UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

CCITT

COMITÉ CONSULTIVO
INTERNACIONAL
TELEGRÁFICO Y TELEFÓNICO

Z.100 Anexo D

(11/1988)

ANEXO D A LA RECOMENDACIÓN Z.100
DIRECTRICES PARA EL USUARIO DEL LED

DIRECTRICES PARA EL USUARIO DEL LED

Reedición de la Recomendación Z.100 Anexo D del CCITT publicada en el Libro Azul, Fascículo X.2 (1988)

NOTAS

1 La Recomendación Z.100 Anexo D del CCITT se publicó en el fascículo X.2 del Libro Azul. Este fichero es un extracto del Libro Azul. Aunque la presentación y disposición del texto son ligeramente diferentes de la versión del Libro Azul, el contenido del fichero es idéntico a la citada versión y los derechos de autor siguen siendo los mismos (véase a continuación).

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.

ANEXO D A LA RECOMENDACIÓN Z.100

DIRECTRICES PARA EL USUARIO DEL LED

ÍNDICE

Página

D.1 Prefacio 4

D.2 Introducción 4

D.2.1 Visión de conjunto del LED 4

D.2.1.1 El LED está basado en un modelo de máquina de estados finitos ampliada 5

D.2.2 Formas sintácticas del LED 5

D.2.3 Aplicabilidad del LED 7

D.3 Conceptos básicos del LED 7

D.3.1 Sistema 7

D.3.2 Bloques 10

D.3.3 Canales 10

D.3.4 Señales 11

D.3.5 Rutas de señales 13

D.3.6 Diagramas de sistema y de bloque 13

D.3.7 Comentarios y ampliación de texto 17

D.3.7.1 Comentarios 17

D.3.7.2 Ampliación de texto 19

D.3.8 Procesos 19

D.3.8.1 Creación de procesos 22

D.3.8.2 Estados 24

D.3.8.3 Entradas 33

D.3.8.4 Conservaciones 41

D.3.8.5 Condiciones habilitadoras y señales continuas 45

D.3.8.6 Salidas 48

D.3.8.7 Tarea 50

D.3.8.8 Decisiones 51

D.3.8.9 Uniones y conectores 54

D.3.9 Procedimientos 55

D.3.9.1 Cuerpo del procedimiento 56

D.3.9.2 Llamada de procedimiento 58

D.3.10 Tratamiento de datos 59

D.3.10.1 Declaraciones de variable 59

D.3.10.2 Variables reveladas/vistas (observadas) 61

D.3.10.3 Valores exportados/importados 62

D.3.10.4 Expresiones 63

D.3.11 Expresión del tiempo en LED 64

Página

D.3.12 Uso de calificadores ... 65
D.3.13 Sintaxis de nombres ... 66
D.4 Estructuración y refinamiento de los sistemas LED ... 68
D.4.1 Generalidades ... 68
D.4.2 Criterios de partición ... 68
D.4.3 Partición de un bloque ... 69
D.4.4 Diagrama de árbol de bloques ... 73
D.4.5 Partición de un canal ... 74
D.4.6 Representación del sistema en caso de partición ... 75
D.4.6.1 Subconjunto de partición consistente ... 78
D.4.7 Refinamiento ... 78
D.4.7.1 Subconjunto de refinamiento consistente ... 79
D.4.7.2 Transformación entre señales y subseñales ... 79
D.5 Conceptos adicionales ... 81
D.5.1 Macros ... 81
D.5.2 Sistemas genéricos ... 84
D.5.3 Servicios ... 84
D.5.3.1 Generalidades ... 84
D.5.3.2 Señales de prioridad ... 88
D.5.3.3 Transformación ... 90
D.5.4 Directrices para la representación por medio de estados y elementos pictográficos ... 95
D.5.4.1 Comentarios generales sobre representación por medio de estados ... 95
D.5.4.2 Pictograma de estado y elemento pictográfico ... 95
D.5.5 Diagramas auxiliares ... 100
D.5.5.1 Diagrama general de estados ... 100
D.5.5.2 Matriz de estados/señales ... 101
D.5.5.3 Esquema secuencial ... 102
D.6 Definición de datos en el LED ... 103
D.6.1 Directrices sobre datos en LED ... 103
D.6.1.1 Introducción general ... 103
D.6.1.2 Géneros ... 104
D.6.1.3 Operadores, literales y términos ... 104
D.6.1.4 Ecuaciones y axiomas ... 106
D.6.1.5 Más sobre ecuaciones y axiomas ... 108
D.6.2 Generadores y herencia ... 111
D.6.2.1 Generadores ... 111
D.6.2.2 Herencia ... 113
D.6.3 Consideraciones de las ecuaciones ... 115
D.6.3.1 Requisitos generales ... 115
D.6.3.2 Aplicación de funciones en constructores ... 115
D.6.3.3 Especificación del conjunto de verificación ... 117

Página

D.6.4 Características ... 118
D.6.4.1 Operadores ocultos ... 118
D.6.4.2 Ordenamiento ... 118
D.6.4.3 Género con campos ... 119
D.6.4.4 Géneros indizados ... 120
D.6.4.5 Valores por defecto de variables ... 120
D.6.4.6 Operadores activos ... 121

D.7 Directrices adicionales para dibujar y escribir ... 121

D.7.1 Directrices para el LED/GR ... 121
D.7.1.1 Generalidades ... 121
D.7.1.2 Accesos de entrada y accesos de salida ... 121
D.7.1.3 Símbolos ... 122
D.7.1.4 Plantilla ... 123

D.7.2 Directrices para el LED/PR ... 124

D.8 Documentación ... 124

D.8.1 Introducción ... 124
D.8.2 Tipos de representación de sistema ... 125
D.8.3 Estructura de documento ... 125
D.8.4 Mecanismo de referenciación ... 130
D.8.5 Clasificación de documentos ... 130
D.8.6 Combinación de LED/GR y LED/PR ... 132

D.9 Relaciones de correspondencia ... 133

D.9.1 Correspondencia entre LED y CHILL ... 133
D.9.2 Correspondencia entre GR y PR ... 137

D.10 Ejemplos de aplicación ... 137

D.10.1 Introducción ... 137
D.10.2 El concepto de servicio ... 137

D.11 Instrumentos para el LED ... 153

D.11.1 Introducción ... 153
D.11.2 Clases de instrumentos ... 153
D.11.3 Entrada de documentos ... 153
D.11.4 Verificación de documentos ... 154
D.11.5 Reproducción de documentos ... 154
D.11.6 Generación de documentos ... 155
D.11.7 Modelado y análisis de sistemas ... 155
D.11.8 Generación de códigos ... 156
D.11.9 Capacitación ... 156

D.1 *Prefacio*

El Lenguaje de Especificación y de Descripción del CCITT, conocido por la sigla LED, fue definido por primera vez en las Recomendaciones Z.101 a Z.103 en 1976 (Libro Naranja, Tomo VI.4), ampliado más tarde en las Recomendaciones Z.101 a Z.104 en 1980 (Libro Amarillo) y ampliado de nuevo y reestructurado en las Recomendaciones Z.100 a Z.104 en 1984 (Libro Rojo). En el periodo de estudios 1985-1988 el lenguaje ha sido ampliado y armonizado aún más, las Recomendaciones existentes han sido fusionadas en una sola y se ha suministrado la definición matemática.

Se hacen necesarias directrices para los usuarios a fin de facilitar la utilización del LED en su aplicación a una amplia gama de sistemas de telecomunicaciones. El propósito de las directrices es asistir a los usuarios en la comprensión de la Recomendación sobre el LED y su aplicación a los diferentes campos.

El LED está siendo ampliamente utilizado por el CCITT y sus organizaciones miembros, y la gama de aplicaciones del LED continúa en aumento. Estas directrices para los usuarios están destinadas a ayudar a las personas que se plantean el empleo del LED o están comenzando a utilizarlo, suplementando las Recomendaciones sobre el LED con consejos útiles y ejemplos valiosos. Debe tenerse en cuenta que habrá cierta superposición entre las directrices para los usuarios y las Recomendaciones; se estima que esto es conveniente a fin de que las directrices para los usuarios sean autónomas y de fácil lectura. No obstante, el documento esencial es la Recomendación.

D.2 *Introducción*

D.2.1 *Visión de conjunto del LED*

El LED puede utilizarse para la especificación del comportamiento requerido de un sistema y para describir el comportamiento real de un sistema. El LED se ha diseñado teniendo presente la especificación del comportamiento de los sistemas de conmutación para telecomunicaciones, pero también puede utilizarse para modelar otras aplicaciones. En realidad, el LED se presta para todos los sistemas cuyo comportamiento pueda modelarse efectivamente por medio de máquinas de estado finito ampliadas (§ D.2.1.1) y cuando el acento deba recaer en particular en aspectos relativos a la interacción.

El LED puede ser también la base de una metodología de documentación que represente completamente la especificación o descripción de un sistema. En este contexto, el significado de especificación y descripción guarda relación con su uso en el ciclo de vida de un sistema. Ambas definen las propiedades funcionales de un sistema de una manera abstracta. La descripción comprenderá habitualmente algunos aspectos dependientes del diseño (por ejemplo, el tratamiento de errores) y será más completa en materia de detalles funcionales. Ambas deben estar relacionadas con el diseño del sistema concreto de una manera uniforme, y servir como documentación posteriormente.

El LED puede utilizarse para representar, con distintos niveles de detalle, las propiedades funcionales de un sistema, una función o una facilidad. Las propiedades funcionales consisten en algunas propiedades estructurales (diagramas de interacción de bloques) así como de comportamiento. Por «comportamiento» se entiende el modo de reaccionar frente a las señales recibidas (entradas), esto es, mediante la ejecución de acciones, por ejemplo, enviar señales (salidas), hacer preguntas (decisiones) y realizar tareas.

Las especificaciones pueden ser muy amplias y generales cuando una administración desea explorar las posibilidades de actualizar un sistema con nuevas características, nuevos servicios, nueva tecnología, etc., permitiendo al proveedor que ofrezca una amplia gama de soluciones en los diseños. Este tipo de especificación a menudo no es muy detallado. El otro extremo es una especificación en la que una administración solicita la sustitución de una central existente o una adición a la misma. Esa especificación tendrá probablemente un elevado nivel de detalle, debido a la necesidad de una especificación muy pormenorizada de los interfaces.

Una especificación y una descripción pueden ser idénticas. En todo caso, en un nuevo desarrollo es preferible que el diseño se derive de la especificación a fin de garantizar su cumplimiento.

Por lo general, los proveedores escriben las descripciones en respuesta a una especificación (aunque pueden escribirse para describir sistemas que el proveedor desea vender). Una descripción tendrá habitualmente un nivel de detalle superior al de una especificación, debido a la necesidad de describir el comportamiento pormenorizado del sistema. Para facilitar la lectura, en lo sucesivo la representación LED se denominará simplemente especificación.

Cabe observar también que el LED ofrece medios para describir un sistema con diversos grados de formalidad.

Primeramente, es posible describir un sistema utilizando las construcciones LED con un lenguaje natural asociado. La especificación resultante es significativa únicamente para un lector que conozca el contexto, pero no para una máquina. Son limitadísimas las verificaciones que pueden efectuarse en forma automática.

En segundo lugar, es posible asociar a las construcciones LED declaraciones formales consistentes en elementos de tipos definidos y operadores relativos a esos elementos. Las propiedades de estos elementos no quedan especificadas: un ejemplo es «conectar A-B», en donde A y B son del tipo abonado, y conectar es una operación permitida para este tipo. La especificación resultante es significativa para los lectores que conocen el significado de los operadores utilizados. Una máquina puede comprender la descripción hasta cierto nivel y puede efectuar verificaciones en base a ella, pero no puede hacer verificaciones completas ni «implementar» el sistema porque las propiedades de los operadores no están especificadas de manera completa.

En tercer lugar, es posible indicar también todas las propiedades de todos los operadores. En este último caso la especificación es completamente formal y una máquina puede hacer todas las verificaciones e implementar conceptualmente el sistema descrito.

Según cual sea la meta perseguida, las especificaciones pueden adaptarse a las necesidades del usuario utilizando estos diferentes niveles de formalismo. Como es natural, cuanto más formal es la especificación, tanto más engorrosa es su lectura por el ser humano.

En lo sucesivo el término especificación será utilizado tanto para la representación del comportamiento requerido como la del real.

D.2.1.1 *El LED está basado en un modelo de máquina de estados finitos ampliada*

En la aplicación del LED, el sistema por especificar se representa mediante un número de máquinas abstractas interconectadas. Una especificación completa requiere:

1) la especificación de la estructura del sistema en términos de las máquinas y sus interconexiones;

2) el comportamiento dinámico de cada máquina en términos de sus interacciones con las otras máquinas;

3) las operaciones con respecto a los datos asociados a las interacciones.

El comportamiento dinámico se describe con ayuda de modelos que definen los mecanismos de funcionamiento de estas máquinas abstractas y la comunicación entre máquinas. La máquina abstracta utilizada en el LED es una extensión de la máquina de estados finitos (MEF) determinista. La MEF tiene una memoria de estado interno finita y opera con un conjunto discreto y finito de entradas y salidas. Para cada combinación de entrada y estado, la memoria define una salida y el estado siguiente. Habitualmente se considera que las transiciones de un estado a otro toman un tiempo cero.

Una limitación de la MEF radica en el hecho de que toda la información que hace falta almacenar debe estar representada en forma de estados explícitos. Aunque es posible representar de esta manera la mayoría de los sistemas, no siempre resulta práctico. Puede haber muchos valores por almacenar que son significativos para el comportamiento futuro pero que no contribuyen mucho a la comprensión global del sistema. Esta información no debe formar parte del espacio de estado explícito, pues abarrotaría la presentación. Para aplicaciones de esta índole se puede ampliar la MEF con un almacenamiento auxiliar y operaciones auxiliares respecto a ese almacenamiento. La información de dirección y los números de secuencias son ejemplos de la información que se presta para almacenamiento en memoria auxiliar.

Las Recomendaciones relativas al LED definen dos operaciones auxiliares que pueden incluirse en las transiciones de la máquina de estados finitos ampliada (MEFA), a saber, decisiones y tareas. Las «decisiones» inspeccionan parámetros asociados a entradas e información en memoria auxiliar cuando tal información es importante para la secuencia de la máquina principal. Las «tareas» realizan funciones tales como el conteo, la operación respecto a la memoria auxiliar y la manipulación de parámetros de entrada y salida.

En el LED, las interacciones entre las máquinas están representadas por señales, o sea que la MEFA recibe señales como entrada y genera señales como salida. Las señales se componen de un identificador de señal único y, facultativamente, un conjunto de parámetros. El LED prevé la posibilidad de un tiempo de transición distinto de cero, y define un mecanismo de fila de espera conceptual basada en el método de «el primero que entra es el primero que sale» para las señales que llegan a una máquina mientras ésta ejecuta una transición. Las señales se consideran una a la vez, por orden de llegada.

D.2.2 *Formas sintácticas del LED*

El LED es un lenguaje que tiene dos formas diferentes, basadas ambas en el mismo modelo semántico. Una, llamada LED/GR (representación gráfica LED), está basada en un conjunto de símbolos gráficos normalizados. La otra, llamada LED/PR (representación por frase textual LED) está fundada en sentencias a modo de programa. Ambas representan los mismos conceptos LED.

Un lenguaje gráfico tiene la ventaja de mostrar claramente la estructura de un sistema y permitir que los seres humanos puedan visualizar fácilmente el flujo de control. La representación por frase textual se presta mejor para su utilización por máquinas.

Como herramienta de diseño el LED debe tener una forma que permita al usuario expresar de manera concisa sus ideas. El LED/GR permite esto y está más en consonancia con la representación tradicional de máquinas de estados finitos ampliadas.

