de comunicación de mensajes, y por otra, a la técnica de tabla. Cuando las señales son pasadas como mensajes, el receptor siempre puede verificar la entrada antes de utilizarla. Así, el *FSM-support* puede verificar la coherencia de los mensajes entrantes antes de aceptarlos. Además, la transición «\*» de *ST-tables* garantiza que todas las entradas inesperadas puedan ser recibidas y tratadas adecuadamente.
- e) **Admite pruebas** – *FSM-support* se puede realizar en una versión de prueba, que permite al probador simular los mensajes de entrada, y seguir las *Actions* de transición y los mensajes de salida generados. Así, un proceso puede ser sometido a prueba convenientemente en un entorno que, para el proceso, se asemeja exactamente al real.

Cabe reconocer que el programa *FSM-support* introduce tara. La búsqueda de registros de transición puede tomar algún tiempo. Para reducir esta tara, se deben organizar los registros en el orden de las señales más frecuentes.

Si se tienen en cuenta todos los aspectos de una aplicación, sucede con frecuencia que la tara de tiempo es despreciable. De hecho, cuando varios procesos se realizan con el mismo *FSM-support*, los cambios de contexto almacenados en el sistema operativo pueden ser ventajosos.

*FSM-support* es un candidato bien definido para la optimización de velocidad, si procede. La tabla también se puede codificar como una matriz bidimensional, en la que el estado y la señal de entrada se utilizan como índices. Esta técnica proporcionará un acceso rápido, pero utilizará demasiado espacio.

Dado que *ST-table* codifica las estructuras de control con poco espacio, se economizará espacio en aplicaciones grandes.

¿Cuáles son las limitaciones de este método? Puesto que otros procesos quedan bloqueados durante una transición, *Actions* no esperarán los procesos externos ni ejecutará operaciones que consuman mucho tiempo. Por las mismas razones, los procesos realizados en la misma instancia de *FSM-support* tendrán la misma prioridad. (No pueden interrumpirse mutuamente.)

### Realización de las características del SDL

El SDL tiene algunas características que difieren de las de las máquinas de estados finitos comunes, por ejemplo, decisiones, conservación, procedimientos, servicios y creación dinámica de procesos. Todas se pueden realizar por encima de las realizaciones FSM descritas hasta el momento.

En la especificación SDL, las decisiones pueden codificarse como estados cuando se utiliza la técnica de apoyo de FSM. Esto no es estrictamente necesario, pero hace que los procedimientos *Action* sean independientes de estados particulares.

Cuando una decisión afecta al estado siguiente se debe tratar como si fuera un estado. La operación de pregunta debe efectuarse antes de pasar a un estado de decisión, y enviar la respuesta como una señal de entrada, que se recibirá en el estado de decisión. De esta manera, las bifurcaciones en valores internos y las señales externas se tratarán igual. Para acelerar la toma de decisiones, y finalizar las decisiones antes de que lleguen nuevas entradas, se tratarán las señales de decisión internamente en *FSM-support*, es decir, se les dará prioridad con respecto a las señales externas. Cuando se haya emitido una decisión, la transición correspondiente se efectuará inmediatamente. Cuando una transición contiene una decisión que no afecta al estado siguiente, se codificará como una decisión interna dentro del procedimiento de *Action* de transición.

## Reemplazada por una versión más reciente

El mecanismo de conservación entraña que cada proceso tiene una cola lógica de señales almacenadas. Esta cola se puede realizar como parte de los datos de proceso. Así, un símbolo de conservación se puede codificar como una transición al estado vigente (estado siguiente := estado vigente), y la acción consiste en poner la señal en la cola de conservación. Cuando un proceso efectúa una transición a un estado nuevo, las señales de la cola de conservación se trasladan a la cola de entrada, y se tratan como señales de entrada normales. Por consiguiente, la conservación se puede realizar añadiendo una cola de conservación a los datos de cada proceso, y trasladando las entradas de esta cola a la cola normal de entradas, al final de cada transición a un nuevo estado. En vista de que esto puede necesitar bastante tiempo, se evitará la conservación en las aplicaciones críticas con respecto al tiempo.

Lo que se trata de hacer con una cola de conservación es mantener alejadas algunas señales hasta que llegue el momento de tratarlas. Esto también se puede hacer de otras maneras, por ejemplo, mediante colas de señal separadas. Antes de la realización, habrá que estudiar bien para qué se utilizará la conservación, y buscar soluciones que eviten actividades que consumen demasiado tiempo.

En el SDL, los procedimientos pueden contener estados. En ese caso, el procedimiento introduce una estructura de nivel en los gráficos de proceso. Esto se realiza fácilmente mediante la técnica de codificación directa, cuando los procedimientos SDL se hacen corresponder directamente con los procedimientos del lenguaje de programación.

Sin embargo, esto no es posible cuando se hacen corresponder varias instancias de proceso SDL con una instancia de soporte lógico. En este caso, cada proceso necesitará una pila, en la que almacenará estados, direcciones de tareas y datos locales con respecto a los procedimientos. Cuando se llama a un procedimiento, el contexto vigente (estado, dirección de tarea y datos) se introduce en la pila, y se pasa al procedimiento. De retorno, se efectúa la operación contraria. Esto se realiza con más facilidad si las llamadas de procedimiento se hacen siempre al final de una transición, porque se retornará a un estado y no a una dirección de tarea arbitraria.

En el SDL, los servicios son, básicamente, máquinas de estados finitos que funcionan casi en paralelo, y utilizan señales de prioridad para comunicar entre sí. Se realizan fácilmente dentro de un proceso de soporte lógico mediante técnicas de tablas o de codificación directa. Para identificar el servicio que recibirá una señal de entrada dada, quizás sea necesario considerar el nombre de tipo de señal, además del PID.

Cuando hay una instancia de proceso de soporte lógico por cada instancia de proceso SDL, la creación dinámica de procesos exige el apoyo del sistema operativo. En un proceso de soporte lógico, la creación dinámica de procesos consiste en asignar e inicializar un registro de la memoria libre, que pueda almacenar los datos de proceso.

### I.7.5.5 Datos

Hemos llegado al problema trascendental que plantea la realización de los datos SDL. Los datos SDL tienen tres aspectos:

- los géneros, definidos en términos de operadores y ecuaciones,
- la instanciación de los géneros como variables de proceso SDL, y
- la instanciación de los operadores especificados en las expresiones de transición de procesos SDL.

En las especificaciones SDL, los géneros ayudan a centrarse en la funcionalidad y a suprimir los detalles que no son pertinentes para la realización. En la realización del soporte lógico, la noción de tipo de datos abstractos ayuda a incrementar la modularidad y la reutilización. En este nivel, el aspecto principal es la *encapsulación* y la *ocultación de información* [18].

Se trata de encapsular datos en módulos, que sólo proporcionan al entorno un conjunto de operadores bien definidos. Esto significa que no se puede acceder a las estructuras de datos directamente, sino que hay que invocar operadores que, con frecuencia, se realizan como procedimientos. De esta manera, las estructuras internas de datos son encapsuladas por los procedimientos de operador, y ocultadas al entorno (véase la Figura I.7-20).

Por consiguiente, la instanciación de los operadores especificados en los gráficos de proceso SDL se realizará mediante llamadas a los procedimientos de operador. Esto significa que las *Actions* de transiciones son independientes de la estructura de datos específica utilizada para los tipos de datos.

Aun si los datos están definidos informalmente en el nivel SDL, la noción de tipo de datos abstractos es útil porque ayuda a conseguir independencia y modularidad.

# Reemplazada por una versión más reciente

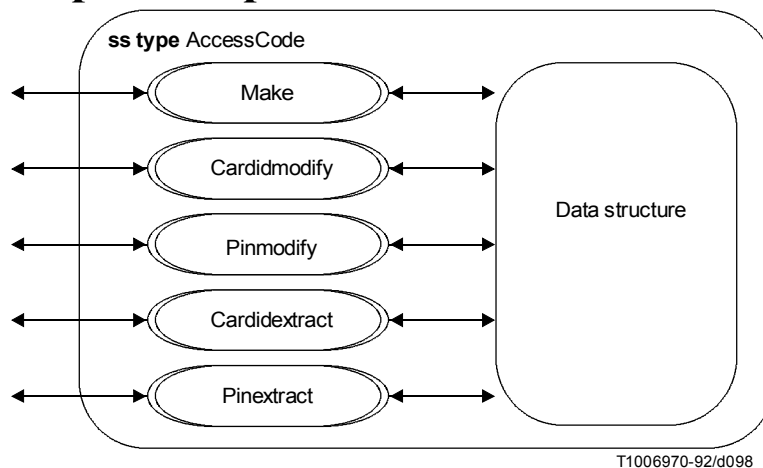


FIGURA I.7-20/Z.100

## Realización de un género es un módulo encapsulado

El primer paso del diseño de soporte lógico de un género consiste en especificar su signatura (la interfaz de operador). Analicemos, como ejemplo, el género SDL *AccessCode*:

```
newtype AccessCode
operators
  Make           : Integer, Integer      ->   AccessCode;
  cardidModify  : AccessCode, Integer   ->   AccessCode;
  pinModify     : AccessCode, Integer   ->   AccessCode;
  cardidExtract : AccessCode           ->   Integer;
  pinExtract    : AccessCode           ->   Integer;
axioms
/* we have omitted them here */
endnewtype AccessCode;
```

La interfaz de operador correspondiente, en términos de los encabezamientos de definición de procedimiento, puede ser:

```
Make(integer, integer)
Cardidmodify(accesscode, integer)
Pinmodify(accesscode, integer)
Cardidextract(accesscode)
Pinextract(accesscode)
```

El paso siguiente consiste en elegir una estructura de datos apropiada. La elección es muy importante, puesto que influirá decisivamente en los algoritmos utilizados para realizar los operadores y, por tanto, el comportamiento. La elección dependerá de las restricciones no funcionales del diseño. Podrá haber varias realizaciones que correspondan a diferentes restricciones de diseño, y todas se conseguirán mediante interfaces de operador idénticos.

En la especificación SDL formal, se puede definir el significado de los operadores mediante ecuaciones o especificaciones de operador. Sin embargo, no se ha encontrado todavía una manera general y directa de convertir una definición axiomática en una realización eficaz. Por consiguiente, habrá que ser pragmático en lo que respecta al diseño de estructuras de datos y algoritmos. Un caso mucho más sencillo consiste en utilizar especificaciones de operador, puesto que una especificación de operador es semejante a un procedimiento de devolución o retorno de valores (de hecho, se define por una transformación en ese tipo de procedimiento).

En primer lugar, hay que analizar varias estructuras de datos posibles, y seleccionar una que cumpla los requisitos de comportamiento, por ejemplo, una lista enlazada. A continuación, hay que diseñar un procedimiento que corresponda a cada operador.

En este ejemplo se ha aplicado un diseño que consiste en definir un procedimiento para cada operador del género. Otra posibilidad es utilizar una macro, o incluso un enunciado simple, si el género está apoyado directamente en el lenguaje de programación. Este será el caso de muchos de los géneros predefinidos del SDL. En los sistemas en tiempo real, muchas estructuras de datos y algoritmos son sencillos, por lo que se obtendrán ventajas considerables utilizando los géneros predefinidos.

## Reemplazada por una versión más reciente

En los casos complejos, hará falta esforzarse más en esta etapa del diseño. Cuando los géneros tienen estructuras de datos complejas, es útil representar la estructura de datos lógica mediante descripciones de datos conceptuales, por ejemplo, descripciones de relación entre entidades, como parte de los requisitos funcionales o del diseño funcional.

El paso siguiente consistiría en hacer corresponder las descripciones funcionales con datos SDL. Puesto que la interfaz de operador de los objetos de datos oculta la estructura interna de los datos al entorno, se puede utilizar un sistema de gestión de base de datos sin que el entorno lo sepa. De hecho, la noción de tipos de datos abstractos puede aplicarse con éxito a estructuras de datos muy elaboradas.

Los tipos de datos abstractos están apoyados en el SDL, pero la noción no es específica del SDL. En realidad, estas directrices se pueden utilizar en el diseño de cualquier soporte lógico, independientemente del método de especificación. Ayudan a descomponer un diseño en módulos con bajo acoplamiento y alta cohesión.

### I.7.5.6 Diseño general de la arquitectura del soporte lógico

En las subcláusulas anteriores se han analizado los problemas y soluciones de diseño más importantes. En ésta, se estudiará cómo se pueden utilizar en un enfoque general de la arquitectura del soporte lógico. El objetivo es encontrar y documentar una estructura de soporte lógico que permita realizar el sistema SDL y cumpla los requisitos no funcionales. La información que se precisa para el diseño de la arquitectura del soporte lógico es

- la especificación SDL;
- el diseño de la arquitectura del soporte físico; y
- los requisitos no funcionales.

Por la especificación SDL se conocen las interfaces funcionales y la funcionalidad del sistema de soporte lógico. El diseño de la arquitectura del soporte físico proporciona información sobre las interfaces físicas, y las restricciones de comportamiento y temporización vienen indicadas en los requisitos no funcionales.