El LED/GR es la forma original del LED. Fue concebido durante el periodo de 1973 a 1976 y apareció por primera vez en la versión de 1976 de las Recomendaciones de la serie Z.100.

El LED/GR se derivó de los lenguajes gráficos desarrollados por diferentes organizaciones para su uso particular.

La representación por frase textual LED, el LED/PR, fue ideada durante el periodo de estudios 1977-1980 pero se necesitaron algunos perfeccionamientos antes de que pudiera recomendarse. Esos perfeccionamientos se consiguieron en el periodo de estudios siguiente y desde 1984 el LED/PR ha sido una de las sintaxis concretas recomendadas del LED.

Al comienzo, el objetivo era utilizar el LED/PR como una manera fácil de introducir documentos LED en una máquina, ya que el GR es más difícil de introducir. (Se necesitan periféricos gráficos para su tratamiento.) Por este motivo, se había puesto énfasis en una correspondencia biunívoca entre PR y GR. La evolución de los terminales gráficos (aumento de capacidad y disminución de coste) ha llevado desde entonces a aceptar el GR como apto para la entrada en una máquina. Esto no disminuye la importancia y el uso del PR, pues a algunos usuarios les gusta más, sobre todo a los que trabajan con lenguajes de programación.

Esta evolución ha conducido a una correlación menos estricta entre el GR y el PR; sin embargo, todavía podemos hacer corresponder (fácilmente) el uno con el otro, pero cada forma tiene sus peculiaridades. A primera vista, el PR se asemeja mucho a un lenguaje de programación (véase la figura D-2.2.1).

```
—
—
STATE esperar_descolgado;
INPUT descolgado;
TASK 'activar tasación';
TASK 'conectar';
OUTPUT reponer_temporizador;
NEXTSTATE conversación;
—
—
```

FIGURA D-2.2.1

Ejemplo de LED/PR

De hecho, todo depende de lo que caracteriza a un texto como lenguaje de programación.

Si suponemos que un programa se define como «información interpretable por una máquina», entonces son «programas» no solamente las especificaciones PR sino también las especificaciones GR.

Existen, sin embargo, algunas diferencias entre una especificación LED y un programa real. Primeramente, no es esencial que una especificación LED deba ser ejecutable por una máquina (aunque ello no está prohibido); lo que es esencial es su capacidad de transportar información precisa de un ser humano a otro ser humano.

Si vemos una especificación LED como un programa, lo que pudiera considerarse como «especificación LED errónea» (a causa de un texto informal incompleto) podría ser un LED perfectamente válido al considerarlo como una representación de los requisitos funcionales de un sistema.

Otra diferencia está en el «estilo» de una especificación LED, en comparación con la representación habitual de un programa.

Dado que el propósito del LED es ayudar a la comunicación entre seres humanos, debe tenerse especial cuidado en permitir diferentes disposiciones a fin de que la disposición LED pueda usarse para guiar al lector de modo que éste concentre su atención en ciertos aspectos que se consideran más importantes que otros. Esto, naturalmente, no tiene importancia alguna en el caso de un programa, que se supone va a ser interpretado por una máquina. La máquina no «concentra su atención» en un aspecto particular cualquiera sino que tiene que considerar el todo igualmente; la máquina tampoco trata de «comprender» el programa.

Por su semejanza con un programa, el PR es preferido por algunos programadores que probablemente utilizan CHILL para implementar los requisitos. Por lo tanto, existe una gran inclinación a encontrar una correspondencia de uno a uno de PR a CHILL, de modo que los requisitos expresados en PR puedan ser automáticamente transformados en código CHILL. La situación inversa es también de interés dado que permitiría la derivación de una especificación PR a partir de un programa CHILL.

En el § D.9 se ilustran posibles modos de hacer corresponder LED con CHILL.

D.2.3 *Aplicabilidad del LED*

La figura D-2.3.1 muestra una gama de posibles utilizaciones del LED en el campo de la compra y el suministro de sistemas de conmutación para telecomunicaciones.

En esa figura, los rectángulos representan grupos funcionales típicos cuyos nombres precisos pueden variar de una organización a otra, pero cuyas actividades serían típicas de muchas administraciones y fabricantes. Cada flecha (línea de flujo) representa un conjunto de documentos que pasan de un grupo funcional a otro; el LED puede utilizarse como parte de cada uno de esos conjuntos de documentos. Esta figura tiene carácter puramente ilustrativo y no se pretende que sea definitiva ni exhaustiva.

Los sectores de aplicación son aquéllos modelados efectivamente por máquinas de estado finito ampliadas en comunicación, por ejemplo, conmutación telefónica, télex y de datos, sistemas de señalización (por ejemplo, sistema de señalización N.° 7), interfuncionamiento entre sistemas de señalización y protocolos de datos, interfaces de usuario (LHM).

Al considerar en particular las áreas de conmutación SPC, figuran las áreas siguientes que pueden ser documentadas utilizando el LED: tratamiento de las comunicaciones (por ejemplo, tratamiento de llamadas, encaminamiento, señalización, tasación, etc.), mantenimiento y reparación de averías (por ejemplo, alarma, liberación automática en caso de fallo, configuración del sistema, pruebas periódicas, etc.), control del sistema (por ejemplo, control de sobrecarga) e interfaces hombre-máquina. En el § D.10 pueden verse ejemplos de aplicación del LED.

La especificación de protocolos mediante el LED se considera en las Recomendaciones de la serie X del CCITT.

D.3 *Conceptos básicos del LED*

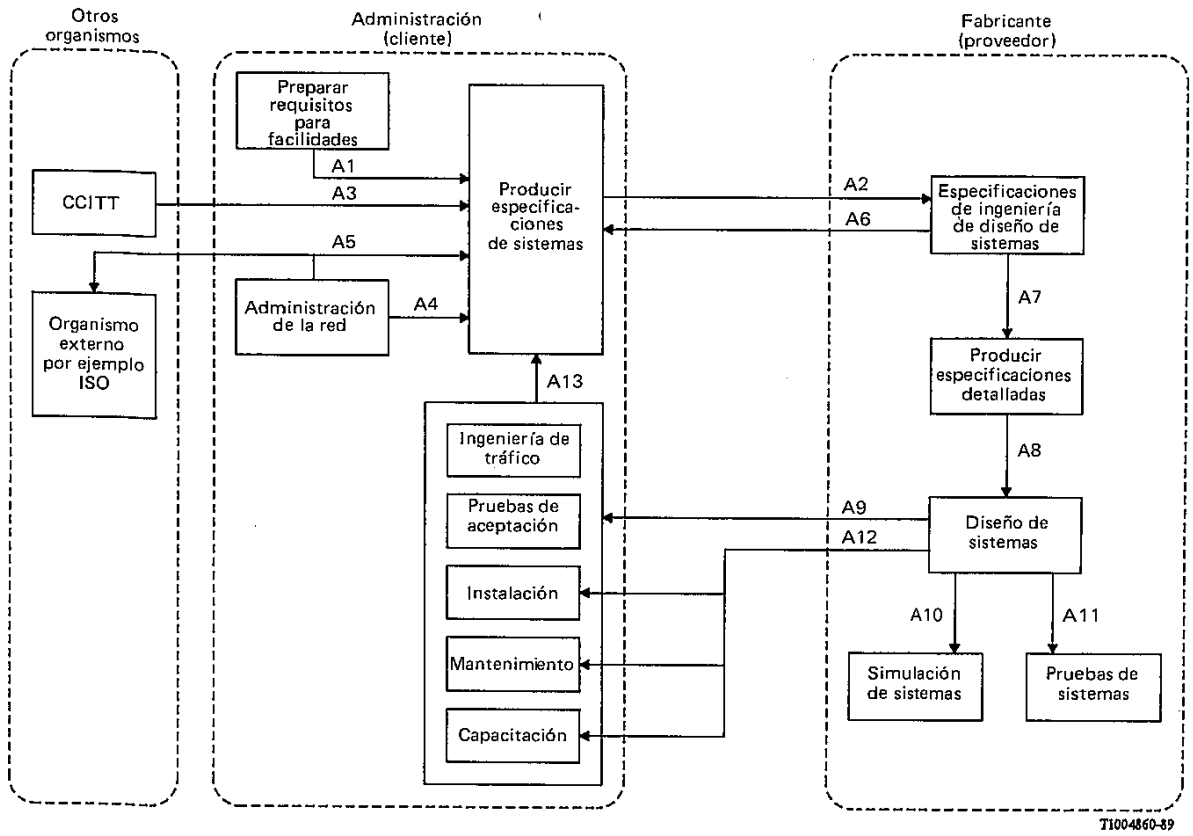
D.3.1 *Sistema*

Como ya se ha dicho, el LED modela sistemas. Un sistema es, pues, lo que la especificación LED define. Como tal, un sistema LED puede modelar una parte de un sistema (o central) telefónico o una red completa de sistemas telefónicos o partes de una multiciplidad de centrales telefónicas (por ejemplo, los controladores interurbanos de ambos extremos de un enlace interurbano). Lo esencial es que, desde el punto de vista del LED, el sistema LED contiene todo lo que la especificación trata de definir. El entorno queda fuera de la especificación y no está definido en LED.

El sistema se relaciona con el entorno a través de canales. En teoría hace falta un solo canal bidireccional para el interfaz con el entorno. En la práctica suelen definirse canales para cada interfaz lógico con el entorno.

Cada sistema consta de cierto número de bloques conectados entre sí por canales. Cada bloque del sistema es independiente de cualquier otro bloque. Cada bloque puede contener uno o más procesos que describen el comportamiento del bloque. El único medio de comunicación entre procesos situados en dos bloques diferentes es el envío de señales que son transportadas por los canales. Los criterios que llevan a determinada división del sistema en bloques pueden ser: definir partes de un tamaño manejable, crear una correspondencia con la división real en soporte lógico/soporte físico, seguir subdivisiones funcionales naturales, minimizar las interacciones, y otros.

Para grandes sistemas LED hay algunas construcciones LED que admiten la especificación de subestructuras de las partes de un sistema, de modo que, partiendo de una amplia visión general del sistema, pueda proporcionarse cada vez más detalles. En este caso, se dice que el sistema está representado a diferentes niveles de detalle. Estas construcciones se explican en el § D.4.



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; incluye una descripción del entorno del sistema
A3 Recomendaciones y directrices del CCITT
A4 Contribuciones a la especificación del sistema que indican los requisitos de administración de la red y operacionales
A5 Otras Recomendaciones 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 descripción de un sistema y de 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 operación
A13 Contribuciones sobre la especificación del sistema preparadas por grupos funcionales especializados dentro de la Administración

Nota 1 – Es posible la iteración en todos los niveles.

Nota 2 – En ciertas circunstancias, la documentación LED que en este diagrama se indica como interna de una organización, por ejemplo, A1, A7, A8, podría suministrarse a otra organización.

FIGURA D-2.3.1

Ámbito general del uso del LED

En el primer nivel de detalle, la especificación LED de un sistema describe la estructura del sistema e incluye los siguientes puntos que serán explicados en los párrafos siguientes:

- nombre del sistema;
- definiciones de señales: la especificación de los tipos de señales intercambiadas entre los bloques del sistema o entre los bloques y el entorno. Incluye la especificación de los tipos de valores transportados por las señales (lista de géneros);
- definiciones de listas de señales: la especificación de identificadores que comprenden varias señales y/u otras listas de señales. Dichos identificadores pueden utilizarse para ahorrar espacio y permitir una especificación más clara;
- definiciones de canales: la especificación de los canales que conectan los bloques del sistema entre sí y con el entorno. Una definición de canal incluye la especificación de los identificadores de señales transportadas por ese canal;

– definiciones de datos: la especificación de los neotipos definidos por el usuario, sintipos y generadores visibles en todos los bloques;

– definiciones de bloques: la especificación de los bloques en los cuales está dividido el sistema;

– definiciones de macros: en el § D.5.1 se dan directrices sobre el uso de macros.

Según la Recomendación sobre LED, existen tipos de datos predefinidos que están a la disposición por cada sistema. No necesitan definirse y pueden utilizarse mediante sus nombres predefinidos, es decir: INTEGER, REAL, CHARACTER, STRING, CHARSTRING, BOOLEAN, PID, TIME, DURATION. Los tipos de datos predefinidos son visibles en cualquier nivel de la definición del sistema; pueden considerarse implícitamente definidos en una biblioteca del sistema accesible en cualquier punto de la especificación.

Los nombres de géneros utilizados en las definiciones de señales a nivel de sistema deben ser introducidos por definiciones parciales de tipo visibles a nivel de sistema, es decir, tipos de datos predefinidos, neotipos definidos por el usuario o sintipos definidos a este nivel.

En los § D.3.10 y D.6 pueden encontrarse explicaciones adicionales sobre el uso de los tipos de datos.

El LED/PR para una definición de sistema consiste en un conjunto de sentencias, cada una terminada por un «;». La definición de una estructura de sistema comienza con la sentencia «SYSTEM nombre;» y termina con la sentencia «ENDSYSTEM nombre;». El nombre en la sentencia de cierre es facultativo pero, si se incluye, debe ser el mismo nombre que aparece después de la palabra clave SYSTEM. Se sugiere incluir siempre el nombre en la sentencia de cierre, con lo que se facilita la lectura del documento.

El esquema del LED/PR de una definición de estructura de sistema se muestra en la figura D-3.1.1.

```
SYSTEM . . .;
   . . . definiciones de señales . . .
   . . . definiciones de listas de señales . . .
   . . . definiciones de canales . . .
   . . . definiciones de datos . . .
   . . . definiciones de bloques . . .
   . . . definiciones de macros . . .
ENDSYSTEM . . .;
```

FIGURA D-3.1.1

Esquema del LED/PR de la definición de estructura de sistema

Con el fin de tener una representación más clara y más simple de la estructura del sistema y también para permitir una especificación de sistema de tipo descendente (o «de arriba a abajo»), se suministra en el LED un mecanismo de referenciación general. A este nivel el mecanismo de referenciación puede aplicarse a definiciones de bloques. Esta prestación del lenguaje permite al usuario especificar solamente el nombre de bloque dentro de la definición de la estructura de sistema; la definición efectiva del bloque puede darse separadamente. (Véase la figura D-3.1.2.)

```
SYSTEM s;
   . . .
   BLOCK b1 REFERENCED;
   BLOCK b2 REFERENCED;
   . . .
ENDSYSTEM s;
```

FIGURA D-3.1.2

Ejemplo de referenciación de definición de bloque

El mecanismo de referenciación se usa especialmente en el LED/GR ya que la mayor parte de los diagramas tienen que estar contenidos en una sola página y a menudo no hay espacio suficiente para especificaciones gráficas anidadas.

En el § D.3.6 pueden encontrarse ejemplos del LED/GR para una definición de sistema.

D.3.2 *Bloques*

Dentro de un bloque, los procesos pueden comunicar entre sí mediante señales o mediante valores compartidos. El bloque constituye así, no sólo un mecanismo conveniente para agrupar procesos, sino también una frontera para la visibilidad de los datos. Por ello al definir bloques hay que cerciorarse de que la agrupación de procesos dentro de un bloque es una agrupación funcional razonable. Las más de las veces resulta útil descomponer primero el sistema (o bloque) en unidades funcionales y definir luego los procesos que van en el bloque.

Dentro de un bloque es posible (facultativamente) definir trayectos de comunicación entre los procesos o entre los procesos y el entorno del bloque (es decir, la frontera del bloque). Esos trayectos de comunicación se llaman rutas de señales.

Para grandes sistemas LED es posible describir la subestructura de un bloque en términos de otros bloques y canales, como si el propio bloque fuese un sistema. En el § D.4 se explica tal mecanismo.

La definición de la estructura de un bloque puede incluir los puntos siguientes:

– nombre del bloque;

– definiciones de señales: la especificación de los tipos de señales intercambiadas internamente en el bloque. Este punto incluye la especificación de los tipos de valores transportados por las señales (lista de géneros);

– definiciones de listas de señales: las especificaciones de los identificadores correspondientes a listas de señales y/u otros identificadores de listas de señales. Esos identificadores, que agrupan varias señales, pueden utilizarse para ahorrar espacio y permitir una especificación más clara;

– definiciones de rutas de señales: las especificaciones de los trayectos de comunicación que conectan los procesos del bloque unos con otros y con el entorno del bloque. Una definición de ruta de señales incluye la especificación de los identificadores de señales transportados por esa ruta de señales;

– conexiones de canal a ruta: las especificaciones de las conexiones entre los canales externos al bloque y las rutas de señales internas del bloque;

– definiciones de procesos: la especificación de los tipos de procesos que describen el comportamiento del bloque. Si el bloque no está descrito en términos de su subestructura, habrá por lo menos una definición de tipo de proceso dentro del bloque. En forma similar al caso de los bloques indicado en el § D.3.1, se suministra un mecanismo de referenciación para la definición del proceso;

– definiciones de datos: la especificación de los neotipos definidos por el usuario, sintipos y generadores visibles en todos los procesos definidos del bloque y/o en la subestructura del bloque;

– definiciones de macros: en el § D.5.1 se dan directrices sobre el uso de macros.

Si hay una subestructura del bloque, algunos de los puntos arriba indicados son facultativos. (Véase el § D.4 para explicaciones sobre la estructuración).

Los siguientes tipos son visibles en un bloque:

– tipos de datos predefinidos,

– tipos de datos definidos por el usuario, definidos en el bloque mismo,

– tipos de datos definidos por el usuario visibles en el bloque progenitor (en caso de partición de bloque).

En el LED/PR las palabras clave BLOCK y ENDBLOCK se usan para delimitar una definición de bloque. En el § D.3.6 pueden encontrarse ejemplos del LED/GR para una definición de bloque.

D.3.3 *Canales*

Los canales son el medio de comunicación entre diferentes bloques del sistema o entre los bloques y el entorno. Un canal puede conectar un bloque a otro bloque o un bloque al entorno en un solo sentido (canal unidireccional) o en ambos sentidos (canal bidireccional). Normalmente, un canal es una entidad funcional que puede utilizarse para designar trayectos específicos de comunicación. En realidad, mediante la partición de canales (descrita en el § D.4.5), es posible especificar formalmente el comportamiento de cada canal.

Para cada sentido designado, o trayecto de comunicación, la especificación de canal contiene una lista de todos los identificadores de señales que puede transportar el canal en ese sentido. Esta lista de señales se suministra como un medio de asegurar que cada señal enviada por un proceso desde un extremo del canal pueda ser recibida por un proceso en el bloque situado al otro extremo del canal. Así, la especificación de canal va a formar parte de la especificación de interfaz para cada bloque. En grandes proyectos multipersonales, un pronto acuerdo sobre las señales que serán

transportadas por un canal, y sobre la especificación de esas señales, reducirá la probabilidad de que dos procesos no puedan comunicar como se deseaba.

La definición de un canal incluye los siguientes puntos:

– nombre del canal;

– uno o dos trayectos de comunicación: un trayecto de comunicación especifica el origen y el destino de una lista de señales. En este contexto pueden utilizarse identificadores de bloque o la palabra clave «ENV» (para el entorno);

– una o dos listas de señales: para cada trayecto de comunicación debe especificarse una lista de señales transportadas en ese sentido. La lista puede incluir identificadores de señales o, también, identificadores de otras listas;

– una definición facultativa de subestructura de canal (o la referencia a ella): véase el § D.4.5.

En LED/PR una definición de canal va encerrada entre las palabras clave CHANNEL y ENDCHANNEL. Las palabras clave FROM y TO se utilizan para expresar los trayectos de comunicación y la palabra clave WITH para expresar listas de señales. En la figura D-3.3.1 se presenta un ejemplo de definiciones de canal en LED/PR.

En LED/GR la definición de canal se representa por medio de una línea que conecta las dos partes que intervienen en la comunicación. El nombre de canal debe estar más cerca de la línea que cualquier otro símbolo. Los trayectos se representan por medio de flechas y las listas de señales deben colocarse entre corchetes, como se muestra en los ejemplos de la figura D-3.3.2. Las flechas no deben colocarse en ninguno de los dos puntos extremos de la línea, para no confundir canales con rutas de señales (véase § D.3.5).

En un canal bidireccional cada una de las dos listas de señales debe estar lo más cerca posible de la flecha correspondiente.

D.3.4 *Señales*

Las señales se pueden definir a nivel de sistema, de bloque, o en la parte interna de una definición de proceso. Las señales definidas a cierto nivel pueden utilizarse a ese nivel o también a niveles más bajos; sin embargo, con el objeto de simplificar cada nivel, se sugiere definir las señales tan localmente como sea posible. Las señales definidas dentro de una definición de proceso pueden intercambiarse entre instancias del mismo tipo de proceso (§ D.3.8) o entre servicios en el proceso (§ D.5.3).

La definición de una señal incluye los puntos siguientes:

– nombre de la señal;

– lista de géneros (facultativa): representa la lista de los tipos de valores transportados por esa señal;

– refinamiento de la señal (facultativo): se explica en el § D.4.7.

Una definición de señal en LED/PR se especifica con la palabra clave SIGNAL. Dentro de la construcción se pueden definir varias señales (no refinadas), proporcionando el nombre de la señal y la lista de géneros. En la figura D-3.4.1 se dan ejemplos de definiciones de señal en LED/PR.

```
SYSTEM tal_y_tal;
   ....
   SIGNALLIST id_lista_s_1 = señ_b, señ_c, señ_d;
   ...
   CHANNEL c1
      FROM bloque_a TO bloque_b WITH señal_1,señal_2,señal_3;
   ENDCHANNEL c1;

   CHANNEL can_2
      FROM ENV TO bloque_c WITH señ_ext_1,señ_ext_2;
      FROM bloque_c TO ENV WITH señ_int_1;
   ENDCHANNEL can_2;

   CHANNEL nomb.can.3
      FROM bx TO by WITH señ_a,(id_lista_s_1),señ_e;
   ENDCHANNEL nomb.can.3;
   ...
ENDSYSTEM tal_y_tal;
```

FIGURA D-3.3.1

Ejemplo de definiciones de canal en LED/PR

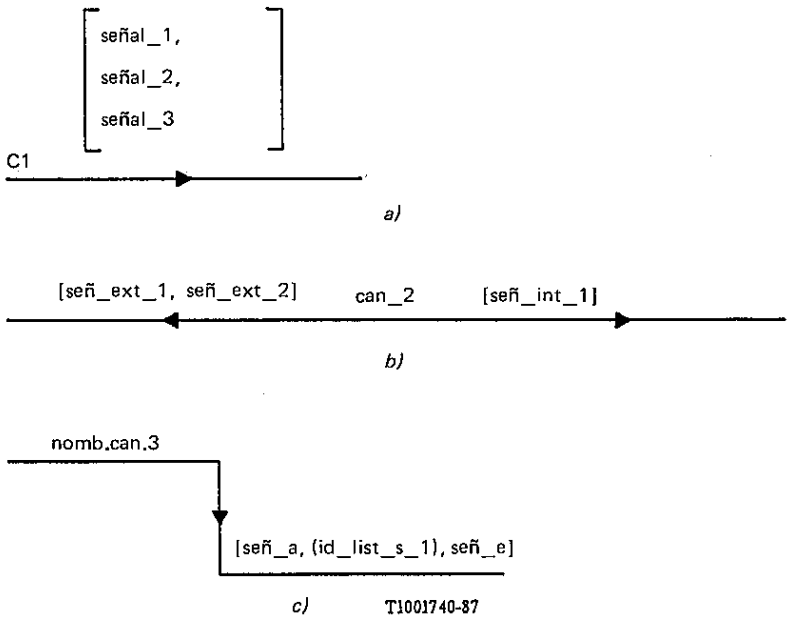


FIGURA D-3.3.2

Ejemplo de definiciones de canal en LED/GR

```
SIGNAL s1,s2,s3;

SIGNAL señ_a(género1,género2),
       señ_b(género3,género4);
```

FIGURA D-3.4.1

Ejemplos de definiciones de señal en LED/PR

En LED/GR una definición de señal se especifica encerrando sentencias lineales dentro de un símbolo de texto según se indica en la figura D-3.4.2.

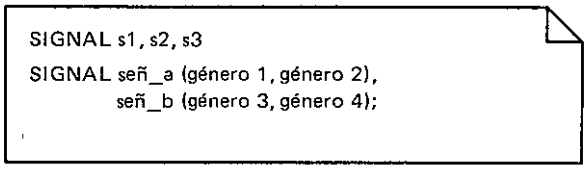


FIGURA D-3.4.2

Ejemplos de definiciones de señal en LED/GR

D.3.5 *Rutas de señales*

Las rutas de señales se utilizan para expresar trayectos de comunicación en forma similar a los canales. Pueden utilizarse a nivel de bloque y también a nivel de proceso. Al igual que los canales, las rutas de señales pueden ser unidireccionales o bidireccionales pero no pueden subdividirse.

A nivel de bloque ellas representan un medio de comunicación entre los procesos de un bloque o entre los procesos y el entorno del bloque, es decir un canal que va hacia dicho bloque o que sale de él.

A nivel de proceso pueden utilizarse rutas de señales cuando el proceso está subestructurado en servicios (véase § D.5.3). En este caso conectan servicios entre sí o a las rutas de señales del proceso.

Cuando se entrega una señal a una ruta de señales que conduce a la frontera del bloque, la señal se da al canal conectado a la ruta de señales. Cuando una señal llega al bloque desde un canal y el canal está conectado a una o más rutas de señales, la señal se entrega a la ruta de señales que es capaz de transportar esa señal.

En LED/PR la definición de una ruta de señales comienza con la palabra clave SIGNALROUTE. La sintaxis de los trayectos de comunicación y las listas de señales es la misma que en el caso de los canales.

En LED/GR la única diferencia entre una ruta de señales y un canal es que para las rutas de señales las flechas deben estar en los extremos de la línea; cerca de cada flecha debe aparecer la correspondiente lista de señales. En las figuras D-3.6.3 y D-3.6.5, en los párrafos siguientes, pueden encontrarse ejemplos de rutas de señales.

D.3.6 *Diagramas de sistema y de bloque*

En LED/GR una definición de sistema se representa por medio de un conjunto de diagramas. La estructura de un sistema en términos de canales y bloques se representa por medio de un diagrama de sistema.

El diagrama de sistema se compone de:

- el símbolo de casilla: un símbolo de forma rectangular que encierra todos los otros símbolos. Representa la frontera del sistema: fuera de esta casilla está el entorno del sistema;
- el encabezamiento del sistema: la palabra clave SYSTEM seguida por el nombre del sistema (en la esquina superior izquierda de la casilla);
- numeración facultativa de las páginas (en la esquina superior derecha de la casilla);
- símbolos de texto: estos símbolos pueden contener sentencias lineales. Se usan generalmente para mostrar en los diagramas las definiciones de señales, listas de señales y datos;
- la zona de interacción de bloques: ésta incluye la especificación de los bloques del sistema, los canales y las listas de señales transportadas por los canales;
- diagramas de macros: en el § D.5.1 se dan directrices sobre el uso de macros.

En el diagrama de sistema la especificación de un bloque puede adoptar una de las dos formas siguientes:

- una referencia de bloque: un símbolo de bloque que contiene solamente el nombre del bloque;
- un diagrama de bloque: una casilla que contiene la especificación de la estructura del bloque en términos de sus procesos y sus interacciones. Si el bloque está subestructurado en subbloques, debe proporcionarse dentro de la casilla del bloque la especificación de la subestructrura o la referencia (§ D.4.3).

La forma de los símbolos utilizados en los diagramas de sistema y de bloque puede encontrarse en el resumen de la sintaxis LED/GR. Para las dimensiones de los símbolos, consúltese el § D.7.1.4.

En la figura D-3.6.1 hay un ejemplo de un diagrama de sistema para el sistema «s». En este ejemplo el sistema s está dividido en dos bloques B1 y B2, conectados entre sí y al entorno por los canales C1, C2, C3 y C4. Sólo se da una referencia en este ejemplo para los bloques B1 y B2. En los párrafos siguientes se dan mas explicaciones sobre símbolos de canales y listas de señales.

En la figura D-3.6.2 se muestra el mismo ejemplo en LED/PR.

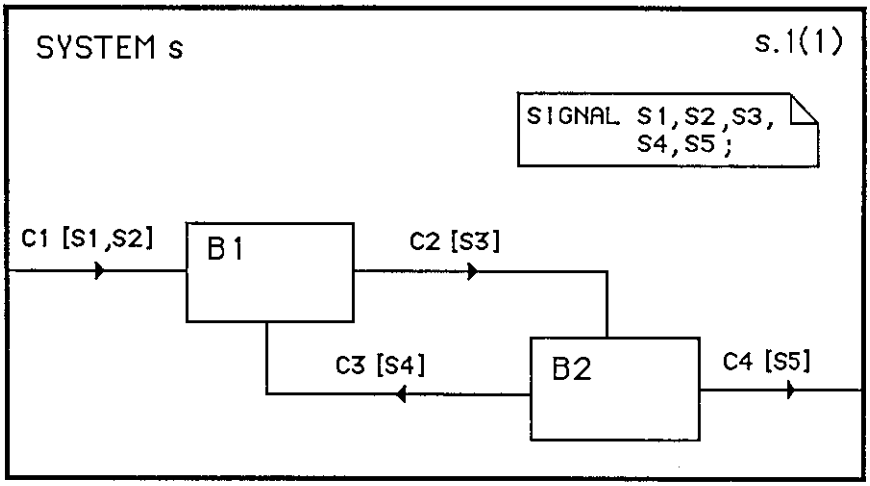


FIGURA D-3.6.1

Ejemplo de un diagrama de sistema

```
SYSTEM s;
   SIGNAL S1,S2,S3,S4,S5;
   CHANNEL C1 FROM ENV TO B1 WITH S1,S2;
   ENDCHANNEL C1;
   CHANNEL C2 FROM B1 TO B2 WITH S3;
   ENDCHANNEL C2;
   CHANNEL C3 FROM B2 TO B1 WITH S4;
   ENDCHANNEL C3;
   CHANNEL C4 FROM B2 TO ENV WITH S5;
   ENDCHANNEL C4;
   BLOCK B1 REFERENCED;
   BLOCK B2 REFERENCED;
ENDSYSTEM s;
```

FIGURA D-3.6.2

Ejemplo de una definición de estructura de sistema en LED/PR

La estructura de un bloque en términos de procesos y rutas de señales se representa por medio del diagrama de bloques en LED/GR.

El diagrama de bloques está compuesto de:

– el símbolo de casilla: un símbolo de forma rectangular que encierra todos los otros símbolos. Representa la frontera del bloque: fuera de esta casilla está el entorno del bloque;

– el encabezamiento del bloque: la palabra clave BLOCK seguida por el nombre del bloque (en la esquina superior izquierda de la casilla);

– numeración facultativa de las páginas (en la esquina superior derecha de la casilla);

– símbolo de texto: esos símbolos pueden incluir sentencias lineales. Por lo general se usan para mostrar las definiciones de señales, listas de señales y datos;

– la zona de interacción de procesos: incluye la especificación de los procesos del bloque y posiblemente las rutas de señales y las listas de señales transportadas por las rutas de señales. En esta zona es también posible mostrar procesos que crean otros procesos; esta característica se describe en el § D.3.8.1;

– identificadores de canal: si el diagrama muestra rutas de señales que van al entorno del bloque o vienen de él, los identificadores de los canales conectados a esas rutas de señales deben especificarse fuera de la casilla, en correspondencia con las líneas de rutas de señales;

– diagramas de macro: en el § D.5.1 se dan directrices sobre el uso de macros.