El primer paso consiste en analizar la comunicación interna y la externa, y diseñar una solución para cada interfaz.

Por una parte, están los problemas de las interfaces físicas y, por otra, las necesidades de comunicación interna. Hay que comenzar por el estudio de las interfaces entrada/salida y sus requisitos relativos a tiempos de respuesta, medición de tiempo, detección de eventos, prioridades y sincronización. Luego hay que considerar las exigencias de comunicación interna de los procesos SDL. ¿El tiempo es crítico? ¿Se pueden utilizar llamadas de procedimiento? ¿Debe utilizarse comunicación con memoria tampón?

En esta etapa, se puede tomar una decisión sobre la necesidad de utilizar o no un sistema operativo.

El paso siguiente consiste en asociar los módulos de interfaz con los módulos de procesos internos. Esto resultará en una estructura general de soporte lógico como la que se muestra en la Figura I.7-4 o la Figura I.7-10.

La mejor manera de realizar la máquina de estados finitos ampliada definida con el SDL dependerá de las limitaciones de diseño. Cuando lo permitan las restricciones de velocidad, se utilizará la realización con técnica de tablas. Esta es una solución compacta, flexible y fiable. Si la velocidad de procesamiento es crítica, quizás haya que utilizar la realización con codificación directa, y realizar las señales mediante procedimientos.

La parte de datos se realizará en módulos que corresponden a géneros.

Se puede realizar cada proceso SDL en un proceso de soporte lógico, pero esto sólo hace falta cuando se necesita la prioridad con apropiación. Otra solución puede ser que varios procesos SDL funcionen por encima de un solo proceso de soporte lógico, lo que se puede conseguir, por ejemplo, con la técnica de tablas descrita en I.7.5.4.

Volvamos al tiempo de ejecución medio estimado, analizado en I.7.4.3. Ahora se pueden mejorar las estimaciones. Calcúlese otra vez la carga máxima media del computador, y si el resultado es superior al límite de carga, debe reconsiderarse otra vez el diseño y elegir entre:

- asignar los procesos SDL diferentemente a los computadores;
- utilizar un computador más veloz; u
- optimizar el soporte lógico para utilizar una mayor velocidad de ejecución.

Si los valores de carga son correctos, se deben verificar las restricciones de tiempo real en función de los tiempos de ejecución máximos exigidos para responder. Si no surgen problemas, debe proseguirse el diseño de cada uno de los procesos SDL y la elaboración de apoyo de soporte lógico.

# Reemplazada por una versión más reciente

Los procesos de soporte lógico que interactúan mediante la comunicación con memorias también proporcionan una interfaz de mensaje a su entorno y ocultan la estructura interna. Por consiguiente, son módulos relativamente independientes, de fácil manejo e integración.

## I.7.6 Diseño de la arquitectura del soporte físico

El diseño de la arquitectura del soporte físico se ha analizado en I.7.4. En contraste con el soporte lógico, el soporte físico tiene diferencias fundamentales con respecto al SDL. Este soporte es físico y, por tanto, genera errores con el correr del tiempo, precisa tiempo para ejecutar las tareas y está expuesto al ruido.

Entre el SDL y el soporte físico existen semejanzas conceptuales. Por ejemplo, la concurrencia es natural en el soporte físico, y durante mucho tiempo se ha especificado y realizado el soporte físico mediante máquinas de estados finitos.

Existen lenguajes de descripción de soporte físico, como el VHDL, que se parecen a los lenguajes de programación. Por ello, y hasta cierto punto, el diseño de la arquitectura del soporte físico es semejante al diseño de la arquitectura del soporte lógico, pues se trata de que la especificación SDL concuerde con una descripción correspondiente en el lenguaje de descripción del soporte físico. Esto se puede lograr con herramientas de traducción automática, igual que para el soporte lógico, pero el diseñador de soporte físico se enfrenta con un conjunto diferente de limitaciones, por lo que su tarea es diferente de la del diseño del soporte lógico.

### Concurrencia y tiempo

En el nivel de circuito, los componentes del soporte físico funcionarán, con frecuencia, sincronamente, e interactuarán sincronamente mediante valores continuos. Estos mecanismos se deben utilizar para realizar el funcionamiento asíncrono y la interacción asíncrona con secuencias de valores existentes en el sistema SDL.

Esto significa que habrá que limitar el sistema SDL de manera que se ajuste a los mecanismos del soporte físico, o bien el soporte físico deberá proporcionar los mecanismos utilizados en el SDL.

En principio, la mayor parte de los mecanismos del SDL se pueden realizar en soporte físico. Se pueden construir componentes asíncronos correspondientes a procesos SDL que comunican asíncronamente a través de canales de tipo SDL.

Sin embargo, por razones técnicas y económicas, normalmente no será práctico realizar sistemas SDL generales en soporte físico. Por tanto, es más común restringir la utilización del SDL para adecuarla a la realización del soporte físico. Esto quizás signifique limitar la estructura de comunicación para utilizar la comunicación síncrona sin conservación o almacenamiento.

En un sistema síncrono, la señal de reloj proporciona una referencia temporal natural. Las mediciones de tiempo pueden efectuarse contando el tictac del reloj.

### Comportamiento secuencial

La realización de la máquina de estados finitos en el soporte lógico no plantea problemas importantes. De hecho, las máquinas de estados finitos se utilizaron en el diseño del soporte físico antes que en el diseño del soporte lógico.

Siempre y cuando la utilización de los datos se limite a los tipos de datos apoyados en el lenguaje de descripción del soporte físico, la realización de las tareas y decisiones no plantea problemas importantes.

Los procedimientos pueden causar problemas, especialmente si se necesita asignación dinámica de datos para sustentarlos. Por consiguiente, se debe restringir la utilización de procedimientos.

En el contexto del soporte físico, la creación dinámica de procesos no es factible, si significa crear una unidad física, pero se puede simular si, desde el comienzo, se incorporan procesos de activación y de pasivización en el soporte físico.

## I.7.7 Directrices para el diseño de la realización por pasos

Los pasos más importantes del diseño de la realización son:

- Paso 1: Soluciones de transacciones entre soporte físico y soporte lógico
- Paso 2: Diseño de la arquitectura del soporte físico
- Paso 3: Diseño de la arquitectura del soporte lógico
- Paso 4: Reestructuración y refinamiento de la especificación SDL
- Paso 5: Diseño detallado del soporte físico y del soporte lógico.

# Reemplazada por una versión más reciente

Téngase presente que éstos son únicamente los pasos principales del diseño. En la práctica, quizás se necesiten iteraciones. Si bien estos pasos se basan en decisiones de alto nivel que posiblemente seguirán siendo manuales, el diseño detallado y la realización se podrán automatizar gradualmente.

Colocados en la perspectiva de la comunicación. Primero han de hallarse soluciones para las interfaces externas y, después, para las internas. Se pasa al diseño de realización de procesos. Debe utilizarse el principio de generalidad, prefiriéndose las técnicas más generales y flexibles, siempre que sea posible.

## Paso 1 – Transacciones entre soporte físico y soporte lógico

- Analizar las necesidades de distribución física de las interfaces y servicios. Seleccionar una estructura física de sistema que sustente esto. Minimizar la anchura de banda de los canales que cubren distancias físicas.
- Examinar los requisitos de comportamiento. Calcular los valores medios de carga de procesamiento para cada canal, ruta de señal y proceso SDL. Asignar procesos a computadores, de forma que la carga media de un computador no rebase 0,3 Erlangs de su capacidad total.
- Considerar los requisitos de tiempo real. Calcular los tiempos de respuesta de las funciones críticas con respecto al tiempo, y verificar que se cumplen esos requisitos. Utilizar la prioridad para garantizar respuestas rápidas. Aislar, en la mayor medida posible, las partes críticas con respecto al tiempo.
- Analizar los requisitos de fiabilidad. Considerar la necesidad de redundancia. Añadir unidades redundantes y reestructurar el sistema hasta que se cumplan esos requisitos.
- Considerar los requisitos de protección y seguridad.
- Analizar la producción en función de la relación costo-eficacia.
- Estudiar procedimientos rentables para las modificaciones.

## Paso 2 – Arquitectura del soporte físico

- Describir la estructura general de los computadores y de otras unidades de soporte físico.
- Describir las conexiones físicas entre las unidades de soporte físico.
- Describir los formatos de señal, esquemas de sincronización y protocolos que se utilizarán en las conexiones físicas que no estén descritos ya en la especificación de requisitos o el diseño funcional.

## Paso 3 – Arquitectura del soporte lógico

- Establecer, lo más directa y fiablemente posible, la correspondencia entre la especificación SDL y la realización de soporte lógico.
- Buscar las interfaces físicas y los módulos de diseño de soporte lógico que puedan encargarse de la capa física, por ejemplo, sincronización, detección de eventos, temporización, conversión de formatos.
- Estudiar las interfaces internas y elegir un mecanismo de comunicación adecuado para cada uno, por ejemplo, llamadas de procedimiento, memorias tampón de mensajes, valores continuos.
- Utilizar el esquema de comunicación más general y flexible para las señales SDL, o sea, comunicación con memoria tampón, a menos que se esté seguro de que se necesitan llamadas de procedimiento y que éstas funcionarán.
- Dar preferencia a un sistema operativo general que admita procesos concurrentes y la comunicación con memoria tampón, salvo cuando sea evidente que basta con una estructura de programa secuencial simple.
- Seleccionar un método de realización para cada proceso SDL. Siempre que sea posible, utilizar sistemas generales de apoyo para facilitar la realización de las funciones de aplicación y aumentar la fiabilidad.

## Paso 4 – Reestructuración y refinamiento de la especificación SDL

- Reestructurar y refinar la especificación SDL de alto nivel en una especificación SDL de bajo nivel, en caso necesario, para plasmar la estructura del sistema concreto.

Todas las propiedades funcionales de la realización deben reflejarse en la especificación SDL de bajo nivel. Esto sirve para garantizar la equivalencia.

# Reemplazada por una versión más reciente

La especificación SDL debe reflejar la estructura del sistema concreto. Esto sirve para simplificar la correspondencia entre la especificación SDL y la realización.

Por lo común, se necesitarán otras funciones para apoyar el sistema concreto. Estas funciones dependerán de las decisiones de diseño, y no se pueden definir antes de elaborar todo el diseño. Algunas funciones serán visibles para los usuarios y otras invisibles.

Entre las funciones normalmente visibles para los usuarios se encuentran:

- tratamiento de errores, por ejemplo, informes de errores, servicios no disponibles;
- operación y mantenimiento de la realización, por ejemplo, unidades de bloqueo, unidades de prueba;
- acceso a recursos limitados, tales como impresoras.

Algunas funciones que serán invisibles para los usuarios son:

- multiplexación de computadores y canales;
- sincronización y exclusión mutua;
- servicios de comunicación;
- control de carga.

Téngase en cuenta que la reestructuración no significa que haya que redefinir todo. Es posible que no se modifique la mayor parte de los procesos de la especificación SDL de alto nivel. Si están definidos como tipos separados, resulta fácil poner las instancias con algunos procesos nuevos en un contexto estructural nuevo.

## Paso 5 – Diseño detallado del soporte físico y del soporte lógico

- Diseñar completamente las unidades de soporte físico y soporte lógico identificadas en los pasos anteriores y utilizar, en la mayor medida posible, técnicas y componentes normalizados.

## I.8 Métodos formales de validación, verificación y prueba

### I.8.1 Introducción

A continuación se ofrece una introducción a los métodos formales de validación, verificación y prueba. Es una cuestión amplia, objeto de intensas actividades de investigación. Se espera poder ofrecer directrices más detalladas en un futuro próximo.

Según la terminología comúnmente aceptada, la *validación* tiene que ver, por una parte, con el establecimiento de una correspondencia entre los requisitos informales y, por otra, con la especificación y/o realización de un sistema. Dicho de otra manera, el objetivo de la validación es responder a la pregunta: «¿Es el sistema que se había previsto, y satisfará las expectativas de sus usuarios futuros?».

La *verificación* se relaciona con el establecimiento de la *corrección*, según ciertos criterios de corrección, de la especificación y/o realización de un sistema. En otras palabras, el objetivo de la verificación es responder a la pregunta «¿Es correcto el sistema?». Va de suyo que una de las expectativas de los usuarios futuros es que el sistema sea correcto, por lo que la verificación puede considerarse parte de la validación.

La *prueba* trata del establecimiento de la *conformidad*, según ciertos criterios de conformidad, de una realización con respecto a su especificación asociada. En general, los criterios de conformidad se expresan en normas, mediante enunciados de conformidad en los que se estipula, entre otras cosas, un conjunto mínimo de características que deberán tener las realizaciones conformes. La prueba puede considerarse una técnica que se utilizará en la verificación y la validación.

En general, las propiedades que se necesitan en la validación/verificación se clasifican en propiedades de *seguridad* y propiedades de *agilidad*. Hay que satisfacer las propiedades de seguridad en cada instante de tiempo en el sistema especificado, por ejemplo, ausencia de atascos. Hay que satisfacer las propiedades de agilidad en algunos instantes de tiempo, por ejemplo, «después de la petición de conexión, el sistema deberá responder con una indicación de conexión o una indicación de desconexión».

Dado que el SDL, al igual que otras técnicas formales de descripción normalizadas (FDT, *formal description technique*), se ocupa principalmente del funcionamiento de los sistemas, esta sección se centra en la validación y verificación de las propiedades funcionales, o sea, el comportamiento especificado en términos de comunicación de señales. No se considera la validación y verificación de las propiedades no funcionales, por ejemplo, requisitos de tiempo real, criterios de calidad de funcionamiento, etc. Además, se restringe el alcance a los casos de prueba y no se analizan los procedimientos de prueba.

## Reemplazada por una versión más reciente

Una de las ventajas de utilizar un lenguaje de especificación formal, como el SDL, consiste en que se puede efectuar una validación más precisa y una verificación formal de las especificaciones. La especificación también se puede utilizar para elaborar casos de prueba para la prueba de las realizaciones de la especificación. Esta es una consecuencia directa de la semántica formal del SDL, puesto que la semántica define una interpretación que no es ambigua y permite el razonamiento formal acerca de las propiedades de comportamiento de las especificaciones.

El modelo de semántica formal definido para el SDL es un modelo *denotacional*, que define el comportamiento de un sistema SDL en términos de una máquina abstracta, que interpreta la especificación SDL. En este modelo, las propiedades de comportamiento de la especificación, importantes para la validación y la verificación, no están expresadas explícitamente; en cambio, se pone el acento en el comportamiento de la máquina abstracta que interpreta la especificación. Esto hace que el modelo semántico actual del SDL no sea directamente aplicable en el contexto de la validación y la verificación.

Otra posibilidad consiste en utilizar un modelo *operacional*, véase [19], que define el comportamiento del sistema en términos del comportamiento observable, mediante una notación de *Sistemas de transiciones etiquetadas (labelled transition systems)*. En [20] y [21] se ofrece un modelo que sustenta este punto de vista, en el que el comportamiento de un sistema SDL se define con respecto a dónde se observan los eventos. Por ejemplo, los eventos considerados observables en el entorno del sistema son la entrada y salida de las señales en y desde el sistema, los eventos considerados observables en el nivel de bloque son la entrada y salida de las señales en y desde el bloque observado, etc. Una de las ventajas más importantes de este modelo es que permite elaborar herramientas informatizadas, que apoyan la validación y verificación de las especificaciones. Usualmente, esas herramientas se basan en la técnica de *exploración de estados*, véase [22], en la que el comportamiento del sistema se calcula y representa en términos de un *gráfico de alcanzabilidad* o un *Arbol de Comunicación Asíncrona*, véase [23]. Informalmente, el proceso de cálculo del gráfico de alcanzabilidad de un sistema se puede describir así:

*Dado un estado inicial del sistema, se ejecutan todos los eventos posibles que pueden ocurrir en ese estado, lo que resulta en un conjunto nuevo de estados del sistema. Los estados están conectados al estado inicial mediante un borde etiquetado, que identifica el evento ejecutado. Este proceso se repite para cada estado, hasta que no se pueda derivar ningún estado nuevo.*

Sin embargo, a todas las herramientas basadas en esta técnica se les plantea el problema de la *explosión de estados*. El número de estados del gráfico de alcanzabilidad rebasa la capacidad del sistema que efectúa el cálculo del comportamiento. El problema de la explosión de estados es una característica inherente del comportamiento intrincado de los sistemas de comunicación más que una deficiencia de una técnica determinada. Por ejemplo, un sistema SDL tiene, en general, un comportamiento infinito, lo que hace que no se pueda calcular el comportamiento completo del sistema. No obstante, se dispone de varias técnicas diferentes para mitigar los efectos de la explosión de estados:

- Técnicas que *reducen* el número de estados necesarios que han de generarse
  - 1) la *ordenación parcial* de los eventos en vez del entrelazado, véase por ejemplo [24];
  - 2) la *abstracción*, agrupando estados semánticamente equivalentes del gráfico de alcanzabilidad en un solo estado, véanse por ejemplo [25] y [28];
  - 3) el *análisis parcial*, generando sólo una parte del comportamiento posible, véase, por ejemplo [26];
  - 4) la *partición*, dividiendo el sistema en subsistemas independientes, que se analizan por separado, véase por ejemplo [27].
- Técnicas de *funcionamiento*, destinadas al almacenamiento eficaz del gráfico de alcanzabilidad, véase por ejemplo [26].

Estas técnicas diferentes se pueden combinar y utilizar en una herramienta.

### I.8.2 Validación y verificación

#### Paso 1 – Análisis

- Asegurarse de que la especificación es sintáctica y semánticamente correcta, mediante herramientas clásicas de análisis SDL.

#### Paso 2 – Simulación

- Validar/verificar el comportamiento normal de la especificación con una herramienta de simulación.



## Reemplazada por una versión más reciente

El propósito de este paso es, sobre todo, comprobar que el comportamiento normal descrito en la especificación de requisito es correctamente tratado por la especificación SDL. Esto se logra aplicando la herramienta de simulación conforme a la «utilización normal» prescrita, por ejemplo, la descrita en los gráficos de secuencias de mensajes, al tiempo que se comprueba si es coherente con la especificación SDL. La simulación pondrá al descubierto las principales deficiencias del diseño (si es que hay alguna) y también, como efecto secundario, ayudará a detectar algunos errores no detectados en el **paso 1**.

Las herramientas de simulación permiten ejecutar una especificación SDL mientras se investiga interactivamente el comportamiento del sistema SDL especificado. Se pueden enviar señales al sistema y verificar las respuestas del sistema. Por lo común, también se puede forzar al sistema a pasar a algún estado predefinido, por ejemplo, cambiando valores de variables o creando instancias de proceso, y proseguir la simulación a partir de ese estado.

En el mercado hay herramientas de simulación para el SDL de diversas marcas, que deben considerarse como una técnica bien conocida para la validación y verificación de las especificaciones SDL.

### Paso 3 – Exploración del espacio del estado

- Comprobar ciertas propiedades específicas de la especificación con una herramienta de exploración del espacio del estado.

El objetivo de este paso es comprobar que una especificación de sistema SDL determinada tiene ciertas propiedades específicas. Puede tratarse de propiedades generales, como la ausencia de atascos u otras situaciones generales de error, o de propiedades específicas del sistema, definidas por el diseñador del sistema SDL. El análisis efectuado con la herramienta de exploración del espacio del estado revelará errores de diseño sutiles, por ejemplo, carreras de señales imprevistas o temporizaciones infrecuentes, que resultan muy difícil de encontrar con la simulación o la inspección manual. Esto conlleva, a veces, el tener que hacer algunas modificaciones o ampliaciones de la especificación del sistema SDL para reducir el espacio del estado que se debe explorar. En la práctica, esto pudiera incluir limitar la cantidad de instancias admisibles en ciertos tipos de proceso o a la longitud de las colas de canal, o elaborar una especificación más o menos detallada del entorno del sistema.

Para sistemas SDL pequeños se puede verificar la ausencia de una propiedad, pero en general, esto resulta imposible debido a los problemas que plantea la explosión de estados. No obstante, aun en el caso de sistemas grandes y complejos, este tipo de herramienta ha demostrado ser muy útil como herramienta de análisis automático, que efectúa un análisis más profundo que el de las herramientas de simulación.

Existen varias herramientas basadas en la exploración del estado que están en uso, pero no están comercializadas, por lo que debe considerarse que están aún en la etapa de investigación.

### I.8.3 Prueba de conformidad

El tema de la *prueba de conformidad*, como se define en [29], implica la prueba tanto de las capacidades como del comportamiento de una realización. Los *requisitos de conformidad estática* definen las capacidades mínimas que tienen que ser sustentadas por una realización conforme. Lo que resulta de interés es el comportamiento externo de una realización, es decir, el comportamiento de la realización se define con respecto a la manera en que los eventos que puede ejecutar se observan en el entorno de la realización. Esto significa que, aun si el comportamiento externo y el interno se describen mediante una norma, la realización sólo tiene que cumplir los requisitos impuestos al comportamiento externo. A estos requisitos se los denomina *requisitos de conformidad dinámica*, y se definen como una relación en el comportamiento externo entre la realización y la norma de especificación.

Actualmente, las normas se indican en lenguaje natural; en algunos casos, van acompañadas de tablas de estado, y las series de prueba se elaboran manualmente. Esto plantea problemas para desarrollar las series de pruebas, puesto que no hay manera de verificar formalmente que una serie de pruebas específica el mismo conjunto de requisitos de la realización como lo hace la norma.

Se considera que la introducción de lenguajes formales, tales como el SDL, en las normas de especificación es una forma de resolver estos problemas. Cuando se dispone de especificaciones formales, surgirán posibilidades de elaborar pruebas informatizadas y/o validar las series de prueba. Durante los últimos años se han desplegado muchos esfuerzos de investigación en la elaboración de series de pruebas informatizadas, y es posible distinguir dos métodos:

- la elaboración automática; y
- la elaboración basada en la simulación.

# Reemplazada por una versión más reciente

## I.8.3.1 Elaboración automática de pruebas

En la elaboración automática de las pruebas se utilizan las técnicas disponibles en el sector de pruebas de soporte físico, o técnicas semejantes a las utilizadas en la validación, tales como la exploración del espacio del estado. La característica principal de este método es que se calcula una representación del comportamiento observable externamente del sistema. A partir de esa representación, se genera automáticamente la serie de pruebas, que se basa en una descripción del comportamiento que debe probarse. Esta información se puede basar en los gráficos de secuencias de mensajes, véase [30], si existen, o en los requisitos informales del sistema. Debe señalarse que se pueden especificar requisitos de realización que no se pueden poner a prueba. Esto debe tenerse presente al elaborar las normas de especificación. Para obtener especificaciones que se puedan probar, es preciso limitar de alguna manera la utilización de los constructivos SDL.

Son ejemplo de las técnicas empleadas en el cálculo del comportamiento observable las basadas en la exploración del espacio del estado y en el concepto de máquinas de estados infinitos. En esta última, los procesos del sistema se transforman en un conjunto de máquinas de estados finitos, que se componen para calcular el comportamiento observable. Puesto que la semántica del SDL se basa en el concepto de máquinas de estados finitos ampliadas, en general no se puede llevar a cabo esta transformación.

## I.8.3.2 Elaboración de pruebas asistida por simulador

El simulador puede servir de ayuda en la elaboración de pruebas, de diversas maneras. Primero, el diseñador de pruebas puede familiarizarse con el comportamiento de la especificación a través de una simulación de la especificación. Cuando se conocen los patrones de comportamiento que son pertinentes para la prueba, el simulador puede prestar ayuda en la elaboración de los casos de prueba al permitir al diseñador de pruebas que recorra interactivamente la especificación y defina los casos de prueba según el comportamiento de la especificación. En el mercado hay varios simuladores para SDL de diferentes marcas. Sin embargo, cuando el comportamiento de la especificación es complejo, la simulación de la especificación puede tomar mucho tiempo y ser una tarea difícil.

La elaboración de las pruebas también se puede automatizar en parte gracias a un simulador para fines especiales. En este caso, el diseñador de pruebas puede guiar la generación de pruebas al tomar decisiones interactivamente sobre los comportamientos para los cuales habrá que formular pruebas, si bien la generación efectiva de las pruebas se efectúa automáticamente.

## I.9 Documentos auxiliares

Los documentos auxiliares contienen información que se puede extraer (manualmente o mediante una herramienta) de una especificación de sistema SDL. El objetivo es proporcionar una visión general. Estos documentos también se pueden crear antes de la especificación del sistema SDL, en cuyo caso pueden servirle de base.

### I.9.1 Comunicación y especificación de interfaz

#### I.9.1.1 Introducción

En muchas situaciones, es deseable especificar la interfaz de una entidad como parte de la especificación de requisitos de esa entidad. A continuación se examina en general la comunicación y la especificación de interfaz, como base de las secciones siguientes, en las que se presentarán otras técnicas para este tipo de especificaciones.

Para comenzar, es preciso distinguir entre *comunicación e interfaz* (dinámica). Se supone que la comunicación se efectúa entre dos o más partes, y que todas las partes participantes tienen cierta influencia sobre el comportamiento combinado resultante.

Una interfaz, por otra parte, se aplica a una sola entidad<sup>7)</sup>. Una entidad puede tener más de una interfaz. Una especificación de interfaz expresa el comportamiento posible de la entidad en la interfaz dada, sin tener en cuenta el comportamiento de las restantes interfaces de la entidad. Se dice que el comportamiento en una interfaz dada es posible pues sólo está influida por la entidad. Cuando esta interfaz está conectada a alguna otra entidad, la comunicación entre esas dos entidades será un subconjunto del comportamiento posible en la interfaz.