En el diagrama de bloques la especificación de un proceso puede ser alternativamente:

– una referencia de proceso: un símbolo de proceso que contiene el nombre del proceso y, facultativamente, la especificación de instancias de proceso. Esa especificación de instancias de proceso consiste en un par de números enteros separados por una coma y encerrados entre paréntesis (véase el § D.3.8);

– un diagrama de proceso: una casilla que contiene un gráfico de símbolos conectados que describen el comportamiento del proceso en términos de estados, entradas, salidas, acciones, etc. (véase el § D.3.8). Si el proceso está subestructurado en servicios, el diagrama de proceso contiene la zona de interacción de servicios (§ D.5.3).

Si existe una subestructura del bloque, algunos de los puntos indicados arriba son facultativos. (Véase el § D.4 para explicaciones sobre estructuración.)

En la figura D-3.6.3 se ilustra un ejemplo de un diagrama de bloque para el bloque «B1» introducido en la casilla de la figura D-3.6.1. El bloque B1 se describe en términos de los dos procesos P1 y P2, conectados por las rutas de señales R1, R2, R3, R4 y R5. Para los procesos P1 y P2 sólo se dan las referencias en este ejemplo. Fuera de la casilla se han especificado también los identificadores de los canales C1, C2 y C3.

En la figura D-3.6.4 se muestra el mismo ejemplo en LED/PR.

Como se ha indicado anteriormente, los diagramas de bloque pueden incluirse dentro del diagrama de sistema, en lugar de tener referencias. Por ejemplo, en la figura D-3.6.5 los diagramas de la figura D-3.6.1 y D-3.6.3 se han fusionado en un solo diagrama de sistema.

Un criterio general para el dibujo de esos diagramas con el fin de facilitar sur lectura es que no sean demasiado complicados y estén contenidos en una sola página.

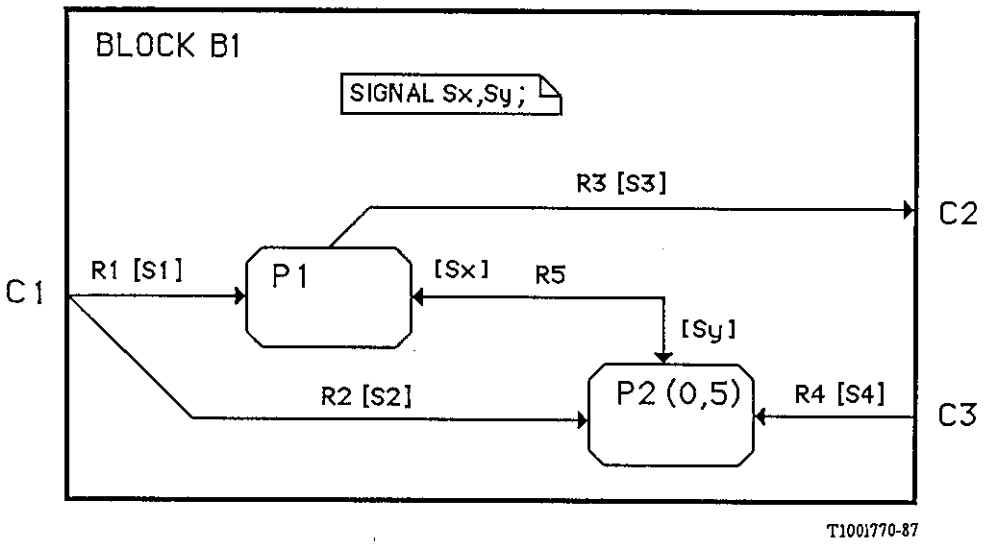


FIGURA D-3.6.3

Ejemplo de diagrama de bloque

```
BLOCK B1;

   SIGNAL Sx,Sy;

   SIGNALROUTE R1 FROM ENV TO P1 WITH S1;
   SIGNALROUTE R2 FROM ENV TO P2 WITH S2;
   SIGNALROUTE R3 FROM P1 TO ENV WITH S3;
   SIGNALROUTE R4 FROM ENV TO P2 WITH S4;
   SIGNALROUTE R5 FROM P1 TO P2 WITH Sy;
                  FROM P2 TO P1 WITH Sx;

   CONNECT C1 AND R1,R2;
   CONNECT C2 AND R3;
   CONNECT C3 AND R4;

   PROCESS P1 REFERENCED;
   PROCESS P2 REFERENCED;

ENDBLOCK B1;
```

FIGURA D-3.6.4

Ejemplo de una definición de estructura de bloque en LED/PR

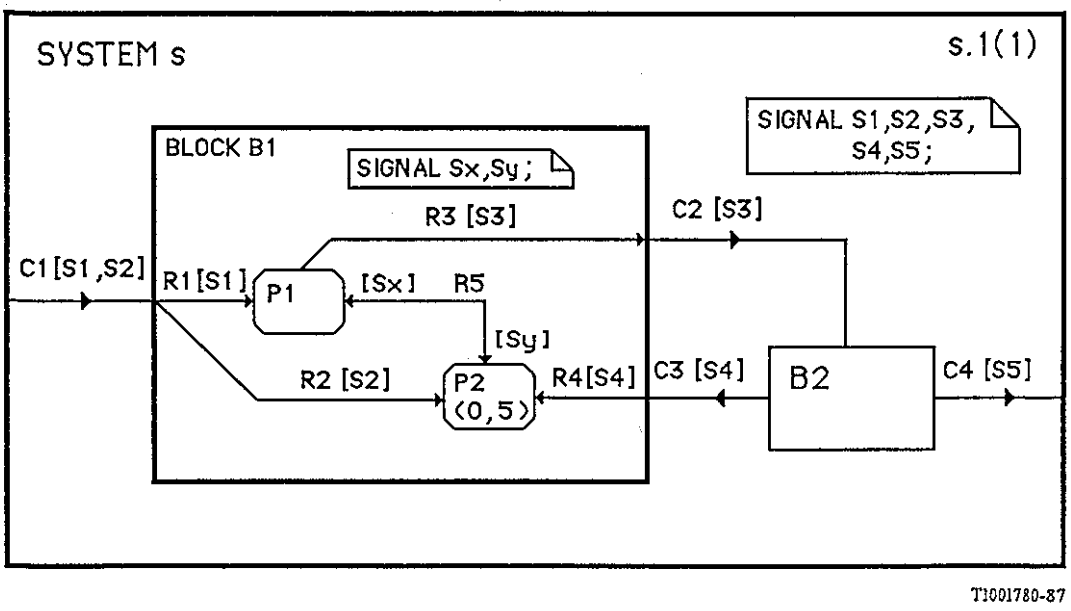


FIGURA D-3.6.5

Ejemplo de un diagrama de sistema que incluye un diagrama de bloque

D.3.7 *Comentarios y ampliación de texto*

D.3.7.1 *Comentarios*

Pueden agregarse comentarios a una especificación LED para ayudar al lector y para aclarar algunas partes. En el LED hay dos tipos de comentarios que han sido introducidos para adaptarse al LED/PR y al LED/GR.

El primer tipo se llama «nota» y se usa especialmente en LED/PR. Está delimitado por «/*» al principio y «*/» al final.

En LED/PR tal comentario puede insertarse dondequiera que pueda aparecer un espacio. El comentario no contendrá la secuencia especial «*/».

En LED/GR tal comentario puede figurar dondequiera que pueda aparecer un espacio en sentencias lineales.

Algunos ejemplos de comentarios de esta forma en LED/PR y LED/GR se muestran en las figuras D-3.7.1 y D-3.7.2, respectivamente.

```
SYSTEM NSS;

/* Definición del sistema de conmutación nacional.

   Nombre del sistema: NSS */

/*******************************/

   ...
   SIGNAL s1; /* Explicaciones sobre el uso de la señal s1 */
   ...
   BLOCK MAN REFERENCED; /* Mantenimiento */
   ...
ENDSYSTEM NSS;
```

FIGURA D-3.7.1

Ejemplos de la primera forma de comentario en LED/PR

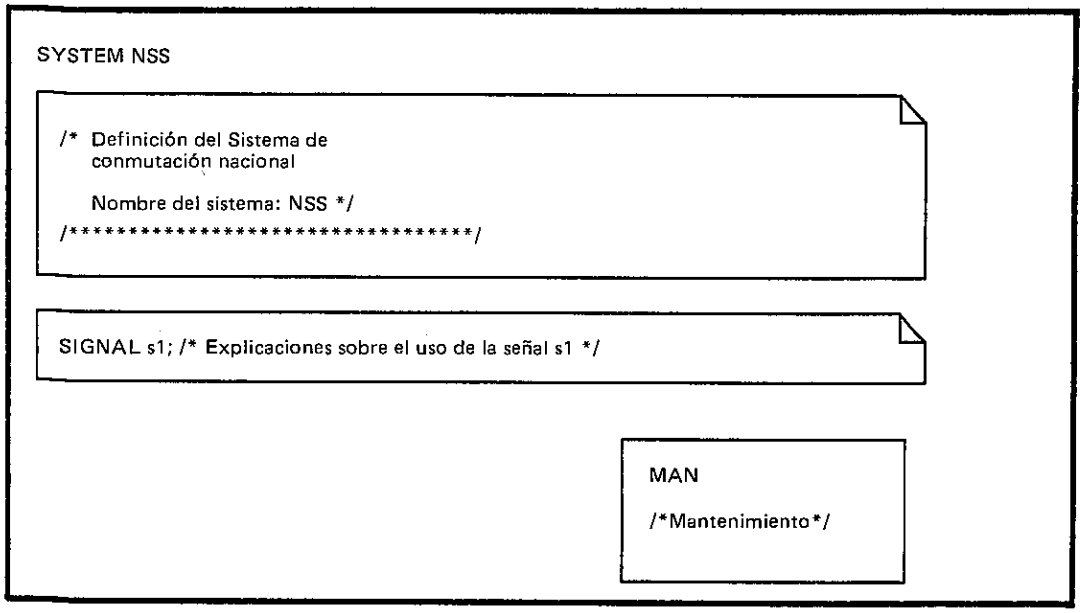


FIGURA D-3.7.2

Ejemplos de la primera forma de comentario en LED/GR

La segunda forma de comentario permite una correspondencia biunívoca entre LED/PR y LED/GR y es más apta para aplicaciones en las que se hacen traducciones automáticas.

En LED/PR un comentario de este tipo consiste en la palabra clave COMMENT seguida por una cadena de caracteres; puede insertarse, como una sentencia (es decir, seguido por un «;»), dondequiera que pueda insertarse una sentencia de tarea. Además, puede insertarse delante del símbolo «;» (punto y coma) al final de cualquier sentencia.

En LED/GR esta forma de comentario se representa por medio de un símbolo de comentario que contiene el texto del comentario. El símbolo de comentario es un símbolo de forma rectangular en que falta el lado izquierdo o el lado derecho. El símbolo debe extenderse horizontal y verticalmente para contener el texto completo. Puede conectarse a cualquier línea de flujo o símbolo LED/GR. Para la conexión debe utilizarse una línea de trazo discontinuo. Si la asociación entre el texto del comentario y el símbolo del comentario no es ambigua, el símbolo de comentario puede dibujarse simplemente como un corchete.

Algunos ejemplos de esta forma de comentario en LED/PR y LED/GR se muestran en las figuras D-3.7.3 y D-3.7.4, respectivamente.

```
SYSTEM s COMMENT 'comentario sobre el sistema';
   CHANNEL C2
      FROM B1 TO B2
      WITH s3          COMMENT 'comentario sobre el canal';
   ENDCHANNEL C2;
   BLOCK B1
   COMMENT 'comentario sobre el bloque';
      ...
      PROCESS p1;
         ...
         TASK 't1';
         TASK 't2';
         ...
      ENDPROCESS p1;
   ENDBLOCK B1;
ENDSYSTEM s;
```

FIGURA D-3.7.3

Ejemplos de la segunda forma de comentario en LED/PR

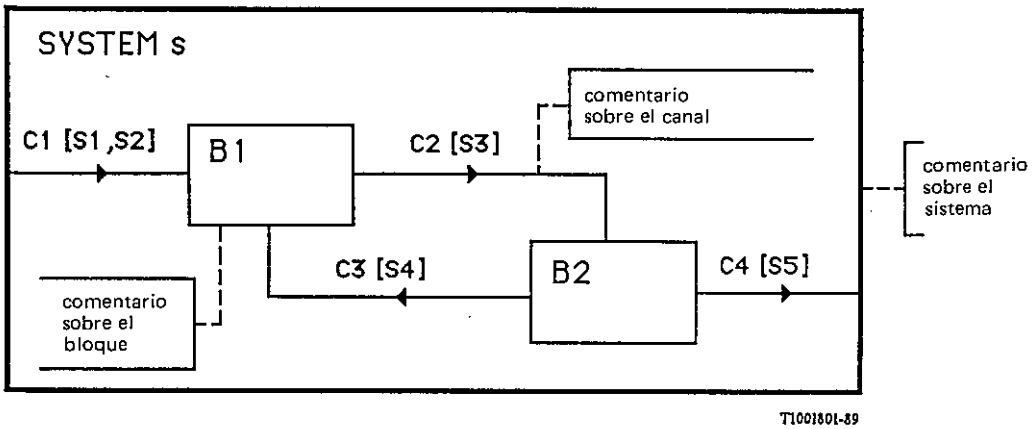


FIGURA D-3.7.4

Ejemplos de la segunda forma de comentario en LED/GR

D.3.7.2 *Ampliación de texto*

Normalmente el texto asociado con un símbolo gráfico debe colocarse dentro de ese símbolo. Sin embargo, a menudo esto no es posible, o no es práctico. Una alternativa es colocar el texto en un símbolo de ampliación de texto conectado al símbolo asociado. El símbolo de ampliación de texto es semejante al símbolo de comentario; la única diferencia es que la línea que lo conecta es de trazo continuo y no discontinuo. (Véase la figura D-3.7.5.)

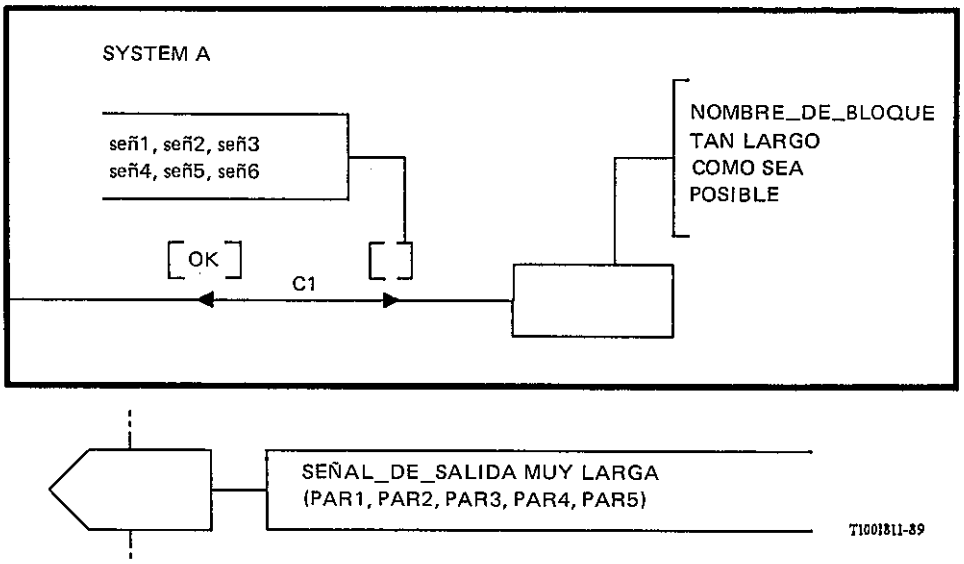


FIGURA D-3.7.5

Ejemplos de uso del símbolo de ampliación de texto

Un carácter de subrayado («_») puede utilizarse al final de una línea de texto como un carácter de continuación. En este caso los espacios restantes en la misma línea no se consideran parte del texto. En el § D.3.13 pueden encontrarse directrices más detalladas sobre la sintaxis de los nombres.