Al analizar una interfaz, es común suponer que la entidad está conectada al *entorno* a través de la interfaz. El entorno, por su parte, no impone ninguna restricción al comportamiento de la entidad observado en la interfaz.

---

<sup>7)</sup> Esta terminología concuerda con los conceptos de modelado utilizados por el CCITT y la ISO para el procesamiento abierto distribuido (ODP, *open distributed processing*). Se reconoce que el término «interfaz» también se utiliza con significado diferente en otros trabajos de normalización.

# Reemplazada por una versión más reciente

En el contexto del SDL, esa situación se ilustra mediante un bloque *A* conectado al entorno por los canales *I1* e *I2*, o un bloque tipo *A* con las puertas *I1* e *I2* como interfaz (véase la Figura I.9-1). Obsérvese que el canal que conecta dos bloques es un medio de comunicación y no una interfaz. La entidad también podría ser un tipo de proceso, pero esta posibilidad no se trata explícitamente en lo que sigue.

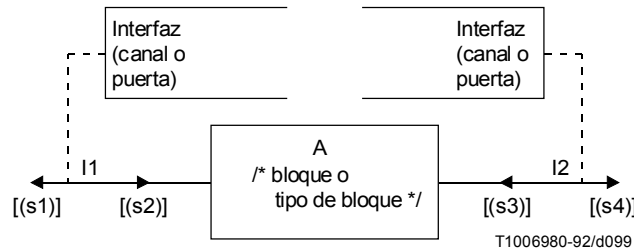


FIGURA I.9-1/Z.100

## Interfaz de un bloque o tipo de bloque

Las especificaciones de los canales/puertas (mediante las listas de señales *s1*, *s2*, *s3* y *s4*, junto con las especificaciones de señal correspondientes que no se muestran aquí) son también la especificación de la interfaz *estática*. En la especificación de una interfaz *dinámica*, o comunicación, se puede utilizar:

- gráficos de secuencias de mensajes (MSC);
- álgebra de procesos; o
- subestructura de canal.

Estas tres técnicas se describirán a continuación. La especificación de la interfaz *dinámica* o comunicación es aplicable cuando se puede utilizar un MSC, por ejemplo, en el **paso 2** y el **paso 5** de I.3.

Obsérvese que, en general, cada interfaz se especifica por separado. Esta es una de las razones por las que la especificación del bloque o del tipo de bloque no resulta demasiado útil en este contexto, ya que esa especificación abarca el comportamiento *combinado* que satisface simultáneamente a todas las interfaces del bloque o del tipo de bloque.

### I.9.1.2 Utilización de gráficos de secuencias de mensajes

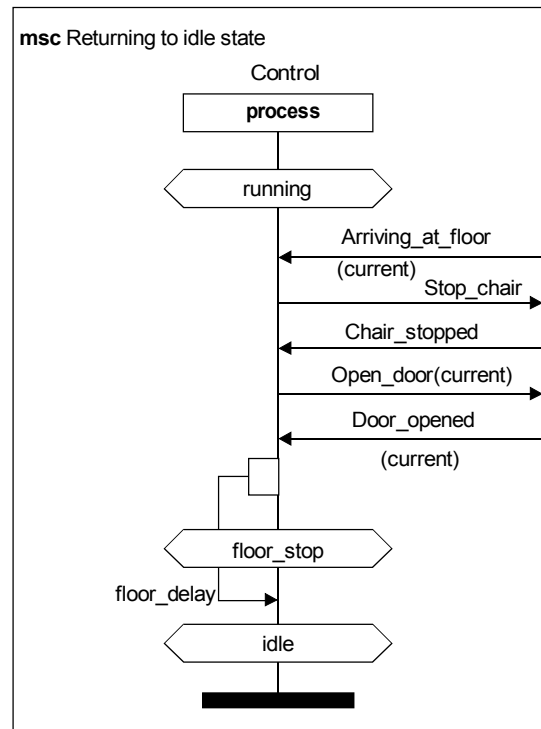
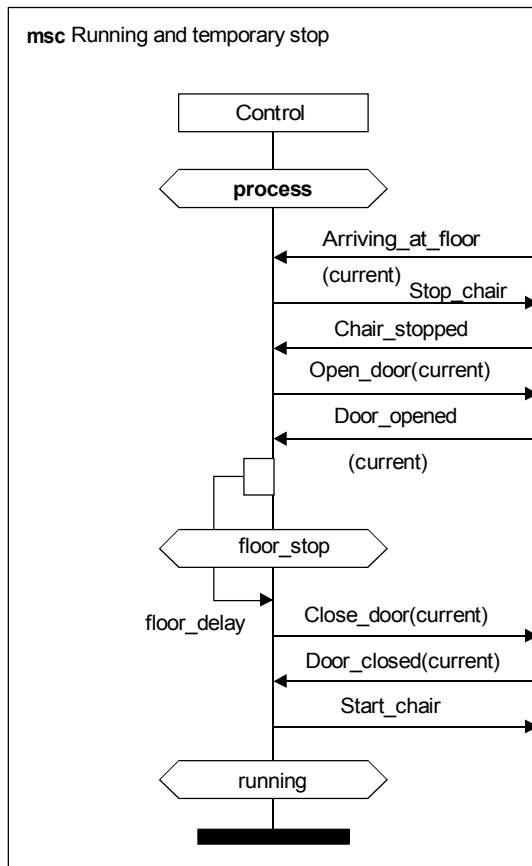
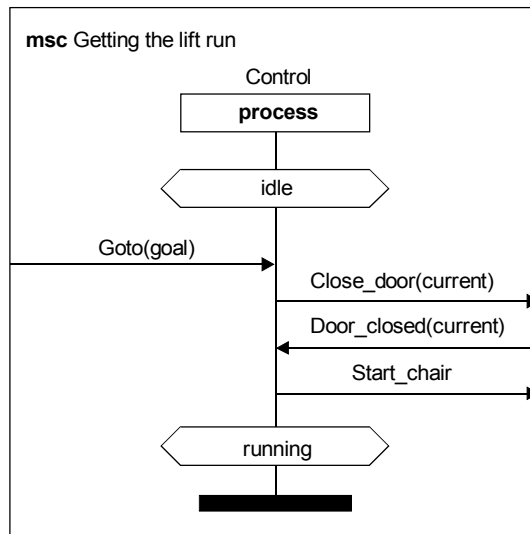
La utilización de gráficos de secuencias de mensajes se trata con detalle en I.6. Un MSC se puede emplear para muchos fines, entre los que cabe mencionar el enunciado de los requisitos de las especificaciones SDL, la especificación semiformal de la comunicación y la especificación de las interfaces.

Los diversos usos de un MSC están correlacionados; por ejemplo, una especificación de interfaz puede considerarse como parte de una especificación de requisitos. En realidad, no hay diferencia entre el empleo de un MSC para una especificación de interfaz y para la especificación de la comunicación. Esto es así porque un MSC sólo trata un comportamiento parcial (una de entre las muchas respuestas posibles de una entidad o del entorno a cierto estímulo).

Para facilitar la comprensión de los tres métodos mencionados más arriba relativos a la especificación de una interfaz dinámica de la comunicación, se utiliza el MSC del ejemplo *Lift* (*ascensor*). Algunos MSC ya se han presentado en I.3, **paso 5**, para el caso de un solo usuario *Lift ride* y *Lift request* (*viaje de ascensor y petición de ascensor*).

El caso de usuarios múltiples también se puede describir con MSC (véase el Ejemplo I.9-1). Dado que un MSC sólo describe el caso de un usuario, resulta ventajoso disponer de varios MSC pequeños, que puedan combinarse de diversas maneras. Es obvio que el MSC *Running and temporary stop* (*funcionamiento y parada provisional*) se puede «añadir» al MSC *Getting the lift run* (*hacer funcionar el ascensor*) varias veces antes de añadir el MSC *Returning to idle state* (*retorno al estado reposo*).

# Reemplazada por una versión más reciente



T1006990-92/d100

EJEMPLO I.9-1

MSC para el caso de múltiples usuarios

# Reemplazada por una versión más reciente

## I.9.1.3 Utilización de álgebra de procesos

### Introducción

En esta subcláusula se expone la utilización de un álgebra de procesos para la especificación de la comunicación y de la interfaz. El álgebra de procesos se basa en LOTOS, y está denotada por LOTOS\*. En comparación con los MSC, la ventaja principal de la utilización de LOTOS\* para la especificación de la comunicación y la interfaz radica en que es una especificación más completa y compacta. Debe señalarse que la utilización combinada de LOTOS\* y SDL que se describe en este texto no tiene fundamentos teóricos sólidos. Además, LOTOS\* no se conforma completamente con la definición de lenguaje oficial de LOTOS.

Cuando se utiliza LOTOS\* para la especificación de comunicación e interfaz, no hacen falta nuevos documentos. En LOTOS\* las expresiones de comportamiento se pueden incluir como comentarios en las especificaciones de puerta o de canal. Una expresión de comportamiento LOTOS\* define, mediante su semántica operativa, un conjunto de trazados, o sea, secuencias posibles de señales que atraviesan la puerta o el canal.

El empleo de LOTOS\* para la especificación de comunicación e interfaz conlleva algunos supuestos especiales. En el SDL, el mecanismo de comunicación es asíncrono, mientras que la comunicación en LOTOS es síncrona. Por consiguiente, no se usan los mecanismos de sincronización de LOTOS. Además, siempre se puede considerar que la comunicación se efectúa entre dos partes: el bloque y su entorno. Cuando sólo se tiene en cuenta el comportamiento observable, se puede suponer que cada una de estas partes sólo contiene un proceso SDL. Por ello, no hay paralelismo, y basta con considerar la parte secuencial de LOTOS.

En la especificación de interfaz hay dos clases de eventos: la entrada y la salida de una señal asíncrona. Para expresar esto en LOTOS\* se necesitan declaraciones de variables en algunas puertas, por ejemplo, las expresiones siguientes, que se corresponden en ambos lenguajes:

SDL	LOTOS*
input a;	? a
output a;	! a

Obsérvese que, por razones prácticas, en una expresión LOTOS\* se omite el nombre de puerta. Puesto que una puerta no está sincronizada con ninguna otra cosa, nada nos impide considerar que el valor y las variables en la puerta son el consumo y el envío de una señal asíncrona.

### Operadores LOTOS\*

LOTOS\* ofrece varios operadores, que se pueden utilizar en las especificaciones de comunicación e interfaz. Algunos se pueden expresar directamente en SDL, mientras otros tienen una representación más compacta que sus constructivos correspondientes en SDL.

- **Prefijo de acción:** El prefijo de acción (;) se puede utilizar para prefijar una expresión de comportamiento existente mediante una acción, por ejemplo, *a;exp*. Esto significa que a la ejecución de la acción *a* sigue inmediatamente la ejecución de la expresión de comportamiento existente *exp*. En el SDL, el prefijo de acción corresponde a la ordenación implícita de acciones en una transición.
- **Elección:** El operador de elección ([ ]) significa una elección no determinística entre dos expresiones de comportamiento. En SDL se puede expresar de dos maneras diferentes. Cuando se puede elegir entre las señales que se recibirán, el estado y las entradas diferentes conectadas al estado se encargarán de esa elección. En los demás casos, se emplea la decisión.
- **Entrelazado:** Como ya se ha dicho, no es necesario tener en cuenta la sincronización en la especificación de interfaz. Esto hace que sólo se pueda utilizar el entrelazado puro (||), por el que las acciones de las expresiones que conecta se pueden entrelazar arbitrariamente.
- **Habilitación:** El operador de habilitación (>>) significa una ordenación secuencial de dos expresiones de comportamiento de modo que la expresión de comportamiento del lado derecho se ejecuta sólo después de la terminación fructuosa (mediante **exit**) de la expresión de comportamiento del lado izquierdo. En el SDL, corresponde a nextstate (siguiente estado), o la ordenación normal de las acciones en una transición.
- **Inhabilitación:** El operador de inhabilitación (!>) significa que la expresión de comportamiento del lado derecho, al ser ejecutada, interrumpirá inmediatamente la expresión de comportamiento del lado izquierdo. En SDL, la inhabilitación se puede expresar convenientemente mediante un estado asterisco, seguido de la transición que interrumpe el comportamiento normal.

## Reemplazada por una versión más reciente

- **Salida:** Significa que la expresión de comportamiento ha terminado con éxito, y corresponde a un `nextstate` en SDL.
- **Parada:** Significa lo mismo que **stop** en SDL.
- **Recursión:** Significa repetidamente pasar a un estado anterior.

### Ejemplo

Veamos el sistema *Lift* que aparece en I.3.3. El bloque *Control* tiene tres interfaces, uno para cada canal conectado. En LOTOS\* las interfaces se representan como puertas. Las especificaciones de interfaz para *Chair* y *Floors* se muestran en el Ejemplo I.9-2:

```
Chair_interface :=
    ?Goto; Chair_interface
[]
    ?Emergency_stop; ?Restart; Chair_interface
Floor_interface :=
    Open_door >> Floor_request >> Close_door >> Running
Running :=
    ?Floor_req; Running
[]
    ?Arriving_at_floor; (Running [] Open_door >> Floor_stop)
Floor_stop :=
    ?Floor_req; Floor_stop
[]
    i; (Close_door; Running [] Floor_request >> Close_door >> Running)
Floor_request :=
    ?Floor_req; (Floor_request [] exit)
Open_door :=
    !Stop_chair; ?Chair_stopped; !Open_door; ?Door_opened; Exit
Close_door :=
    !Close_door; ?Door_closed; !Start_chair; Exit
```

### EJEMPLO I.9-2

#### Especificaciones de interfaz LOTOS\*

Compárese con la especificación del proceso *Control* que aparece en I.3.3. Como puede verse, la ejecución repetida de las transiciones en un estado se expresa en LOTOS\* mediante procesos recursivos, tales como *Running*.

Como ya se ha dicho, la especificación de interfaz puede ser utilizada como una especificación de requisitos para el bloque SDL. Al especificar el bloque (y sus procesos), se necesitan generalmente decisiones adicionales relativas al comportamiento detallado y a la combinación de comportamientos en cada interfaz. Son ejemplos de esas decisiones: ¿Qué sucederá si llegan *Floor\_req* u otra *Goto* mientras el ascensor está funcionando? ¿Qué información deberán transportar las señales y qué información se almacenará dentro del bloque? No obstante, de una especificación de interfaz se puede extraer automáticamente una especificación de proceso SDL esquemática.

#### Semántica y conformidad de las especificaciones en LOTOS\*

A continuación se analizará brevemente la relación entre las especificaciones expresadas en LOTOS\* y las especificaciones SDL completas.

Primero, hay que construir un modelo de semántica dinámica de un sistema formado por procesos SDL. Para ello, se emplean árboles de comunicación asíncrona. Este modelo se basa en sistemas de actor y sus diagramas de eventos [23]. La definición de los árboles indica cuáles son las secuencias posibles de entradas y salidas de señal observables. La definición de árbol de comunicación asíncrona de un sistema SDL y su entorno proporciona la base semántica necesaria para formalizar la conformidad de una especificación de proceso SDL con una especificación de interfaz.

## Reemplazada por una versión más reciente

El paso siguiente consiste en definir un árbol similar para las especificaciones de interfaz. Esto es más o menos, la semántica operativa de LOTOS\* y los gráficos de transiciones etiquetadas que se basan en ella [31]. Intuitivamente, los trayectos de ese gráfico de transición representan las secuencias de acciones que autoriza la expresión de comportamiento correspondiente, al igual que lo haría un árbol de comunicación asíncrona.

La conformidad de una especificación de proceso SDL con una especificación de interfaz se puede definir como una relación binaria entre los nodos de los árboles de comunicación asíncrona de la especificación de proceso SDL y la especificación de interfaz. Intuitivamente, se deberán cumplir tres condiciones:

- todo lo que puede hacer el proceso SDL debe estar permitido en la especificación de interfaz, lo que constituye el requisito normal de refinamiento;
- el proceso SDL deberá estar preparado para aceptar en todo momento cualquier señal que pueda ser enviada al proceso, según la especificación de interfaz;
- el proceso SDL debe llevar a cabo su tarea hasta el final, es decir, no puede detenerse en ningún momento.

Un aspecto interesante de la conformidad es la distinción entre la entrada y la salida de las señales. Intuitivamente está claro que si, en algún momento, se pueden enviar varias señales diferentes al proceso según lo indica la especificación de interfaz, el proceso debe ser capaz de tratarlas todas. Si algunas son descartadas, lo que significa que no se obtiene la respuesta prevista en la especificación de interfaz, la especificación SDL detallada correspondiente a la especificación de interfaz no puede ser correcta. Sin embargo, cuando se permite varias señales de respuesta a una señal de entrada, la elección se deja al arbitrio del especificador. A este respecto, los requisitos determinados por las señales de entrada y de salida son duales.

### I.9.1.4 Utilización de subestructura de canal

La subestructura de canal no es un documento auxiliar, pero se puede utilizar en la especificación dinámica de comunicación y de interfaz, lo que se muestra para completar la visión general. Se introduce la subestructura de canal *Floorsub* para el canal *Floors* del sistema *Lift* (véase el Ejemplo I.9-3).

La parte dinámica de la especificación de interfaz viene dada por la especificación de proceso *Floor*. Esto debe ser equivalente a la especificación de proceso *Floor\_interface* en LOTOS\* del Ejemplo I.9-2.

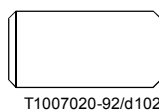
La temporización se expresa mediante una transición espontánea, y las decisiones no especificadas, mediante decisiones no determinísticas. Los procedimientos *Open\_door* y *Close\_door* se especifican en I.3.3 como parte de la especificación de proceso *Control*.

### I.9.2 Diagrama arborescente

El diagrama arborescente muestra los componentes de un sistema SDL. Los componentes (denominados también nodos) forman una estructura jerárquica, con el sistema como la raíz, según las relaciones de contención expresadas por las reglas sintácticas. Los nodos de un diagrama arborescente (además del nodo-sistema) pueden ser:

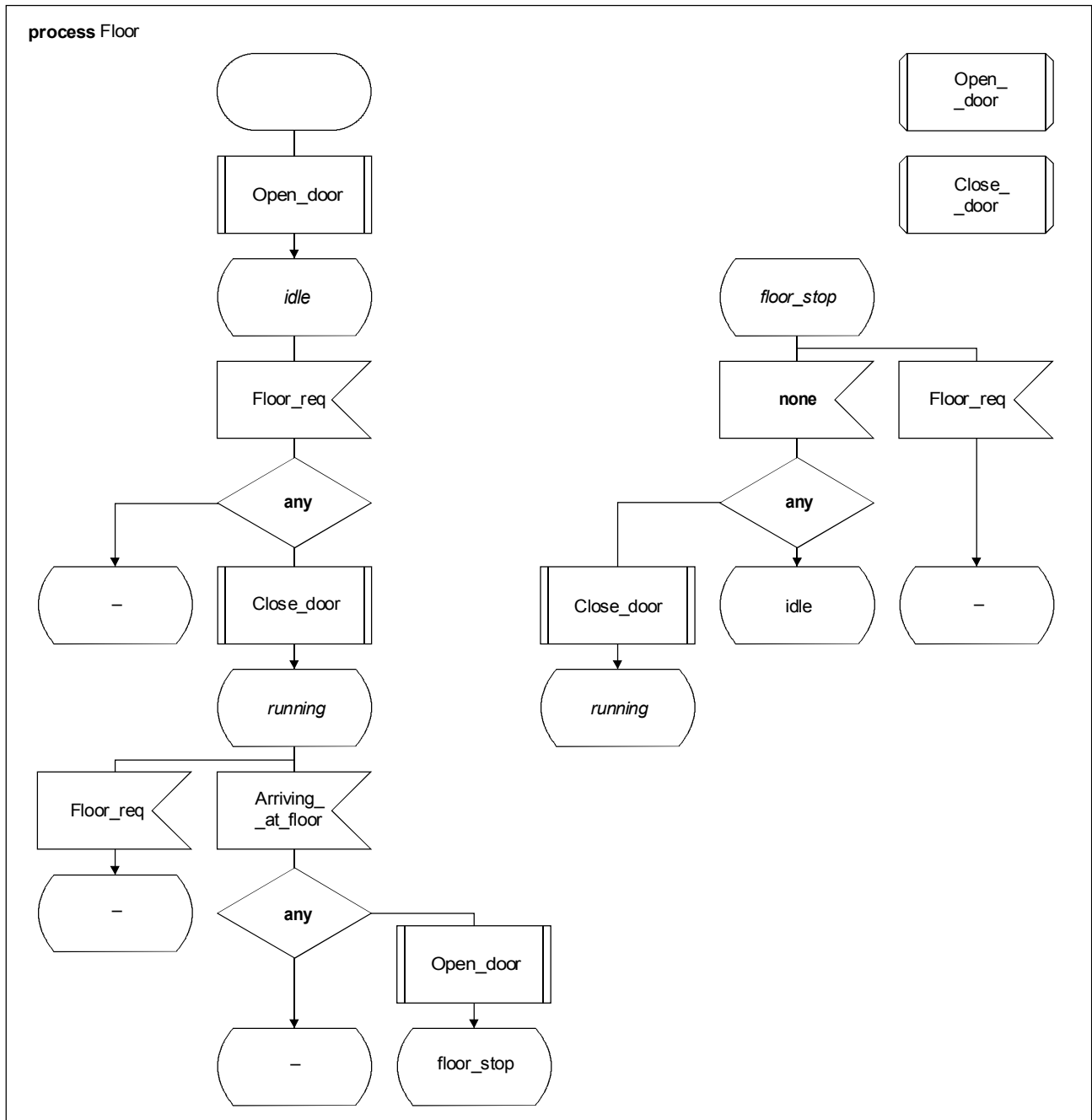
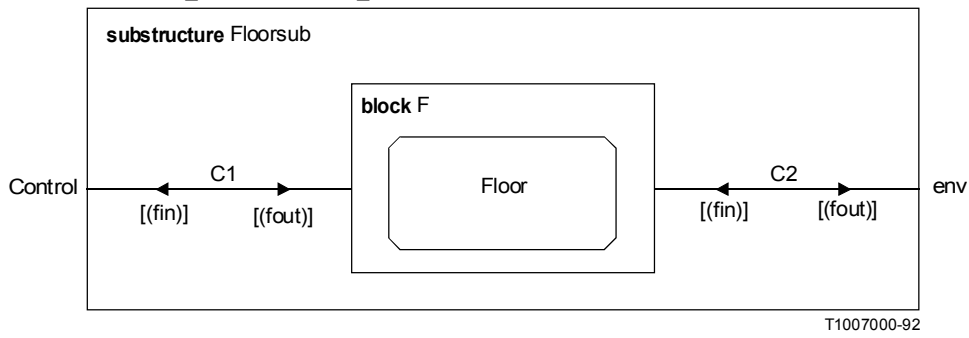
- bloques únicamente, denominado *diagrama arborescente de bloques* (véase la Figura I.9-2);
- bloques, procesos y servicios, denominado *diagrama arborescente básico* (véase la Figura I.9-3);
- además de los nodos del *diagrama arborescente básico*, procedimientos, macros, etc., denominado *diagrama arborescente general* (véase la Figura I.9-5).

Nótese que las diversas ramas de un diagrama arborescente no tienen que contener el mismo número de nodos. Todos los nodos representan definiciones de entidades. Se ha introducido un símbolo nuevo para el nodo de macro



puesto que en la presente Recomendación no se define un símbolo para la definición de macro. Téngase presente que este símbolo no es normativo.

# Reemplazada por una versión más reciente



EJEMPLO I.9-3  
Subestructura del canal *floors*



## Reemplazada por una versión más reciente

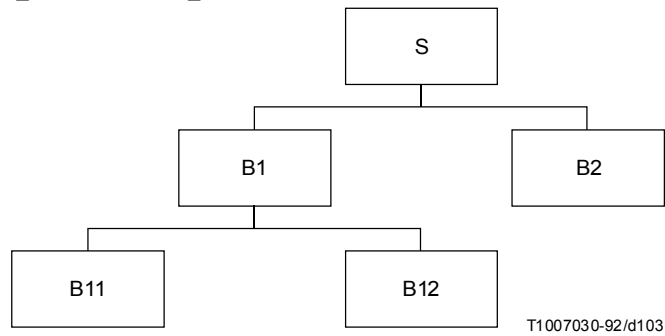


FIGURA I.9-2/Z.100

Diagrama arborescente de bloques

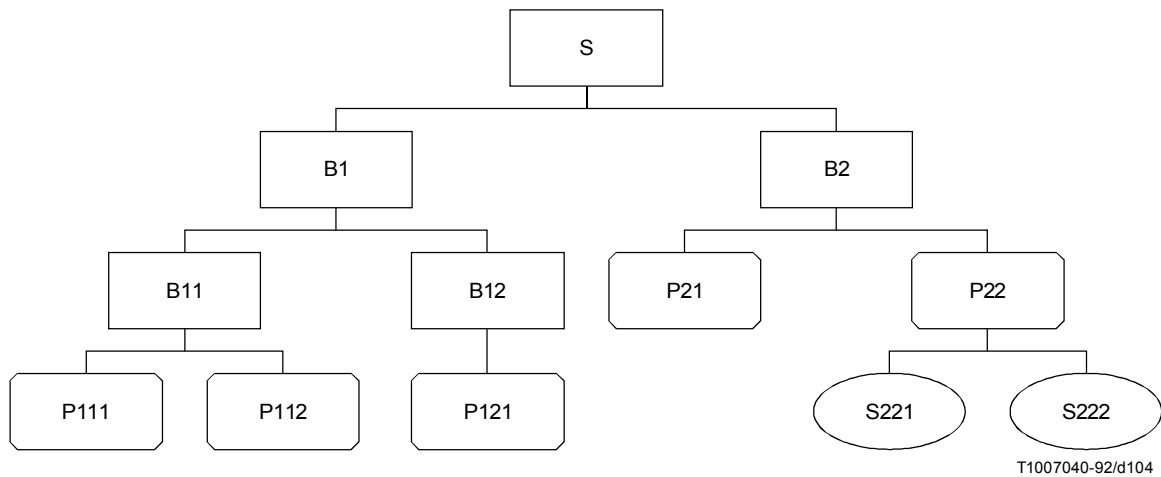


FIGURA I.9-3/Z.100

Diagrama arborescente básico

Los nodos de macro son siempre nodos terminales, y pueden estar unidos a cualquier otro nodo. Un nodo de procedimiento puede estar unido a cualquier otro nodo, excepto el nodo de macro.