D.3.8 *Procesos*

Un proceso es una máquina de estado finito ampliada que define el comportamiento dinámico de un sistema. Básicamente, en un estado el proceso está esperando señales. Al recibir una señal, el proceso responde efectuando las acciones específicas que están especificadas para cada tipo de señal que el proceso puede recibir. Los procesos contienen muchos estados diferentes, para que puedan efectuar diferentes acciones cuando reciben una señal. Esos estados constituyen la memoria de las acciones que han tenido lugar previamente. Una vez que han tenido lugar todas las acciones asociadas a la recepción de una determinada señal, se pasa a un nuevo estado y el proceso espera otra señal.

Los procesos pueden ya sea existir en el momento en que se crea el sistema o pueden crearse como resultado de una petición de creación proveniente de otro proceso. Además, los procesos pueden existir siempre o pueden cesar mediante la ejecución de una acción de parada.

Una definición de proceso representa la especificación de un tipo de proceso; varias instancias del mismo tipo de proceso pueden ser creadas y existir al mismo tiempo; pueden ejecutarse independientemente y concurrentemente. Una definición de proceso consta de los puntos siguientes (algunos de ellos son facultativos):

- nombre del proceso;
- un par de números enteros: el primer entero especifica el número de instancias de proceso creadas cuando se crea el sistema; si se omite, tiene un valor por defecto de 1; el segundo entero especifica el número máximo de instancias de proceso simultáneas; si se omite, el valor máximo por defecto es ilimitado;
- parámetros formales: una lista de identificadores de variable asociados con sus géneros que se utiliza para pasar información al proceso en el momento de su creación. En la petición de creación de proceso puede proporcionarse con este fin una lista de parámetros efectivos. Los valores de los parámetros formales de procesos creados en el momento de la creación del sistema son indefinidos;
- conjunto de señales de entrada válidas: una lista de identificadores de señal que definen las señales que pueden ser recibidas por el proceso;

– definiciones de señal: la especificación de las señales que pueden intercambiarse entre instancias del mismo proceso o entre servicios en el proceso (§ D.5.3);

– definiciones de procedimiento: la especificación de los procedimientos que pueden ser llamados por el proceso. Puede usarse en este contexto una referencia de procedimiento. (Los procedimientos se explican en el § D.3.9);

– definición de datos: las especificaciones de los neotipos definidos por el usuario, sintipos y generadores locales con respecto al proceso;

– definición de variables: la declaración de las variables del proceso. Facultativamente, una variable puede ser declarada como compartida con otros procesos del mismo bloque (variable REVEALED) o exportada a otros procesos también en otros bloques (variable EXPORTED). Para cada variable declarada debe especificarse el identificador de su género. Puede especificarse facultativamente un valor inicial;

– definiciones de visión: la declaración de identificadores de variables que pueden usarse para obtener los valores de variables pertenecientes a otras instancias de proceso. Para cada identificador de variable debe especificarse el género de la variable;

– definiciones de importación: la especificación de identificadores de variables pertenecientes a otros procesos y que el proceso en cuestión quiere importar. Para cada identificador debe especificarse el género de la variable;

– definición de temporizador: se explica en el § D.3.11;

– definiciones de macro: en el § D.5.1 se dan directrices sobre el uso de macros;

– cuerpo del proceso: la especificación del comportamiento real del proceso en términos de estados, entrada, salida, tarea, etc. Si el proceso está estructurado en subpartes (servicios), la definición de proceso incluye una sección de descomposición de servicio en vez del cuerpo del proceso. En el § D.5.3 se dan directrices sobre esta característica.

En el § D.3.10 pueden encontrarse ejemplos y explicaciones sobre datos, variables, definiciones de visión y definiciones de importación.

En la figura D-3.8.1 se muestra un ejemplo parcial de definición de proceso en LED/PR (las palabras clave del lenguaje están en letras mayúsculas).

```
PROCESS p1 (2,100);
   FPAR var1,var2 gen1, var3 gen2, var4 gen3;  } Parámetros formales
   SIGNALSET s1,s2,s3;                         } Conjunto de señales de entrada válidas
   SIGNAL s4,s5;                               }
   SIGNAL s6 (gen6,gen7);                      } Definiciones de señal
   PROCEDURE . . .                             } Definiciones de procedimiento
   . . . definiciones de tipos de datos . . .  } Definiciones de tipos de datos
   DCL . . .                                   } Definiciones de variables
   . . . cuerpo del proceso . . .              } Cuerpo del proceso
ENDPROCESS p1;
```

FIGURA D-3.8.1

Ejemplo parcial de definición de proceso en LED/PR

El cuerpo del proceso representa el gráfico efectivo de la máquina de estado finito. En LED/PR consiste en una secuencia de sentencias ordenadas, y en LED/GR es una secuencia de símbolos que están conectados por arcos dirigidos (en forma similar a un diagrama de flujo). La especificación del cuerpo del proceso debe comenzar siempre con un START seguido por un conjunto de acciones (transición). La interpretación de una instancia de proceso comienza cuando se crea el proceso.

Las acciones que pueden realizarse en una transición son:

– tarea: asignación de variable (o texto informal);

– exportación: exportación de variable;

– inicializar (poner): petición de activación de un temporizador;

– reinicializar (reponer): reinicialización (reposición) de un temporizador;

– salida: envío de una señal a otro proceso;

– petición de crear: creación de una instancia del tipo de proceso especificado;

– decisión: selección de un conjunto de acciones que dependen de una pregunta;

– llamada de procedimiento: petición de interpretación de un conjunto específico y autónomo de acciones (como en los lenguajes de programación);

– unión: especificación de un «salto» a otro conjunto de acciones.

Una transición puede terminar con una de las siguientes acciones:

– estado-siguiente: especificación del estado que asumirá la instancia de proceso;

– parada: detención inmediata de la instancia de proceso.

Después de la especificación de la acción de arrancar y de la transición de arranque facultativa, el cuerpo del proceso incluye las definiciones de todos los posibles estados del proceso. Cada definición de estado comienza con la especificación de los estímulos posibles que el proceso espera en ese estado. Los estímulos posibles son:

– entradas: señales que pueden ser recibidas;

– conservaciones: señales que necesitan conservarse para su procesamiento futuro;

– condiciones habilitadoras: se explican en el § D.3.8.5;

– señales continuas: se explican en el § D.3.8.5.

Debe especificarse una transición correspondiente a cada estímulo, excepto las conservaciones. Esta transición representa la secuencia de acciones que ejecutará el proceso si tiene lugar ese estímulo.

Si un proceso ejecuta una acción de parada y hay señales pendientes que le fueron enviadas pero no han sido aún recibidas, dichas señales se desechan.

Una definición de proceso se representa en LED/GR por medio de un diagrama de proceso. Un diagrama de proceso consta de los puntos siguientes:

– un símbolo de casilla: un símbolo de forma rectangular que contiene todos los otros símbolos. Si no hay rutas de señales conectadas al símbolo de casilla, éste puede omitirse;

– el encabezamiento de proceso: la palabra clave PROCESS seguida del identificador de proceso, después por la posible especificación de instancias de proceso y por la especificación de los parámetros formales. El encabezamiento de proceso se coloca en la esquina superior izquierda de la casilla;

– numeración facultativa de páginas (colocada en la esquina superior derecha);

– símbolos de texto: en el caso de un diagrama de proceso puede utilizarse un símbolo de texto que contenga definiciones de señal, variable, visión, importación, datos y temporizador;

– referencias de procedimiento: una referencia de procedimiento es un símbolo de procedimiento que contiene un nombre de procedimiento que representa un procedimiento del proceso definido por separado;

– diagramas de procedimiento: uno para cada procedimiento local del proceso que no está referenciado;

– la zona de gráfico de proceso: la especificación del comportamiento del proceso en términos de arranque, estados, entradas, salidas, tareas, . . . y arcos dirigidos. Si el proceso está estructurado en servicios, la zona de gráfico de proceso contiene la especificación de los servicios o sus referencias (véase § D.5.3). Los símbolos GR utilizados para el cuerpo del proceso pueden encontrarse en el resumen de la sintaxis LED/GR;

– diagramas de macro: en el § D.5.1 se dan directrices para el uso de macros.

La figura D-3.8.2 es un ejemplo de definición de proceso en LED/GR; en los párrafos siguientes puede encontrarse más explicaciones y ejemplos de gráficos de proceso.

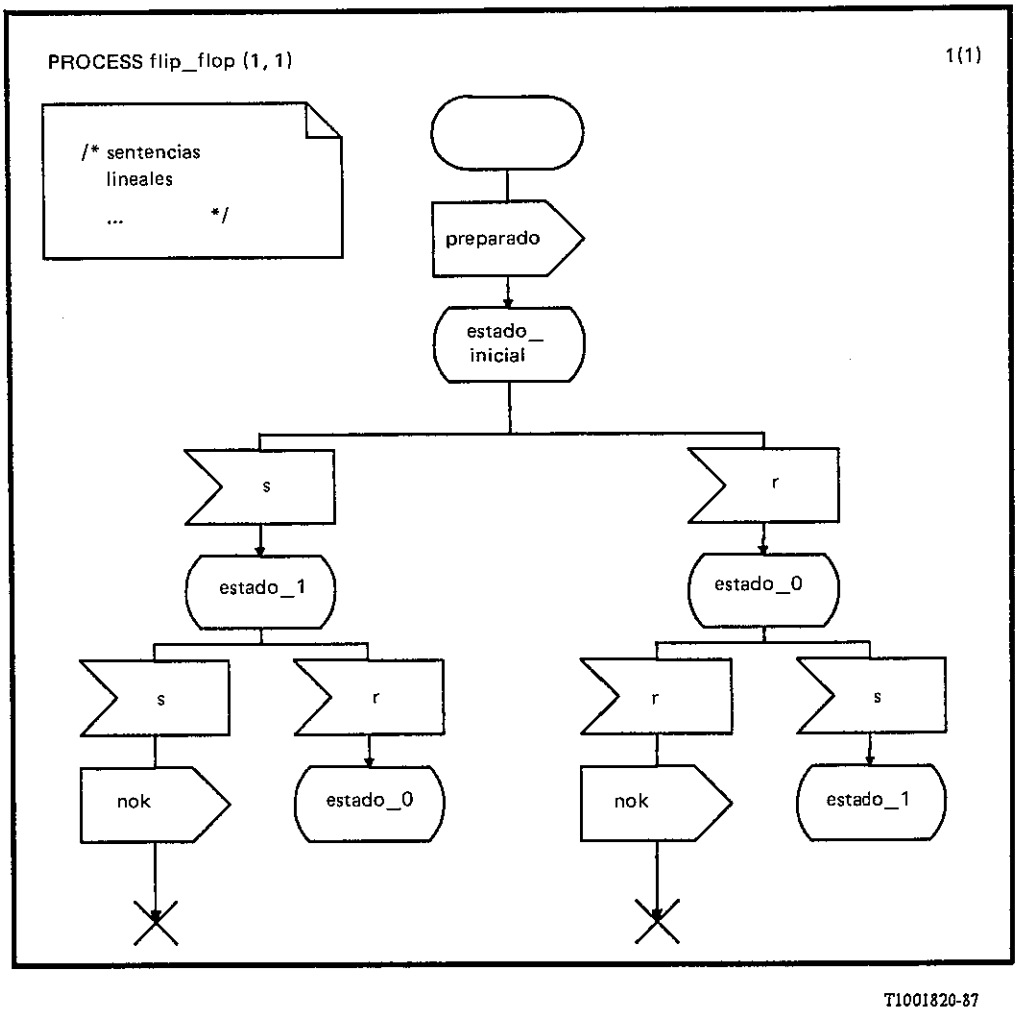


FIGURA D-3.8.2

Ejemplo de diagrama de proceso

Si no hay espacio suficiente en una sola página para un gráfico de proceso, el diagrama puede dibujarse en varias páginas, haciendo notar que:

- Debe proporcionarse una numeración de las páginas que comprenda el número de la página y el número total de páginas, es decir: 1 (9).
- El símbolo de unión o el símbolo de estado-siguiente pueden utilizarse para representar conexiones entre las diferentes partes del gráfico de proceso.

Un buen criterio para dividir un gráfico de proceso en partes es representar una definición de estado en cada página. Si no hay espacio suficiente en una página para una sola definición de estado puede utilizarse el símbolo de unión para expresar conexiones con una porción del gráfico dibujada en otra página.

D.3.8.1 *Creación de procesos*

Como se ha expresado en el punto precedente, pueden crearse procesos (es decir, instancias de proceso) como resultado de una petición explícita o al crearse el sistema.

La petición explícita de crear puede ser emitida por otro proceso en el mismo bloque del proceso que está siendo creado y permite la especificación de parámetros efectivos para transferir información a la nueva instancia creada. Véase la figura D-3.8.3 para un ejemplo de creación en las formas PR y GR.

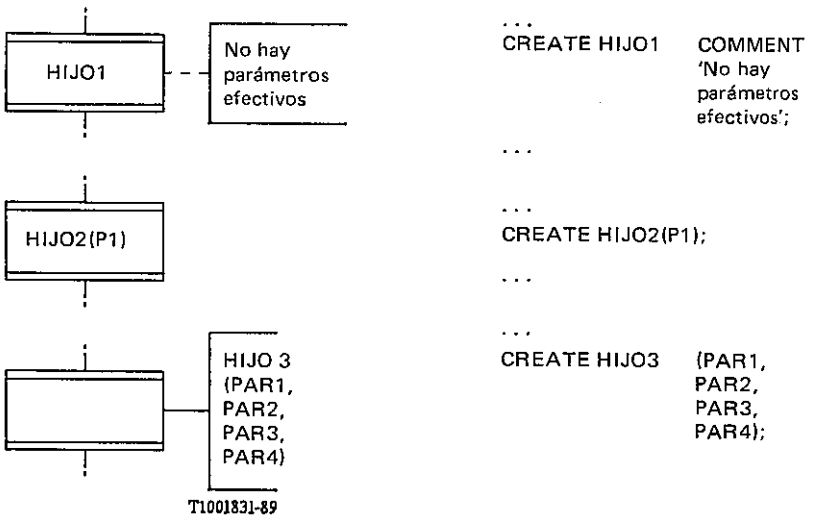


FIGURA D-3.8.3

Ejemplos de peticiones de crear en LED/GR y LED/PR

Si, en el momento de la creación del sistema, se crean una o más instancias de proceso de un tipo dado de proceso, la definición de ese proceso no deberá ganar acceso a parámetros formales porque éstos estarán indefinidos. En consecuencia, cada definición de proceso con parámetros formales debe, o bien asegurar que no se ganará acceso a sus parámetros formales antes que se les den valores, o bien incluir explícitamente la especificación de cero instancias de proceso en el momento de la creación del sistema (véase la figura D-3.8.4).

```
PROCESS proc_creado_explicitamente_1 (0, . . .);
   FPAR . . .
   . . .
ENDPROCESS proc_creado_explicitamente_1;
```

FIGURA D-3.8.4

Definición de un proceso con especificación de parámetros formales

Cuando se crea un proceso, pueden determinarse valores de instancia de proceso por medio de las siguientes expresiones predefinidas:

OFFSPRING (DESCENDIENTE) (o VÁSTAGO): retorna el valor de PId de la última instancia creada.