De preferencia, todos los símbolos de nodo del diagrama tendrán un tamaño uniforme, lo que permitirá que todos los nodos que se encuentran en el mismo nivel de partición aparezcan como un nivel uniforme en el diagrama.

Un diagrama arborescente general también puede contener subestructura de canal, como se muestra en la Figura I.9-4. En ese ejemplo, hay una subestructura de canal bidireccional *CI* entre los bloques *B1* y *B2*. El nodo de subestructura de canal es análogo a un nodo de bloque y, por consiguiente, puede ser la raíz de un árbol similar de nodos.

# Reemplazada por una versión más reciente

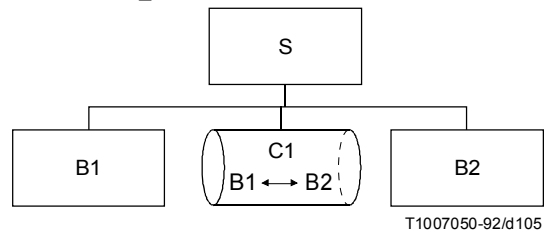


FIGURA I.9-4/Z.100

**Diagrama arborescente general con subestructura de canal**

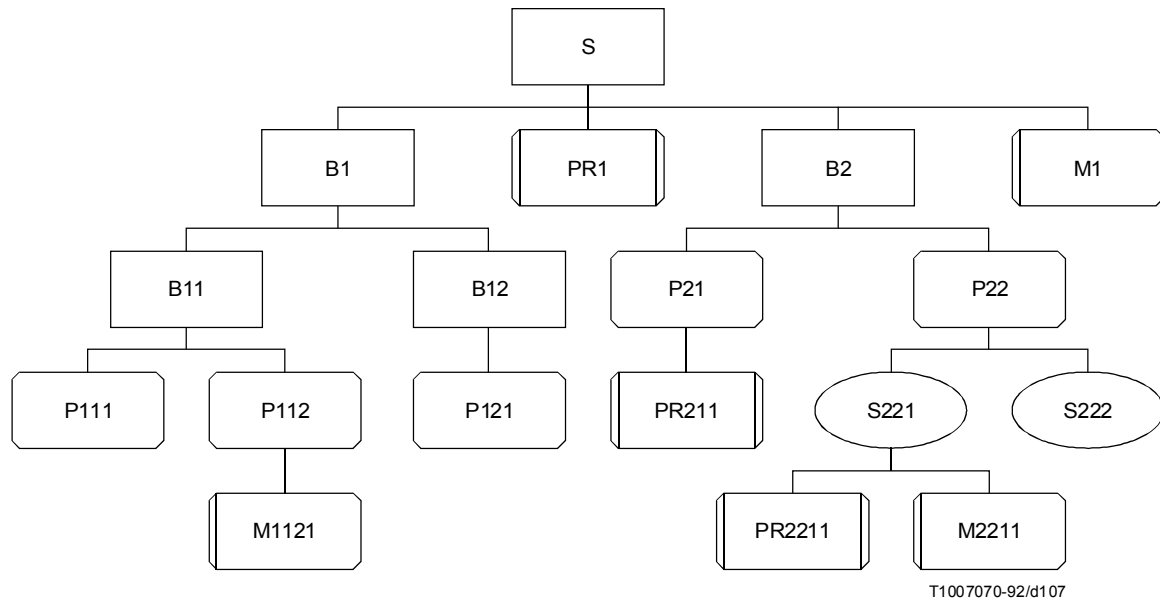
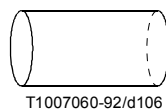


FIGURA I.9-5/Z.100

**Diagrama arborescente general**

Se ha introducido un símbolo nuevo para el nodo de subestructura de canal



porque la presente Recomendación no define un símbolo especial para la definición de subestructura de canal. Este símbolo no es normativo.

## Reemplazada por una versión más reciente

Con frecuencia, resulta útil fraccionar un diagrama arborescente en varios diagramas parciales, por ejemplo, cuando el diagrama es tan grande que abarca más de una página. La división se efectúa de manera que las raíces de los diagramas adicionales aparezcan como nodos terminales del primer diagrama (véase la Figura I.9-6).

Cuando no resulta evidente que un nodo terminal de un diagrama se seguirá dividiendo en otros diagramas y/o dónde hallar los diagramas de continuación, se insertará referencias mediante el símbolo de comentario.

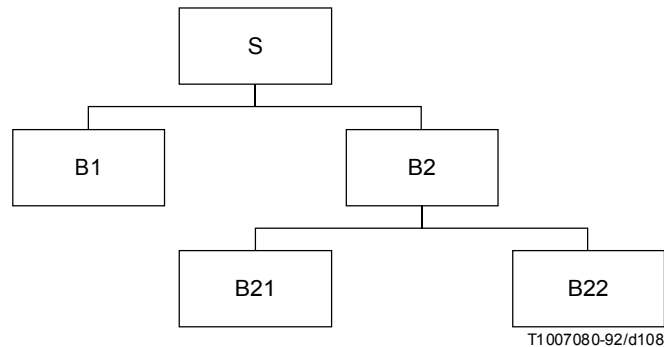


FIGURA I.9-6a/Z.100

### Partición de un diagrama arborescente – Diagrama completo

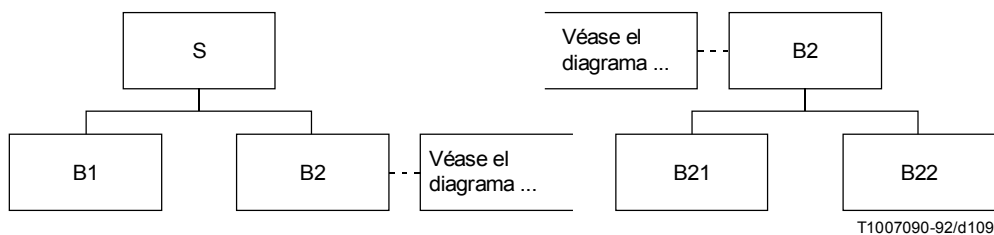


FIGURA I.9-6b/Z.100

### Partición de un diagrama arborescente – Diagrama parcial

### I.9.3 Diagrama general de estados

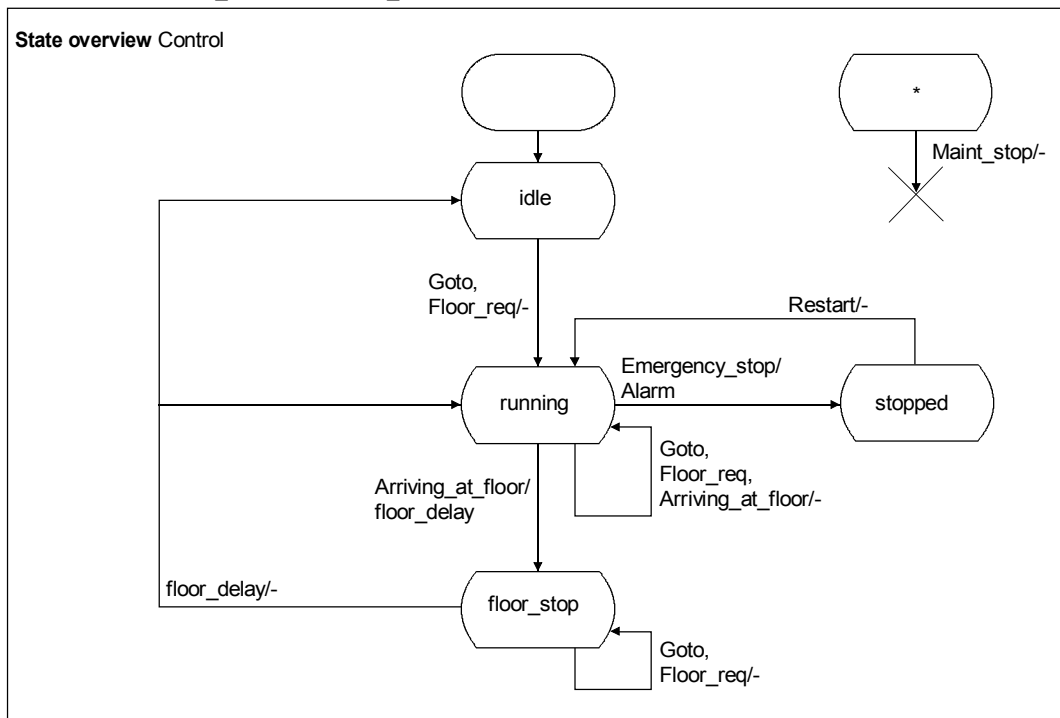
El propósito del diagrama general de estados es dar una visión general de los estados de un proceso, y mostrar las transiciones posibles entre ellos. El diagrama se complica muy pronto, al aumentar el número de estados y transiciones. Por consiguiente, sólo es aplicable en casos sencillos, o cuando hay estados «no importantes» o transiciones que se puede omitir.

El diagrama se compone de símbolos de estado, arcos dirigidos que representan transiciones y, facultativamente, símbolos de arranque y parada.

El símbolo de estado contendrá el nombre del estado referido. El símbolo puede contener varios nombres de estado o una notación de asterisco (\*).

A cada arco dirigido se le puede asociar el nombre de la señal, o conjunto de señales, que causa la transición, así como las señales enviadas durante la transición. La lista de señales enviadas está precedida por el carácter /. Las señales de temporizador y de fijación de temporizadores pueden tratarse como señales ordinarias. En la Figura I.9-7 aparece un ejemplo de diagrama general de estados para el proceso *Control* de I.3-3.

# Reemplazada por una versión más reciente



T1007100-92/d110

FIGURA I.9-7/Z.100

Diagrama general de estados

## I.9.4 Matriz de señales y estados

La matriz de señales y estados es una alternativa al diagrama general de estados que contiene exactamente la misma información. Se puede utilizar incluso si hay un gran número de estados y transiciones. Además, permite comprobar que se tiene en cuenta todas las combinaciones de estados y señales de entrada. Véase la Figura I.9-8, que corresponde al diagrama general de estados de la Figura I.9-7.

El diagrama consiste en una matriz bidimensional, con índices en un eje de todos los estados del proceso, y en el otro, de todas las señales de entrada válidas del proceso. En cada uno de los elementos de la matriz se ofrece el estado siguiente, junto con las salidas posibles durante la transición. Se puede dar una referencia sobre dónde encontrar la combinación indicada por los índices, si es que existe.

Se utiliza el elemento correspondiente al estado ficticio «start» y la señal vacía para indicar cuál es el estado inicial del proceso.

La matriz se puede dividir en matrices parciales contenidas en páginas diferentes. Las referencias serán las que el usuario utiliza normalmente en la documentación.

De preferencia, se agruparán las señales y los estados, para lograr que cada matriz parcial abarque un aspecto del comportamiento del proceso.

## I.10 Documentación

### I.10.1 Introducción

La ISO define un documento como *un volumen de información limitado y coherente almacenado en un medio en una forma recuperable*. Por consiguiente, debe considerarse como una unidad lógica, que está estrictamente delimitada. Los documentos se utilizan para transmitir toda la información relacionada con un sistema que se especifica con el SDL.

## Reemplazada por una versión más reciente

	<b>signal state</b>	Control					
	state → signal ↓	'start'	<i>idle</i>	<i>running</i>	<i>floor_stop</i>	<i>stopped</i>	
		idle/-					
	Goto		running/-	-/-	-/-		
	Floor_req		running/-	-/-	-/-		
	floor_delay				idle, running/-		
	Arriving_ _at_floor			floor_stop/ floor delay; -/-			
	Emergen_ cy_stop			stopped/ Alarm			
	Restart					running/-	
	Maint_stop		'stop'/-	'stop'/-	'stop'/-	'stop'/-	

FIGURA I.9-8/Z.100

### Matriz de señales y estados

Cuando el medio físico de almacenamiento de un documento es el papel, con frecuencia se aplica erróneamente el término «documento» a las hojas de papel en vez de a su contenido lógico. Con el creciente uso de medios magnéticos de almacenamiento se está utilizando el término cada vez más en su sentido original.