SELF (MISMO): retorna el valor de PId de la misma instancia de proceso.

PARENT (ASCENDIENTE) (o PROGENITOR): retorna el valor de PId de la instancia creatriz.

SENDER (EMISOR): retorna el valor de PId del proceso que envía la última señal consumida.

Cuando un proceso es creado como resultado de la creación del sistema, sólo la expresión SELF retorna un valor de PId; las expresiones OFFSPRING y PARENT dan el valor NULL (NULO).

Estos valores son muy importantes cuando hay más instancias de proceso del mismo tipo de proceso, porque son el único medio para direccionar inequívocamente señales a las instancias. De hecho, como se explicará mejor en el § D.3.8.6, cuando un proceso envía una señal debe especificar la instancia de destino, a menos que ésta sea inequívocamente determinable.

El usuario debe asegurarse cuidadosamente de que las instancias de proceso creadas puedan comunicarse entre sí si es necesario. Para conseguir esto, debe a menudo proporcionarse alguna clase de procesos de inicialización. Dichos procesos, creados en el momento de la creación del sistema, tendrán que crear los otros procesos y posiblemente comunicar valores apropiados de PId a todos los procesos que necesitan saberlos.

Otras notas importantes sobre la creación de procesos son las siguientes:

1) Una vez creado el sistema, los procesos sólo pueden ser creados por otro proceso del mismo bloque. Una manera de permitir la creación de procesos en otros bloques consiste en prever procesos especiales en cada bloque que causen la creación de un proceso cuando reciban una señal de un proceso de otro bloque.

2) Una vez creados, los procesos tienen un periodo de vida propio. Los procesos mueren solamente cuando ejecutan una acción de parada durante una transición. Una manera de modelar sistemas que permite operaciones externas que matan procesos consiste en prever una señal eliminadora especial. Cuando se recibe la señal eliminadora, el proceso ejecuta una acción de parada.

La relación entre procesos creadores y creados puede mostrarse en un diagrama de bloques por medio de símbolos de línea de crear, según se ilustra en la figura D-3.8.5.

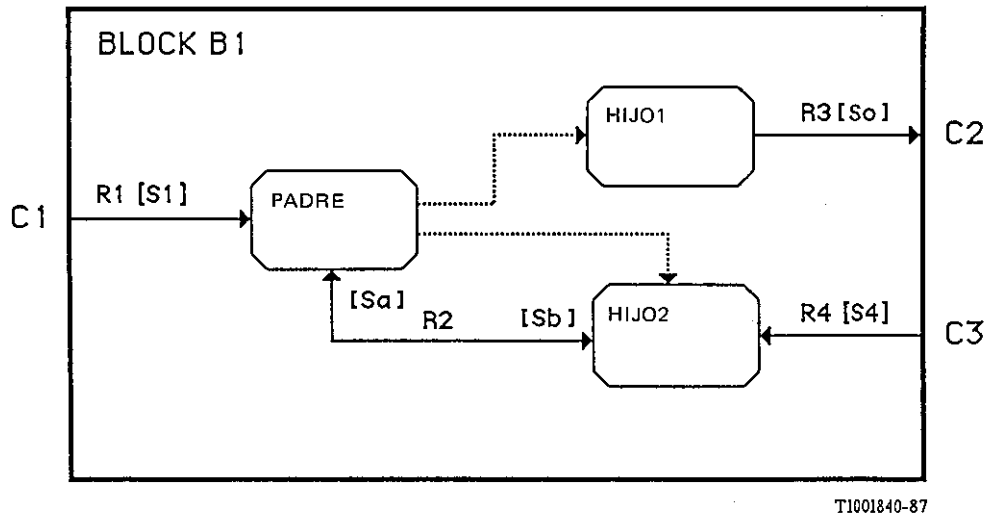


FIGURA D-3.8.5

Ejemplo de diagrama de bloque con símbolos de línea de crear

D.3.8.2 *Estados*

Un estado es un punto del proceso en el que no se ejecutan acciones sino que la fila de espera de entrada espera la llegada de señales entrantes. Según el identificador de señal que aparece en la señal de entrada, la llegada de la señal hará que el proceso deje el estado y ejecute una determinada secuencia de acciones. Una señal que ha llegado y causado una transición ha sido «consumida» y deja de existir.

En LED/PR un estado se representa por la palabra clave STATE seguida del nombre del estado. La especificación del estado termina, ya sea con la siguiente sentencia de estado o al final del proceso o por medio de la palabra clave explícita ENDSTATE. Puede utilizarse un asterisco en la sentencia de estado en vez del nombre del estado; esta es una notación abreviada para indicar que las entradas o conservaciones que siguen y las transiciones correspondientes deberán interpretarse en cada estado.

Para indicar el estado que sigue se utiliza la palabra clave NEXTSTATE seguida del nombre del estado. Para indicar que el estado que sigue es el mismo estado en el que se originó la transición en curso, se puede utilizar un guión en la sentencia de estado-siguiente en lugar del nombre del estado.

En la figura D-3.8.6 hay un ejemplo parcial en LED/PR:

```
SYSTEM s;
   ...
   BLOCK b;
      ...
      PROCESS p;
         ...
         START;
            ...
         STATE st1;
            ...
         STATE st2;
            ...
            NEXTSTATE st1;
         ...
         ENDSTATE st2;
         STATE st3;
            ...
      ENDPROCESS p;
   ENDBLOCK b;
ENDSYSTEM s;
```

FIGURA D-3.8.6

Ejemplo parcial de LED/PR para definiciones de estado

En LED/GR un estado se representa por medio de un símbolo de estado que contiene el nombre del estado y está conectado a símbolos de entrada o a símbolos de conservación. El mismo símbolo de estado con una flecha de entrada se utiliza para representar una sentencia de estado siguiente.

Por conveniencia, para simplificar el dibujo o como una ayuda para la mejor comprensión, el mismo estado puede aparecer varias veces en un diagrama LED. Cuando esto sucede, se considera que el diagrama es totalmente equivalente al diagrama que resultaría de la fusión de todas las apariciones del mismo estado. Las figuras D-3.8.7 y D-3.8.8 presentan ejemplos. En la figura D-3.8.7 b) se utiliza un símbolo de estado como un conector con el estado principal que tiene el mismo nombre, cuando éste se menciona en un símbolo de estado-siguiente. En la figura D-3.8.8 se representa un estado por múltiples símbolos cada uno de los cuales tiene solamente un subconjunto de entradas (o conservaciones).

En la figura D-3.8.7 los diagramas a) y b) son equivalentes lógicamente. El diagrama a) contiene una sola aparición de cada estado mientras que el diagrama b) presenta múltiples apariciones. En el diagrama b) el estado tiene una aparición principal donde se muestran todos los símbolos de entrada (y conservación) correspondientes. Donde este estado puede alcanzarse desde otros puntos del diagrama (como el punto terminal de una transición), se muestra como un estado sin ninguna entrada ni conservación asociada. Un comentario referente al símbolo de estado desde el símbolo de estado-siguiente mejorará la claridad, particularmente cuando aparezca en diferentes páginas.

La figura D-3.8.8 utiliza múltiples apariciones de un estado para formar el conjunto completo de entradas (y conservaciones). Cada ocurrencia del estado se muestra con un subconjunto de éstas solamente.

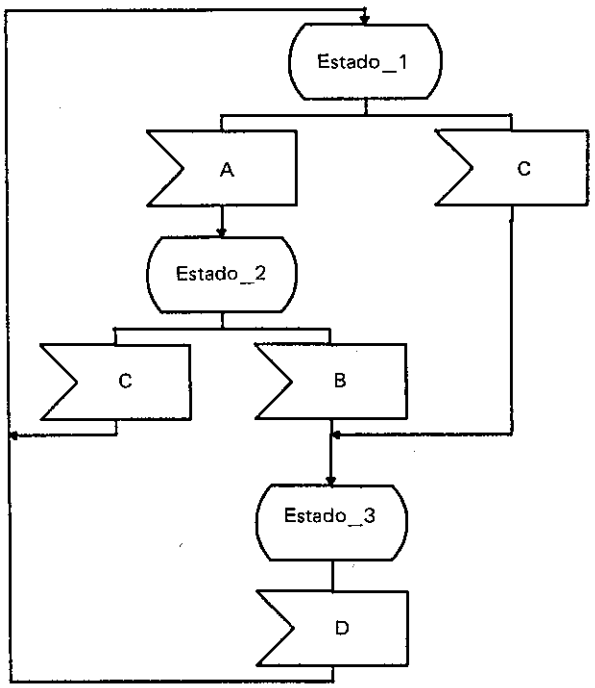


FIGURA D-3.8.7 a

Diagrama en que cada estado aparece una sola vez

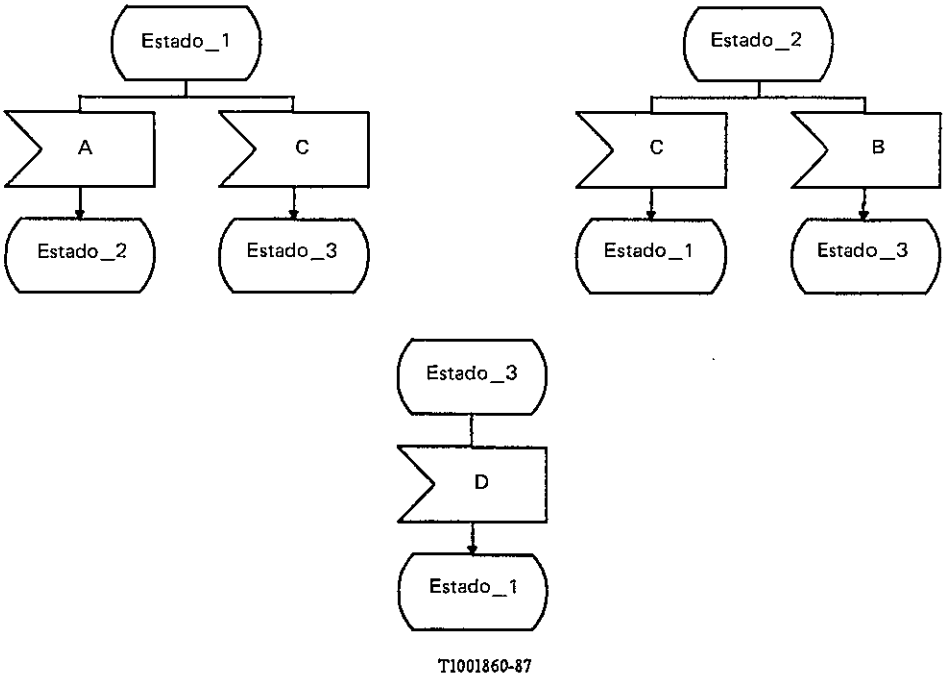


FIGURA D-3.8.7 b

Diagrama a) con los estados principales y los estados ulteriores utilizados como conectores con los estados

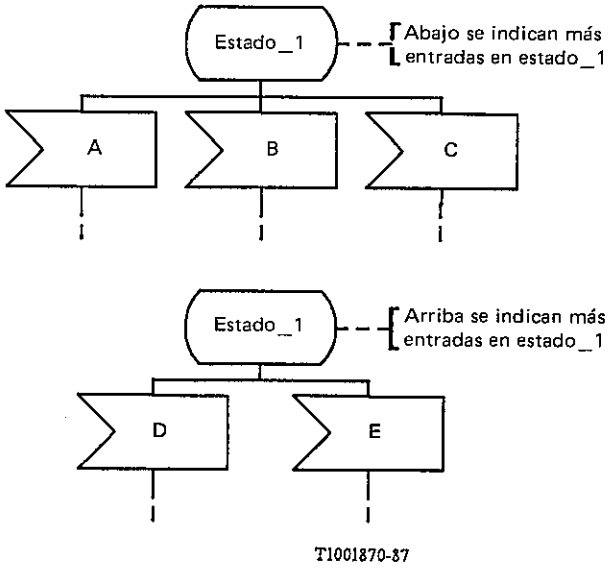


FIGURA D-3.8.8

Diagrama en que un estado aparece más de una vez cuando no pueden dibujarse claramente todas las entradas asociadas al mismo símbolo

Este método se presta a ser utilizado cuando ciertos estados tienen un número relativamente grande de entradas o conservaciones, pero puede tener el inconveniente de que los lectores interpreten mal el diagrama si ignoran que hay múltiples apariciones. Para evitar esta mala interpretación, los estados que indiquen solamente un subconjunto de entradas/conservaciones deben ir acompañados de un comentario que haga referencia a los otros estados con sus entradas y conservaciones asociadas, según se muestra en la figura D-3.8.8.

Puede resultar conveniente utilizar múltiples apariciones para llamar la atención del lector sobre ciertos aspectos (por ejemplo, la secuencia normal de señales procesadas), dejando otros aspectos para otras páginas (por ejemplo, tratamiento de situaciones de alarma).

Durante una transición, un proceso no sabe explícitamente qué señal de entrada ha causado la transición. Esto sólo puede inferirse del contexto (es decir, esa transición sólo puede ocurrir si se recibe una determinada señal). En la figura D-3.8.9, la tarea T1 se efectúa únicamente si se recibe I1. Pero la tarea T2 será ejecutada si se recibe I2 ó I3. Si es importante que T2 sepa qué entrada se ha recibido, será mejor diseñar el proceso como se muestra en la figura D-3.8.10

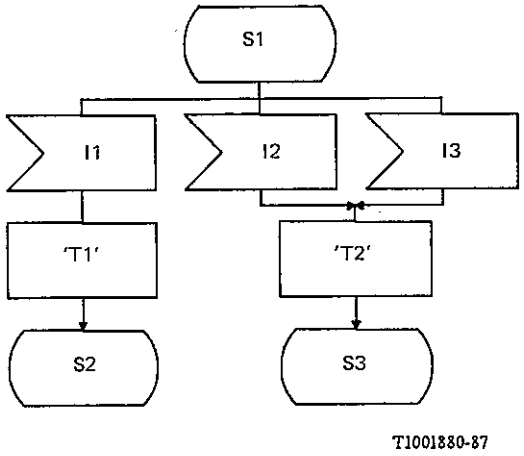


FIGURA D-3.8.9

Ejecución de una tarea en función de una cualquiera de dos de las tres señales recibidas

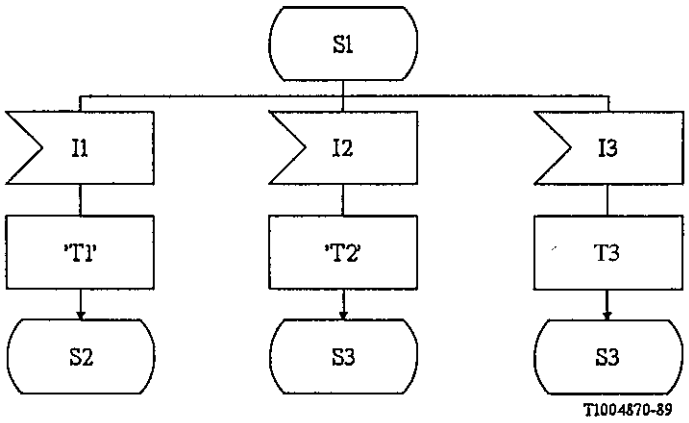


FIGURA D-3.8.10

Ejecución de una tarea en función de la señal recibida

D.3.8.2.1 *Determinación de los estados necesarios*

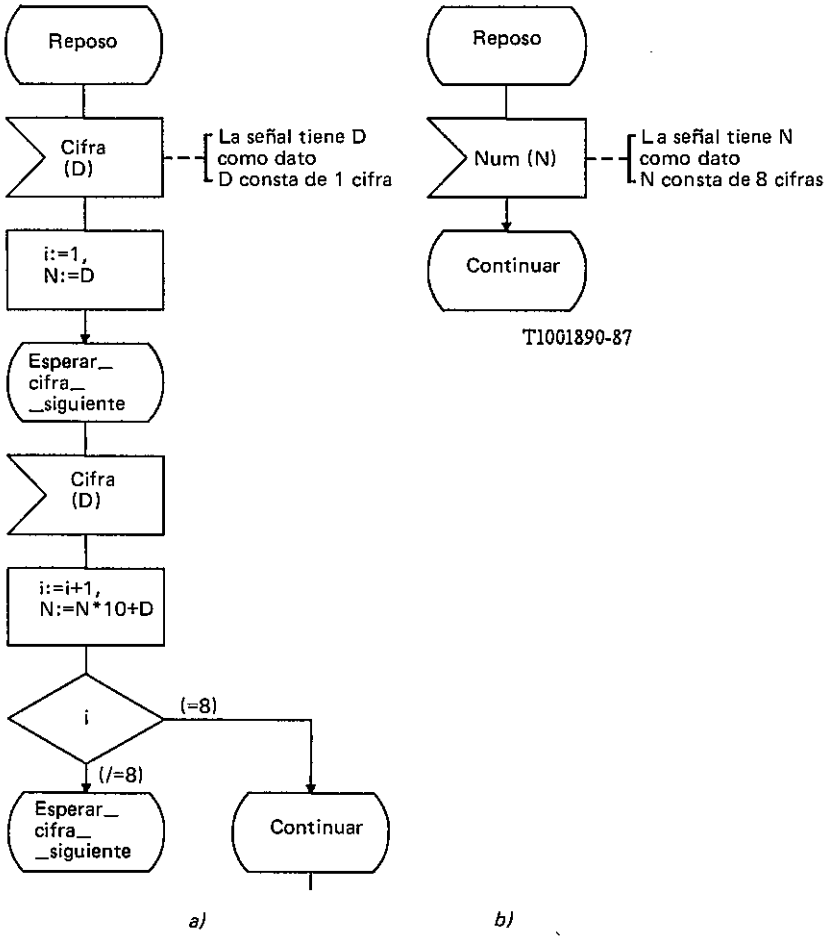
Habitualmente, el autor de un diagrama LED dispone de cierta flexibilidad en su método de definir los estados de un proceso. Puede necesitar una estrategia que le permita identificar los estados del proceso; esta estrategia puede ser informal o formal. Se requiere un buen juicio (obtenido mediante la práctica) para producir diagramas LED que no sean innecesariamente complejos debido a la identificación de un número excesivo de estados distintos, o que dejen de aprovechar las ventajas propias del LED reduciendo artificialmente el número de estados. Antes de que el autor comience la construcción del diagrama deben haber quedado resueltas las cuestiones preliminares (examinadas en el § D.3.2), por ejemplo:

- la estructuración del sistema en bloques funcionales;
- la representación de los bloques funcionales por medio de uno o más procesos LED por bloque;

– la elección de señales de entrada y de salida;

– la utilización de datos en el proceso.

Todos esos factores son de capital importancia para la determinación de los estados de cada proceso. La influencia de la «elección» de las señales de entrada en el número de estados de un diagrama LED se ilustra en los dos ejemplos de la figura D-3.8.11.



Nota – Los ejemplos *a)* y *b)* representan la misma función con diferentes niveles de detalle. En consecuencia el número de estados varía.

FIGURA D-3.8.11

Recepción de un número telefónico de 8 cifras

D.3.8.2.2 *Reducción del número de estados*

Después de haber seguido determinados principios para la identificación de los estados de un proceso el autor de un diagrama LED puede considerar que ha utilizado «demasiados estados». El número de estados es importante porque el tamaño y la complejidad de un diagrama LED suelen estar estrechamente relacionados con el número de estados. Hay ciertos medios para reducir el número de estados; no obstante, el hecho de que un diagrama LED sea complejo no es motivo por sí solo para modificarlo, pues la complejidad del diagrama puede no ser más que la consecuencia natural de la complejidad del proceso definido. En general, debe elegirse un conjunto de estados tal que la interacción secuencial entre el proceso y su entorno tenga una claridad máxima. Por lo general no puede obtenerse esta claridad reduciendo al mínimo el número de estados. El número de secuencias independientes tratadas por un proceso tiene un efecto multiplicativo sobre el número de estados. Por ello conviene tratar las secuencias independientes en procesos separados, pues esto reduce el número de estados y aumenta la claridad.

El número de estados puede reducirse separando funciones comunes, fusionando estados, o utilizando el concepto de procedimiento o el de servicio. Ciertas estructuras de datos pueden conducir también a un número reducido de estados. Otra representación puede beneficiarse del empleo de macros.

D.3.8.2.3 *Separación de funciones comunes*

Cuando se proyecta un diagrama LED puede resultar evidente que, para definir un aspecto particular y repetitivo de un proceso, se requiere la representación de estados repetitivos. En la figura D-3.8.12 se ilustra una parte de un diagrama LED de un proceso de señalización de línea que ilustra la siguiente condición: para considerar que se ha detectado una señal de línea deberá identificarse la presencia de un tono de señalización de línea durante un periodo de tiempo determinado.

Para especificar esto se necesita un estado intermedio entre los estados de no_se_detecta_señal_de_línea y conversación. Supongamos que, en un diagrama completo, esta función común tuviera que repetirse en cada punto en que se detecta la señal. Podría procederse de otro modo: definir un proceso separado que se encargara de supervisar el tono de señalización de línea y detectar señales sobre la base del tiempo de identificación especificado. Al existir este nuevo procedimiento, el diagrama de la figura D-3.8.12 se podría dibujar como muestra la figura D-3.8.13. (En un contexto dado, las figuras podrían hacerse equivalentes, si bien para ello se necesitaría introducir una nueva señal: señal_de_línea_válida.)

Conviene señalar la ligera diferencia existente entre el proceso mostrado en la figura D-3.8.12 y los dos mostrados en la figura D-3.8.13.

El proceso de la figura D-3.8.12 comienza a tasar inmediatamente después de la recepción de T1, mientras que el proceso de tratamiento de llamada (proceso b) de la figura D-3.8.13) comienza a medir al recibirse la señal_de_línea_válida, que es generada y enviada a la recepción de T1 por el proceso a) de la figura D-3.8.13. Ello significa que en este segundo caso, la tasación comienza después de T1 + tiempo de generación, envío y recepción de la señal. Además, puede haber otras señales que se han acumulado en espera durante el tiempo que se necesita para generar y enviar la señal_de_línea_válida.

Un segundo aspecto que conviene señalar es que esta separación de una subfunción no será posible si la señal que ha de recibir la subfunción ha de recibirla también la función principal, porque una señal es dirigida siempre a un solo proceso. Si en el ejemplo la misma señal S_aus ha de recibirse en otro estado, donde no es preciso validarla durante el tiempo T1, no será posible separar la subfunción de validación en otro proceso.

En general, las soluciones con procesos separados resultan útiles cuando las señales deben procesarse de una manera que es independiente de los estados del proceso principal. En este caso, unos procesos previos y posteriores pueden tratar las secuencias detalladas y liberar al proceso principal de los detalles. A menudo esto resultará en una útil modularidad también, ya que las particularidades de, por ejemplo, los sistemas de señalización, pueden aislarse del proceso principal, que está más orientado al servicio.

Otro medio de abordar el problema consiste en utilizar el concepto de servicio, que se explica en el § D.5.3.

Una solución distinta de este problema consistiría en utilizar la notación MACRO, como se indica en el § D.3.3.1. En este caso obtenemos un diagrama compacto como el deseado sin alterar en lo más mínimo la semántica del diagrama original. Además, la MACRO puede ser llamada desde varios estados si así lo exige la lógica del proceso.

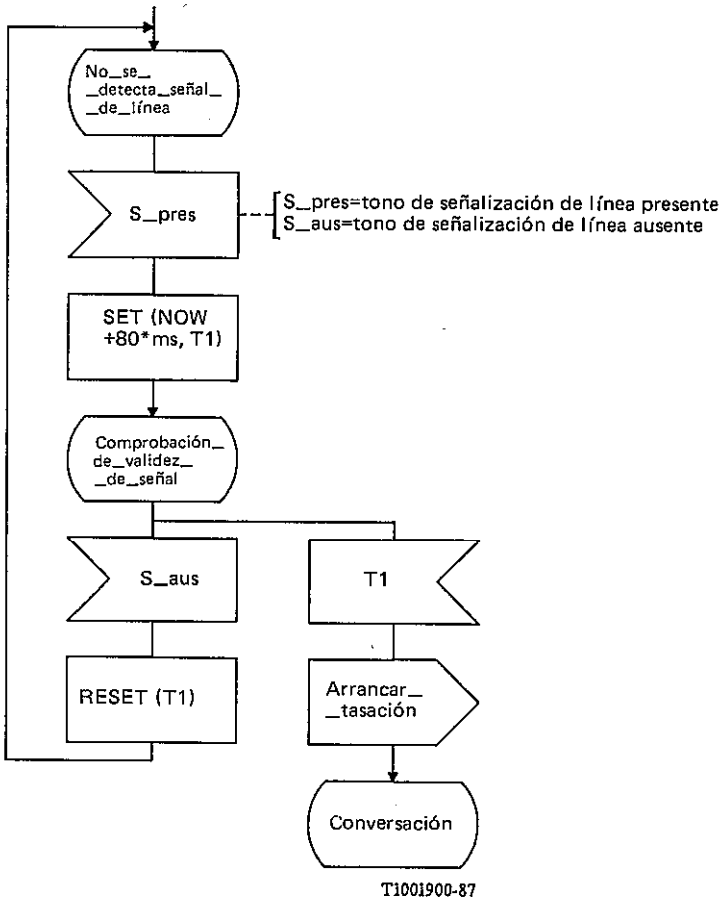
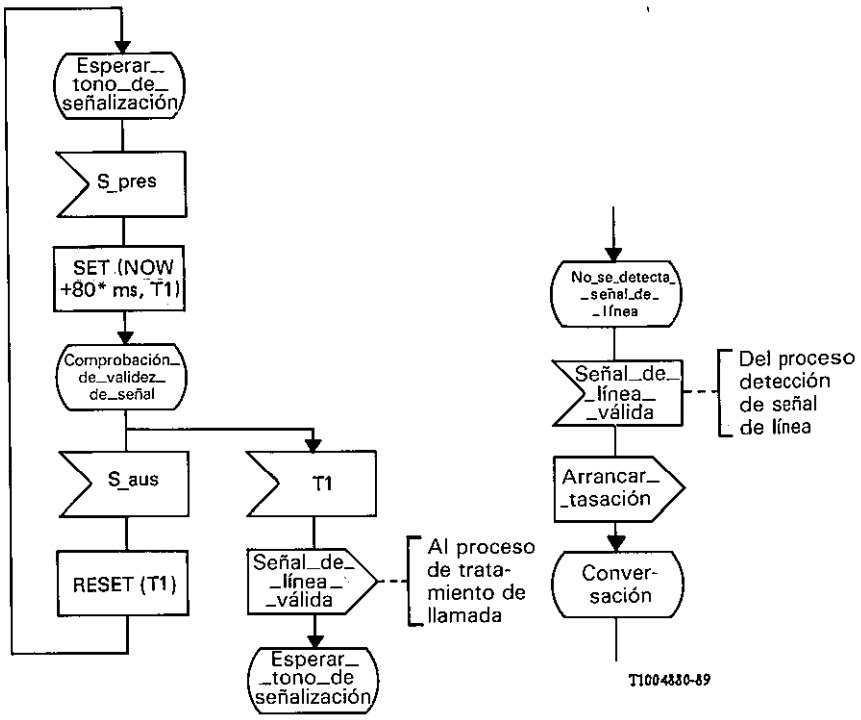


FIGURA D-3.8.12

Ejemplo de un diagrama LED para un proceso compuesto de tratamiento de llamada y detección de señal de línea



a) Proceso de detección de señal de línea

b) Proceso de tratamiento de llamada

FIGURA D-3.8.13

Ejemplo de segregación de una función común (detección de señal de línea) para evitar estados repetitivos como el de comprobación de validez de señal

D.3.8.2.4 *Fusión de estados*

Si en un diagrama LED dos estados tienen el mismo futuro, estos estados, cualesquiera que hayan sido sus antecedentes (historia), podrán fusionarse en uno solo sin que esto afecte a la lógica del diagrama.

La figura D-3.8.14 muestra una parte de un diagrama LED con dos estados que difieren en su historia pero cuyos futuros son «idénticos». En la figura D-3.8.15 los dos estados han sido combinados para formar uno solo. Éste es un ejemplo bastante trivial en que sólo se obtiene una pequeña reducción de la complejidad, pero esta técnica puede utilizarse para obtener una simplificación apreciable. La semántica del LED no permite que una decisión consecutiva a un estado determine la historia del proceso antes del estado (esto es, a los efectos de este ejemplo ¿se envió A6 o B4?) a menos que esta información haya sido almacenada explícitamente antes de que el proceso haya entrado a este estado. Obsérvese que el nombrar los datos en una entrada hace que el valor sea almacenado.

Un estado debe tener una relación clara con las posibles situaciones lógicas del proceso, y por lo tanto no es aconsejable fusionar situaciones lógicas diferentes en un solo estado.

Ha de prestarse atención cuando un diagrama fusionado se modifica más adelante. El usuario debe investigar si el cambio previsto influye del mismo modo o no en las dos (o más) ramas originales de la línea de flujo.

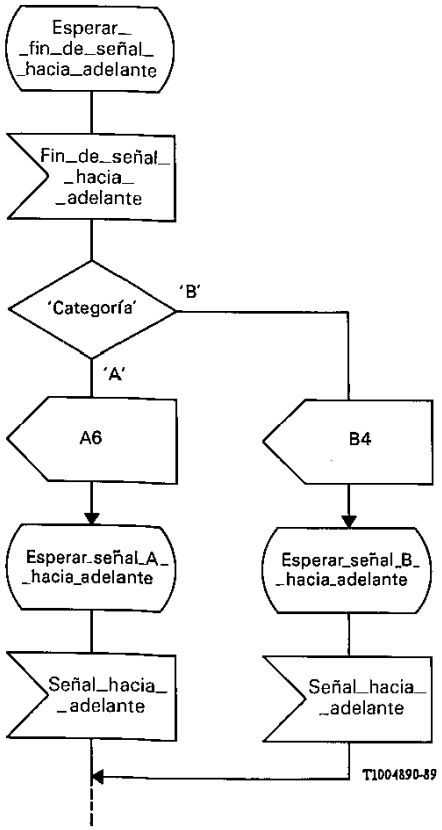


FIGURA D-3.8.14

Ejemplo de una parte de un diagrama LED antes de la fusión de estados

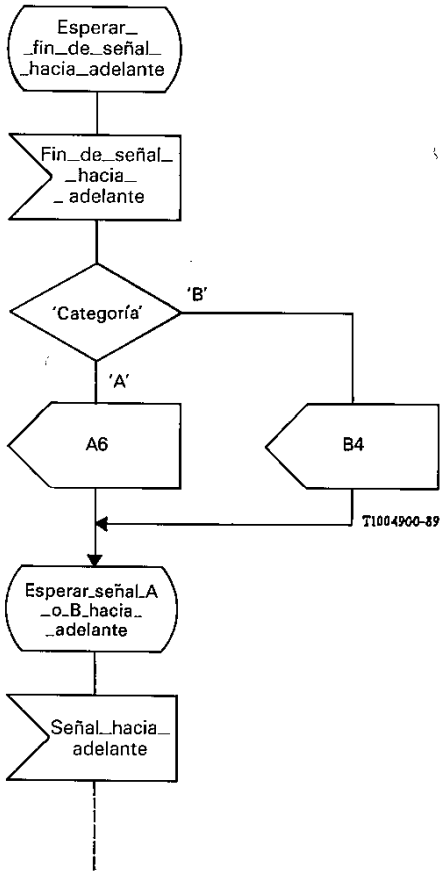


FIGURA D-3.8.15

Ejemplo de una parte de un diagrama LED con estados fusionados

D.3.8.3 *Entradas*

El presente § D.3.8.3 tiene por objeto explicar el concepto de entrada y el uso de entradas en diagramas LED sin el concepto de conservación. El concepto de conservación y el uso conjunto de entradas y conservaciones en un diagrama LED se explica en el § D.3.8.4.