A continuación se analiza la organización lógica de los documentos en vez de organización física. Esta se deja al arbitrio del usuario, pero la semejanza entre los requisitos de organización lógica y física permite ofrecer algunas orientaciones que ayudarán a los usuarios a establecer la organización física de sus documentos.

Se puede facilitar la lectura y el manejo de la descripción del sistema si se divide la información en un número conveniente de documentos. La estructura de la documentación proporcionará una visión general y los detalles.

Son propiedades de un documento:

- la identificación única;
- la designación de revisión;
- el tamaño manejable;
- es (generalmente) parte de una estructura de documentación;
- no es parte de otro documento (o sea, los documentos no están anidados);
- está dividido (generalmente) en páginas.

El SDL no recomienda ciertos documentos o estructuras de documentación. No obstante, proporciona algunos constructivos de lenguaje, que ayudan al usuario a manejar los documentos. Esto se analiza también en esta sección para completarla.

# Reemplazada por una versión más reciente

## I.10.2 Apoyo de lenguajes para la documentación

### Especificaciones anidadas

Como muchos otros lenguajes, una especificación SDL contiene una jerarquía de componentes, que se dispone en una estructura de tipo árbol. Ello hace que la especificación de sistema tenga un número de niveles de jerarquía o abstracción. En la Figura I.10-1 se ilustra la estructura sintáctica clásica de una especificación de sistema en SDL. Se muestran las especificaciones anidadas, con las especificaciones de orden más bajo contenidas dentro del orden superior siguiente de especificación. Esto puede compararse con un diagrama de circuitos, que está completamente contenido en una sola hoja.

Si bien es cierto que la anidación de especificaciones es conveniente para el diseñador de herramientas, y cuando se trata de especificaciones muy pequeñas, plantea los problemas siguientes al usuario (humano):

- no ofrece una visión general;
- no presenta separación de los niveles de abstracción;
- hay demasiada información en un mismo lugar;
- resulta difícil la correspondencia de documentos y especificaciones.

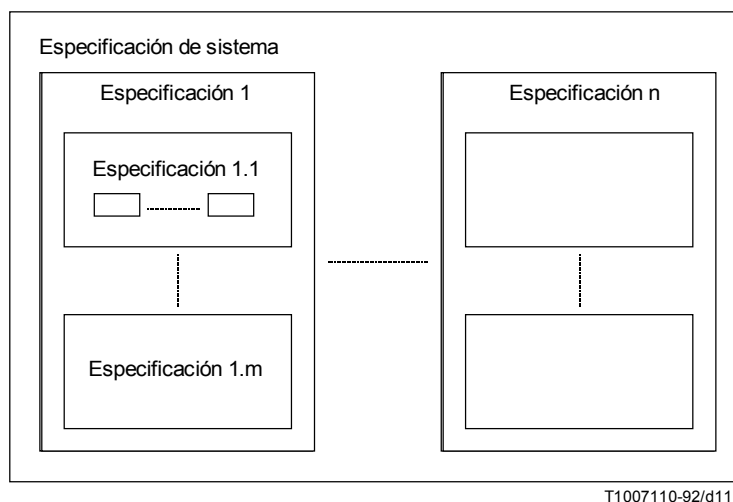


FIGURA I.10-1/Z.100

### Estructura sintáctica clásica de una especificación de sistema

### Especificaciones distantes

Estos problemas se han resuelto en el SDL mediante la introducción de constructivos de **especificación distante**. Una especificación distante es una especificación que ha sido eliminada de su contexto de definición para obtener una visión general. Es algo similar a llamar y definir un procedimiento, sólo que la «llamada» se hace exactamente desde un lugar (el contexto de definición) mediante una **referencia**. Dicho de otra manera, entre la referencia y la especificación distante hay una correspondencia de uno a uno (véase la Figura I.10-2). Esto se puede comparar a un diagrama de circuitos, que se presenta como diagrama de bloques, en el que los bloques se describen en páginas separadas como circuitos con componentes electrónicos.

Una especificación de sistema que utiliza especificaciones distantes es una representación plana, a diferencia de la representación jerárquica cuando se utilizan especificaciones anidadas.

### Combinación de representación gráfica y textual

Mediante especificaciones distantes, el usuario puede mezclar a discreción las formas de representación gráfica y la textual (véase la Figura I.10-3).

# Reemplazada por una versión más reciente

Una buena práctica consiste en comenzar por un diagrama de sistema para dar una visión general. En las especificaciones que exigen mucho texto se utilizará la representación textual. Esto se ilustra en el ejemplo del ascensor, de I.3.

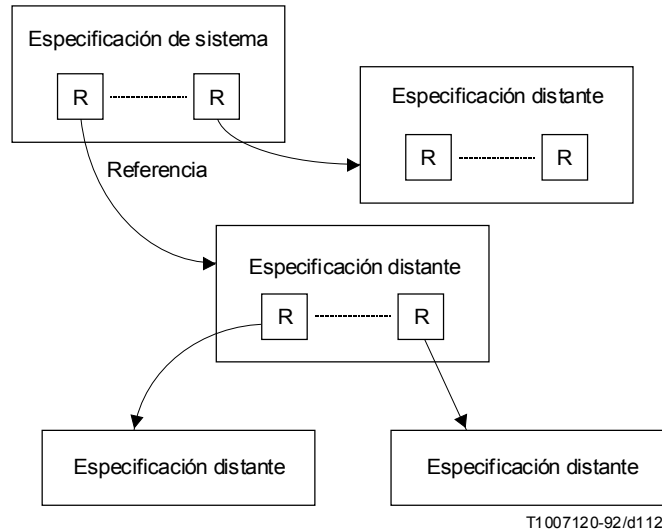


FIGURA I.10-2/Z.100  
Especificaciones distantes

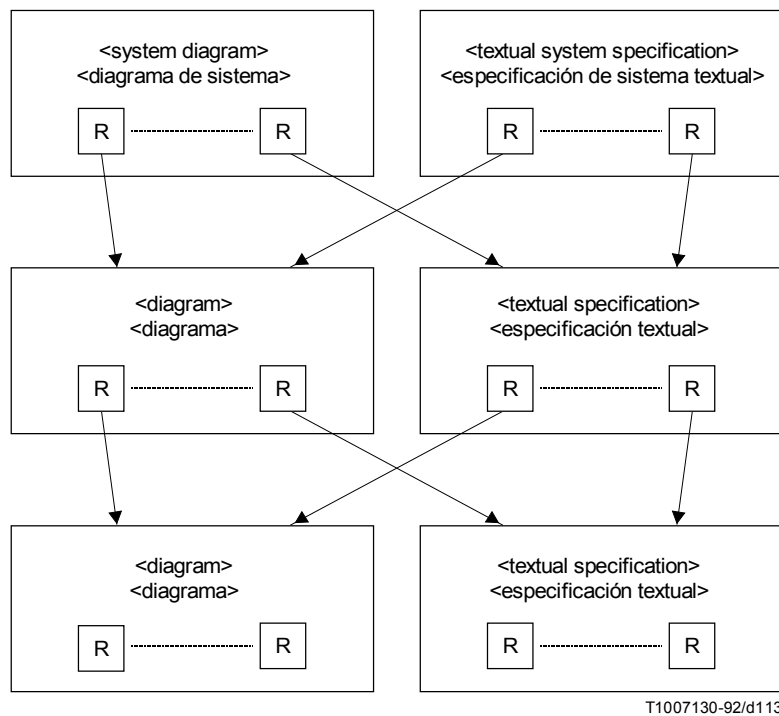


FIGURA I.10-3/Z.100  
Combinación de representación gráfica y textual

# Reemplazada por una versión más reciente

## I.10.3 Correspondencia de las especificaciones con los documentos

Si se considera la especificación de sistema como un conjunto de especificaciones distantes, puede decirse que un documento es un contenedor de una o más de estas especificaciones:

```
<document> ::=  
    <system specification>  
    | {<remote specification> }+
```

Téngase en cuenta que una especificación distante puede contener especificaciones anidadas.

Si la especificación de sistema es pequeña y anidada, bastará con un solo documento. Cuando se utiliza una representación plana, puede emplearse más documentos, por ejemplo, un documento por especificación (diagrama).

Es probable que el caso normal consista en una mezcla de representaciones planas y anidadas. Al decidir esta mezcla, se cumplirán las reglas siguientes:

- una especificación no debe dividirse entre varios documentos;
- cuando una especificación se coloca en un documento separado, debe ser una especificación distante;
- si se utiliza el concepto de página lógica para dividir un diagrama en varias páginas, las páginas del diagrama coincidirán con las páginas físicas del documento;
- si un diagrama abarca más de una página, debe ser una especificación distante;
- una zona de interacción de bloques, procesos y servicios debe caber en una sola página.

La sintaxis gráfica proporciona los medios para numerar cada página lógica de un diagrama, aunque esto no es absolutamente necesario. Por lo común, la ordenación de las páginas no importa, o se puede obtener de cualquier manera. Lo importante es repetir el encabezamiento en cada página para saber a qué diagrama pertenece la página.

En la representación textual, el lenguaje no proporciona los medios para numerar páginas aunque esto es crucial para evitar ambigüedades. El motivo es seguir el convenio de lenguajes de programación, que utilizan un esquema de numeración, externo al lenguaje.

## I.10.4 Cuestiones relativas a la documentación

### I.10.4.1 Estructura de documentación

El conjunto de documentos que abarca un sistema completo se deberá estructurar de forma que la relación entre un documento y los demás resulte clara. Normalmente, un documento contiene información sobre todo el sistema o sobre uno de sus componentes, y esto también debe quedar claro. Esto se consigue en general asociando un «plano» de documentos a cada componente y con una referencia a todos los documentos que pertenecen al componente. La relación entre componentes viene dada por la estructura del sistema (véase la Figura I.10-4).

Cuando la especificación SDL de bajo nivel (véase I.7) refleja la estructura del sistema realizado, la descripción de la estructura de sistema se puede generar automáticamente mediante una herramienta de la especificación de sistema SDL.

Sin embargo, la especificación SDL solo ofrece una vista del sistema, que puede cambiar con el tiempo, quizás varias veces al año. Para la mayoría de los sistemas reales, habrá que proporcionar más información sobre configuración, gestión de las revisiones, ordenación, producción, etc. También habrá que seguir elaborando la naturaleza y las clases de componentes del sistema. Estas cuestiones se analizan a continuación.

### I.10.4.2 Componentes de sistema

Un componente de sistema (incluido el propio sistema) evoluciona a través de varias etapas de revisión bien definidas. Cada nueva etapa de revisión será compatible con las etapas de revisión anteriores, conforme a ciertos criterios estipulados en la norma de organización. La corrección de errores y/o la introducción de características nuevas determinan una nueva etapa de revisión. Si no se puede lograr la compatibilidad hacia atrás, se deberá crear un nuevo componente (variante) en vez de una nueva etapa de revisión. Un componente de sistema tendrá una identidad y designación únicos para la etapa de revisión.



# Reemplazada por una versión más reciente

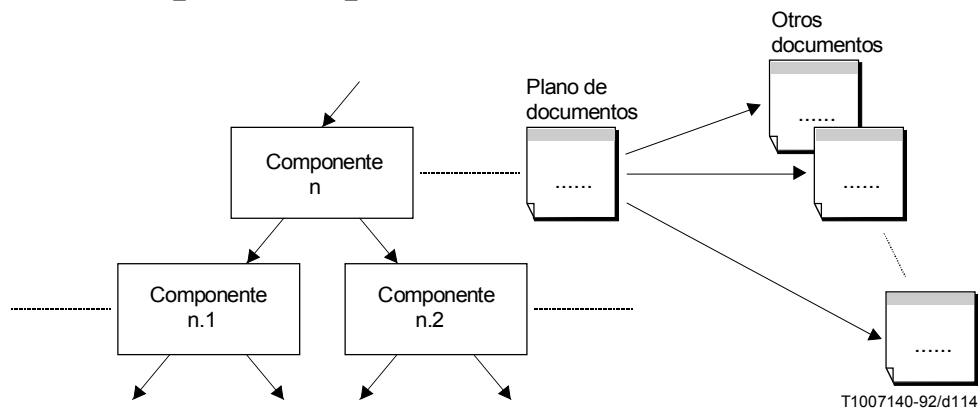


FIGURA I.10-4/Z.100  
Estructura de documentación

Estas características generales de un componente de sistema se pueden aplicar de manera más o menos estricta según el uso que se haga del componente. Desde el punto de vista del uso, cabe distinguir entre:

- componentes locales;
- componentes normalizados; y
- componentes de entrega.

Un componente local podrá ser un procedimiento utilizado en un sistema por un grupo de desarrollo. Es evidente que el tratamiento seguro (identificación, documentación, cambios, etc.) de un componente local no plantea mayores problemas.

Un componente normalizado está generalmente disponible para todos los sistemas de una organización. Son ejemplos los componentes de soporte físico y los tipos de datos abstractos (géneros). Es obvio que los componentes normalizados no deben modificarse con frecuencia, o nunca se modificarán.

Un componente de entrega será utilizado por las organizaciones cliente, en etapas de revisión diferentes, posiblemente a nivel mundial. Es obvio que en el manejo de los componentes de entrega debe observarse reglas estrictas.

¿Cómo se tratarán las entidades SDL a este respecto? Los *tipos* son candidatos naturales a componentes normalizados. Los *bloques* se utilizarán para componentes de entrega, si forman parte de una especificación SDL (de bajo nivel). No obstante, hay componentes de entrega que la especificación SDL no puede tratar de forma perceptible. Todas las entidades SDL restantes se considerarán componentes locales.

Los identificadores SDL no incluyen la etapa de revisión y no son suficientes para los componentes de entrega. En la configuración y gestión de las revisiones de los componentes normalizados y de entrega se utilizará alguna técnica o herramienta existente.

### I.10.4.3 Clase de documentos

Las clases de documentos que necesita un sistema dependen de la finalidad del sistema y de las actividades que será preciso ejecutar para lograrla. Son actividades pertinentes, en general:

- ordenación;
- producción;
- operación;
- desarrollo;
- corrección de fallos.

# Reemplazada por una versión más reciente

Una regla general es que una pieza de información debe estar contenida en un solo documento, para evitar las incoherencias ocasionadas por los cambios. Cuando esto no sea posible o conveniente, uno de los documentos se considerará como documento básico, y los restantes, como documentos derivados.

En general, los documentos necesarios de *Desarrollo* forman normalmente la base de los otros documentos. Una especificación SDL abarca la estructura y el comportamiento, y debe ser el núcleo de esta clase de documentos.

Los documentos de *Operación* contienen principalmente información derivada. Un problema especial que plantean estos documentos es cómo adaptarlos a las necesidades del cliente, de manera que contengan únicamente los elementos que se han entregado al cliente. Algunos constructivos SDL que resultan útiles en esta actividad son *option*, *external synonym* y *subtyping* (especialización).

## I.10.4.4 Plantillas de documentos

Se suministrará una plantilla para cada tipo de documento, con el fin de lograr un estilo de documentación uniforme y también facilitar la elaboración de documentos individuales. Una plantilla especifica el tipo de información que se tratará, un índice de materias, la presentación y quizás, otros aspectos de la documentación que se considere necesario normalizar.

Las diferentes categorías de componentes necesitan diferentes conjuntos de tipos de documentos. En esta subcláusula sólo se analizará la documentación de un componente de entrega. Por lo común, se utilizan los siguientes tipos de documento; algunos son obligatorios, otros se emplean según proceda.

- **Especificación de estructura** – Componentes en forma de lista, que puede también incluir la etapa de revisión más antigua que se necesita.
- **Información de revisión** – Se prepara para cada etapa de revisión, y describe las diferencias con respecto a la etapa de revisión previa.
- **Plano de documentos** – Enumera todos los demás documentos que pertenecen al componente, a la vez que indica las etapas de revisión correspondientes.
- **Información de ordenación** – Describe las propiedades de los componentes desde el punto de vista de la ordenación, y también las características facultativas y los códigos (valores de sinónimos externos) para seleccionarlos.
- **Descripción** – Proporciona la estructura completa y quizá también una descripción del comportamiento en SDL. Puede contener la descripción de comportamiento sólo para componentes del nivel más bajo del sistema (véase I.3). Se puede presentar la especificación SDL como anexo al documento principal, con una breve descripción informal como introducción a la especificación SDL.
- **Especificación de prueba** – Especifica las pruebas que el componente debe pasar antes de ser utilizado.
- **Descripción de instrucciones** – Describe la sintaxis y la semántica de una instrucción, impartida durante *Operación*, que es ejecutada principalmente por el componente.
- **Descripción de copia impresa** – Describe la sintaxis y la semántica de un mensaje impreso en papel, generado por el componente durante *Operación*.
- **Manual de instalación** – Describe el proceso de instalación del componente.
- **Manual de operación** – Describe cómo un usuario/operador debe manejar correctamente el componente durante *Operación*.

## Referencias

- [1] *Basic Reference Model*, International Standard, ISO/IS 7498, 1984.
- [2] *OSI Service conventions*, Technical Report ISO/TR 8509, 1987.
- [3] Recomendación I.130 del CCITT – *Método de caracterización de los servicios de telecomunicación soportados por una RDSI y de las capacidades de red de una RDSI*.
- [4] Recomendación Q.65 del CCITT – *Etapa 2 del método de caracterización de los servicios soportados por una RDSI*.

## Reemplazada por una versión más reciente

- [5] Recomendación Z.120 del CCITT – *Gráficos de secuencias de mensajes*, Ginebra, 1993.
- [6] HARTMUT (B): *Fundamentals of algebraic specification*, Volume 1, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.  
NACHUM: *Termination of rewriting*, Journal on Symbolic Computation, (3), pp 69-116, 1987.  
HARALD: *A completion procedure for conditional equations*, Proc. 1st Int. Workshop on Conditional Term Rewriting, LNCS 308, pp. 62-83, Orsay, 1988, Springer-Verlag.
- [7] CHONG-YI: *Process periods and system reconstruction*, Lecture Notes in Computer Science 222, Advances in Petri Nets, (G. Rozenberg ed.) Springer Verlag 1986.
- [8] ENCONTRE, DELBOULBE, GAVAUD, LEBLANC, BOUSSALEM: *Combining services, Message Sequence Chart and SDL: Formalism, method, tools*, en [10].
- [9] FÆRGEMAND, MARQUES Editors: *SDL '89 The language at work*, Proceedings of the 4th SDL Forum, North-Holland, Amsterdam, 429 p. 1989.
- [10] FÆRGEMAND, REED Editors: *SDL '91 Evolving methods*, Proceedings of the 5th SDL Forum, North-Holland, Amsterdam, 1991.
- [11] GRABOWSKI (R): *Putting extended sequence charts to practice*, en [9].
- [12] HOGREFE: *OSI formal specification case study: the INRES protocol and service*, Technical Report, Universidad de Berna, abril de 1991.
- [13] KRISTOFFERSEN: *Message Sequence Charts and SDL specification consistency check*, en [10].
- [14] NAHM: *Consistency analysis of Message Sequence Charts and SDL systems*, en [10].
- [15] REISIG: *Petri Nets*, EATC Monographs on Theoretical Computer Sciences, Vol. 4, Springer Publ. Comp., 1985.
- [16] GRAUBMANN, GRABOWSKI (R): *Towards an SDL design methodology using sequence chart segments*, en [10].
- [17] HOARE: *Communicating sequential processes*, Prentice-Hall International, 1985.
- [18] PARNAS: *On the criteria to be used to decompose systems into modules*, Comm. ACM, vol. 15 1972.
- [19] PLOTKIN: *A structural approach to denotational semantics*, Report DAIM-FN-19, Computer Science Dept, Århus University, Dinamarca, 1981.
- [20] *A common semantics representation for SDL and TTCN*, European Telecommunications Standards Institute (ETSI) Technical Report, 1992.
- [21] GODSKESSEN: *A compositional operational semantics for Basic SDL*, en [10].
- [22] WEST: *General technique for communications protocol validation*, IBM J. Res. Develop., 22(4), pp 393-404.
- [23] AGHA: *ACTORS: A model of concurrent computation in distributed systems*. The MIT Press, Cambridge, Massachusetts, 1986.
- [24] VALMARI: *A stubborn attack on state explosion*, Proc. Computer-Aided Verification, New Brunswick, New Jersey, 1990.
- [25] GRAF (S): *Compositional minimization of finite state processes*, Proc. Computer-Aided Verification, 1990.
- [26] HOLZMANN: *On limits and possibilities of automated protocol analysis*, Protocol Specification, Testing and Verification VII. Elsevier Science Publishers B V (North Holland), 1987.
- [27] YOUNG, TAYLOR, FORESTER, BRODBECK: *Integrated cuncurrence analysis in a software development environment*, Proc. SCM SIGSOFT '89 Third Symp on Software Testing, Analysis, and Verification, Key West, Florida, 1989.
- [28] EK, ELLSBERGER: *A dynamic analysis tool for SDL*, en [10].
- [29] *Information Technology – Open Systems Interconnection – Conformance testing methodology and framework*, International Standard IS 9646, 1991.

## Reemplazada por una versión más reciente

- [30] HOGREFE: *Conformance testing based on formal methods*, Proc. FORTE 90 3<sup>rd</sup> Int Conf on Formal Description Techniques, Madrid, 1990.
- [31] BOLOGNESI, BRINKSMA: *Introduction to the ISO Specification Language LOTOS*, Computer Networks and ISDN Systems, 14, 1987, pp 25-59.
- [32] BRAEK, HASNES, HAUGEN: *Engineering real-time systems with an object-oriented methodology based on SDL*, SISU Project Report 1992, Norwegian Computing Center, P O Box 114 Blindern, N-0314 Oslo, Noruega.
- [33] HAUGEN, MÖLLER-PEDERSEN: *Tutorial on object-oriented SDL*, SISU Project Report 91002, Norwegian Computing Center, P O Box 114 Blindern, N-0314 Oslo, Noruega.

# Reemplazada por una versión más reciente

## Apéndice II

### Bibliografía para el lenguaje de especificación y descripción

(Este apéndice es parte integrante de la presente Recomendación)

(Helsinki, 1993)

- [1] BELINA (editor): *SDL Newsletter* Telia Research AB, P O Box 85, S-201 20 Malmö, Suecia.  
/\* Se publica aproximadamente una vez al año, gratuitamente, bajo los auspicios del CCITT \*/.
- [2] BELINA, HOGREFE, SARMA: *SDL with Applications from Protocol Specification*, Prentice-Hall International (Reino Unido), 270 p., (1991). Carl Hanser Verlag publicará la versión en alemán en 1992.  
/\* Libro de texto sobre el SDL, que también puede servir de referencia. Puede utilizarse como introducción a la especificación de protocolos \*/.
- [3] BELINA, HOGREFE: *The CCITT-Specification and Description Language SDL*, Computer Networks and ISDN System, 16, pp.311-341, (1988/89), North-Holland, Amsterdam.  
/\* Documento didáctico \*/.
- [4] BRAEK, HASNES, HAUGEN: *Engineering real-time systems with an object-oriented methodology based on SDL*, SISU Project Report 1992, 320 p., Norwegian Computing Center, P.O. Box 114, N-0314 Oslo 3, Noruega.  
/\* Libro de texto sobre metodología, que incluye técnicas de realización y diseño. Actualizada la versión SDL92 \*/.
- [5] Manual del CCITT – *Directrices para la aplicación de Estelle, LOTOS y SDL*, UIT, 350 p., (1988); también disponible como Documento ISO DTR 10167 (diciembre de 1989).  
/\* Contiene una colección de ejemplos, cada uno en los tres idiomas, para facilitar la evaluación subjetiva y la comparación. También contiene motivaciones para utilizar lenguajes de especificación formales \*/.
- [6] FÆRGEMAND, MARQUES (editors): *SDL '89 The language at work*, Proceedings of the Fourth SDL Forum, North-Holland, Amsterdam, 429 p., (1989).
- [7] FÆRGEMAND, REED (editors): *SDL '91 Evolving Methods*, Proceedings of the 5th SDL Forum, North-Holland, Amsterdam, 524 p., (1991).
- [8] HAUGEN, MÖLLER-PEDERSEN: *Tutorial on object-oriented SDL*, SISU Project Report 91002, Norwegian Computing Center, P.O. Box 114, N-0314 Oslo 3, Noruega.
- [9] HOGREFE, SARMA: *The application of SDL to ISDN and OSI*, A paper in the Proceeding of the Seventh International Conference on Software, Engineering for Telecommunication Switching Systems (1989).
- [10] HOGREFE: *OSI formal specification case study: the INRES protocol and service*, Technical Report, Universidad de Berna, abril de 1991.
- [11] HOGREFE: *Protocol and Service Specification with SDL: the X.25 case study*, Bericht Nr. FBI-HH-B-134/88, Universidad de Hamburgo, (1988).
- [12] SARACCO, SMITH, REED: *Telecommunications system engineering using SDL*, North-Holland, Amsterdam, 631 p., (1989).  
/\* Libro de texto sobre el SDL. También contiene directrices para el sector de aplicación determinado, con muchos ejemplos \*/.
- [13] SARACCO, TILANUS (editors): *SDL '87 State of the art and future trends*, Proceedings of the Third SDL Forum, North-Holland, Amsterdam, 463 p., (1987).
- [14] TURNER (editor): *Using Formal Description Techniques – An Introduction to Estelle, LOTOS and SDL*, será publicado por John Wiley & Sons en 1992.  
/\* Se basa en el punto 5 \*/.