D.3.8.3.1 Generalidades

Un símbolo de entrada asociado a un estado significa que si la señal cuya denominación está inscrita en el símbolo de entrada llega cuando el proceso se encuentra en ese estado habrá que interpretar la transición que sigue al símbolo de entrada. Cuando una señal provoca la interpretación de una transición, deja de existir y se dice que ha sido consumida.

Una señal puede tener valores asociados. Por ejemplo, una señal llamada «cifra» sirve no sólo para provocar la ejecución de una transición por el proceso receptor sino también para transportar el valor de la cifra (0-9), dato que puede utilizar el proceso receptor.

En LED/PR la sentencia INPUT contiene una lista de identificadores de señal. Los valores contenidos en las señales se nombran utilizando identificadores de variable. Los identificadores de variable tienen que ser del género indicado en la definición de señal, de modo que su posición es muy importante. Estos identificadores de variable están contenidos entre paréntesis ordinarios, y están separados por comas (véase la figura D-3.8.16). Si uno o más valores de señal se descartan, las variables correspondientes no están presentes, y esto se representa por dos o más comas consecutivas (figura D-3.8.17).

```
...
SIGNAL señ1 (INTEGER,BOOLEAN,INTEGER);
...
PROCESS ...
   ...
   DCL a INTEGER, b BOOLEAN, c INTEGER; /* declaraciones */
   ...
   STATE st1;
      INPUT señ1(a,b,c); /* una entrada correcta */
      ...
   STATE st2;
      INPUT señ1(a,c,b); /* una entrada incorrecta */
      ...
ENDPROCESS ...
```

FIGURA D-3.8.16

Sentencias INPUT

```
INPUT a(var1,var2,,var4);
```

Nota – En esta sentencia, el tercer valor de la señal se ha descartado.

FIGURA D-3.8.17

Entrada de una señal a con sólo 3, de sus 4 valores, definidos

En LED/GR una entrada se representa por medio de un símbolo de entrada que contiene la lista de los identificadores de señal y los correspondientes identificadores de variable para los valores transportados. Para hacer estos valores accesibles al proceso, deben ser nombrados en los símbolos de entrada entre paréntesis.

En las figuras D-3.8.18, D-3.8.19 y D-3.8.20 se muestran ejemplos de la recepción de valores a partir de entradas.

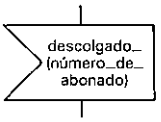
Los valores nombrados se hacen disponibles para el proceso receptor cuando se interpreta la entrada.

La figura D-3.8.18 muestra la recepción de la señal «descolgado». La señal «descolgado» tiene asociado el valor 9269 (número de abonado). Esta señal puede recibirse de tres maneras, como se indica en las partes a) a c) de la figura.

La figura D-3.8.19 muestra cómo enviar y recibir varios valores en una señal. Cada item debe estar separado del siguiente por una coma. En la figura D-3.8.20 c), se muestra cómo ignorar valores no deseados dejando un hueco en la lista de géneros.

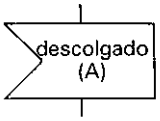
Obsérvese que en la señal de salida podrían escribirse expresiones para E1, E2 o E3, pero en la señal de entrada hay que utilizar variables para recibir los valores enviados.

En LED no es necesario incluir símbolos de entrada para representar señales cuya llegada iniciaría una transición nula (esto es, una transición que no contuviera acciones y que condujera al mismo estado de que se partió). Por convenio, para una señal cualquiera no indicada en un símbolo de entrada explícito en un estado particular, existe, en ese estado, un símbolo de entrada implícito y una transición nula. En virtud de este convenio, los dos diagramas indicados en la figura D-3.8.21 son lógicamente equivalentes y tanto da utilizar el uno como el otro.



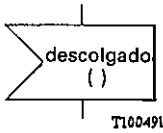
Nota – El valor 9269 se almacena en una variable denominada «número de abonado».

a)



Nota – El valor 9269 se almacena en una variable denominada A.

b)



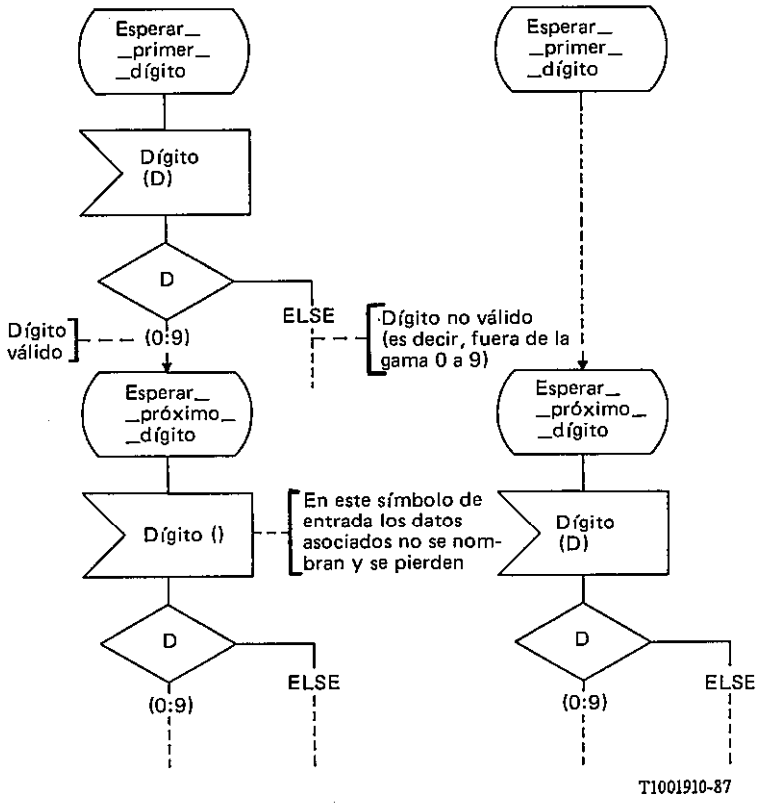
Nota 1 – No hay nombre de datos y el valor 9269 se pierde y no está disponible para el proceso receptor.

Nota 2 – El nombre de la señal (descolgado) debe corresponder al nombre de la señal de salida pero los nombres de datos no necesitan corresponder.

c)

FIGURA D-3.8.18

Ejemplos de recepción de valores en un proceso

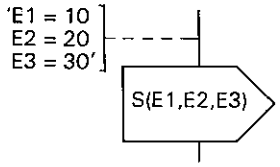


a) Versión incorrecta *b) Versión correcta*

Nota 1 – En *a)*, la segunda decisión utilizará el valor de D obtenido de la primera señal.
Nota 2 – En *b)*, el valor de D será el del dígito actual, que remplazará el valor anterior.

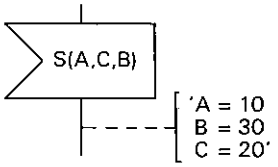
FIGURA D-3.8.19

Parte de un proceso de recepción de dígitos



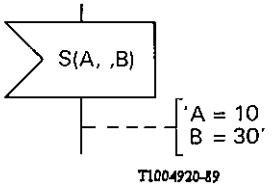
Nota – La señal de salida S tiene tres variables denominadas E1, E2 y E3. Estos elementos se relacionan con tres valores, por ejemplo, actualmente, 10, 20 y 30.

a)



Nota – La señal de entrada correspondiente S en el proceso receptor denomina estos elementos A, C y B respectivamente.

b)

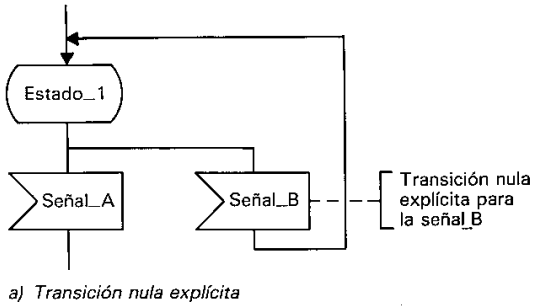


Nota – Esta señal de entrada sólo denomina (nombra) dos variables. El valor intermedio se pierde.

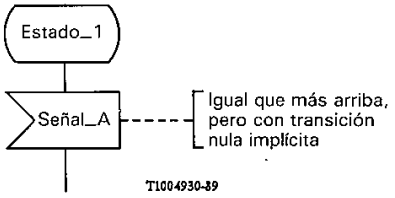
c)

FIGURA D-3.8.20

Señal con varios valores asociados



a) Transición nula explícita



b) Transición nula implícita

Nota – Si hay datos asociados a la señal_B, se pierden en ambos casos. Sin embargo, si se muestra un nombre de datos [por ejemplo, señal B(x)] en el caso *a)*, se conserva el valor del dato. En este caso los dos ejemplos ya no son idénticos.

FIGURA D-3.8.21

Representación explícita e implícita de una transición «nula»

Cuando varias entradas conducen a la misma transición, todos los identificadores de señales de interés pueden hacerse figurar dentro de un solo símbolo de entrada. En LED se proporciona una notación abreviada para representar una entrada de todas las señales (válidas para ese proceso) no mencionadas explícitamente en ese estado. La figura D-3.8.22 ilustra este punto y los diagramas de esta figura son lógicamente equivalentes. Si se mencionan todos los identificadores de señales, deberán estar separados por comas.

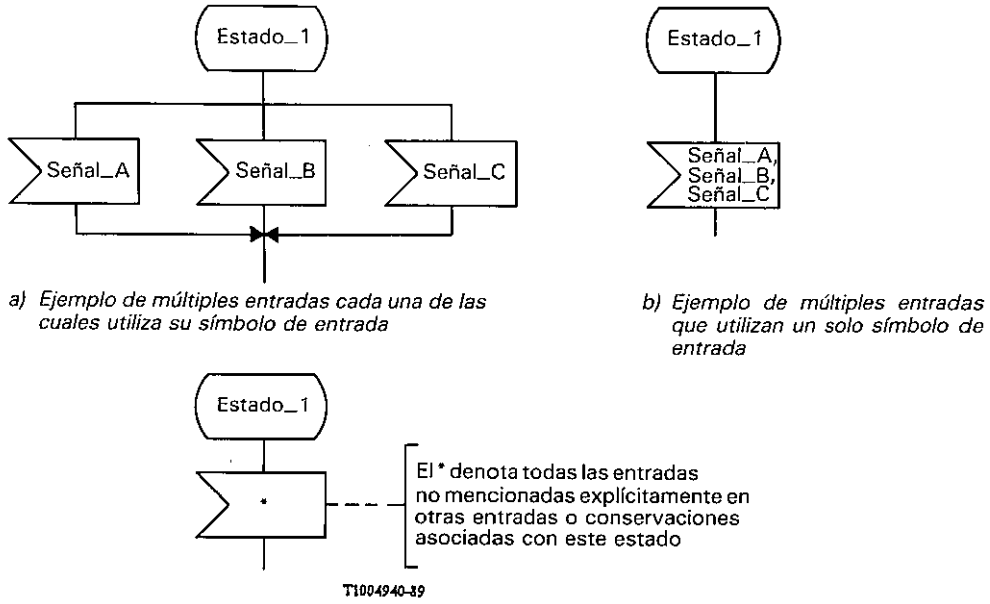
D.3.8.3.2 *Mecanismo implícito de cola de espera*

Puede haber una o más señales en espera de ser consumidas cuando un proceso llega a un nuevo estado. Esto significa que las señales tienen que formar, de alguna manera, una cola de espera. Cuando una señal llega al proceso de destino, se la entrega a su cola de espera de entrada. La semántica del LED define para cada proceso un mecanismo conceptual de cola de espera según el principio de primero en entrar primero en salir, esto es, un proceso que considera las señales, con vista a su consumo, en el orden de llegada de éstas al proceso. Cuando un proceso llega a un estado, se le entrega una y sólo una señal de la cola de espera. Quiere decir que si la cola no está vacía, el proceso consume la primera señal de la cola. Si está vacía, el proceso espera en el estado hasta que llegue una señal a la cola de espera, a continuación de lo cual la misma es consumida por el proceso.

La figura D-3.8.23 utiliza el mecanismo conceptual de cola de espera para explicar la operación del proceso LED representado cuando los tiempos de transición no son nulos. Debe señalarse lo siguiente:

– no se utiliza el concepto de conservación, y por consiguiente las señales son consumidas en el orden de llegada;

– el orden de llegada de las señales es importante. Si «C» hubiera llegado antes que «B» en la transición entre el estado 1 y el estado 2, la secuencia de los estados hubiera sido 1, 2, 3 en lugar de 1, 2, 4;



a) Ejemplo de múltiples entradas cada una de las cuales utiliza su símbolo de entrada

b) Ejemplo de múltiples entradas que utilizan un solo símbolo de entrada

c) Ejemplo de múltiples entradas que utilizan la notación asterisco

FIGURA D-3.8.22

Representación alternativa de múltiples entradas

– como la cola no está vacía cuando el proceso llega a los estados 2 y 4 el proceso no espera en ninguno de estos estados;

– no es posible asignar ninguna prioridad a una señal. Para comunicaciones entre servicios, se proporciona un mecanismo especial, de modo que las señales intercambiadas entre servicios sean tratadas antes que las otras señales (véase el § D.5.3).

Si los tiempos de transición son nulos, toda señal será consumida en el instante en que llega a un proceso, a menos que se utilice la operación de conservación (véase el § D.3.8.4).

D.3.8.3.3 *Recepción de señales que no aparecen normalmente*

En cada estado deben presentarse todas las señales posibles en forma implícita o explícita. Pueden producirse excepciones (señales inesperadas pero teóricamente posibles) en casi todos los estados. Normalmente, el autor no presenta tales posibilidades, y ello conduce a la eliminación de dichas señales cuando llegan. Sin embargo, si el autor desea incluir excepciones en su diagrama, todos los estados deberán ser ampliados con una entrada adicional.

Otra posibilidad consiste en utilizar las múltiples apariciones de un estado junto con el símbolo «todos» (*). Por ejemplo, si puede recibirse la señal A_COLGADO en todos los estados y si las acciones subsiguientes son idénticas, podrá utilizarse el método indicado en la figura D-3.8.24.

D.3.8.3.4 *Llegada simultánea de señales*

En el § 2 de la Recomendación LED se establece que las señales pueden llegar simultáneamente a un proceso y que se ordenarán arbitrariamente.

Si un usuario diseña un proceso que puede captar señales simultáneas, debe cuidar de que el orden de llegada no pueda alterar el funcionamiento deseado del proceso.

El LED no define prioridad de señales, lo que significa que al llegar señales simultáneamente se elige una cualquiera de ellas, arbitrariamente. Obsérvese, sin embargo, que las señales para comunicaciones entre servicios se tratan siempre primero (véase el § D.5.3).

Si hay varias señales disponibles cuando un proceso pasa a un estado, una sola señal es presentada al proceso y reconocida por tanto como entrada. La semántica del LED implica que las otras señales quedan, de hecho, retenidas.

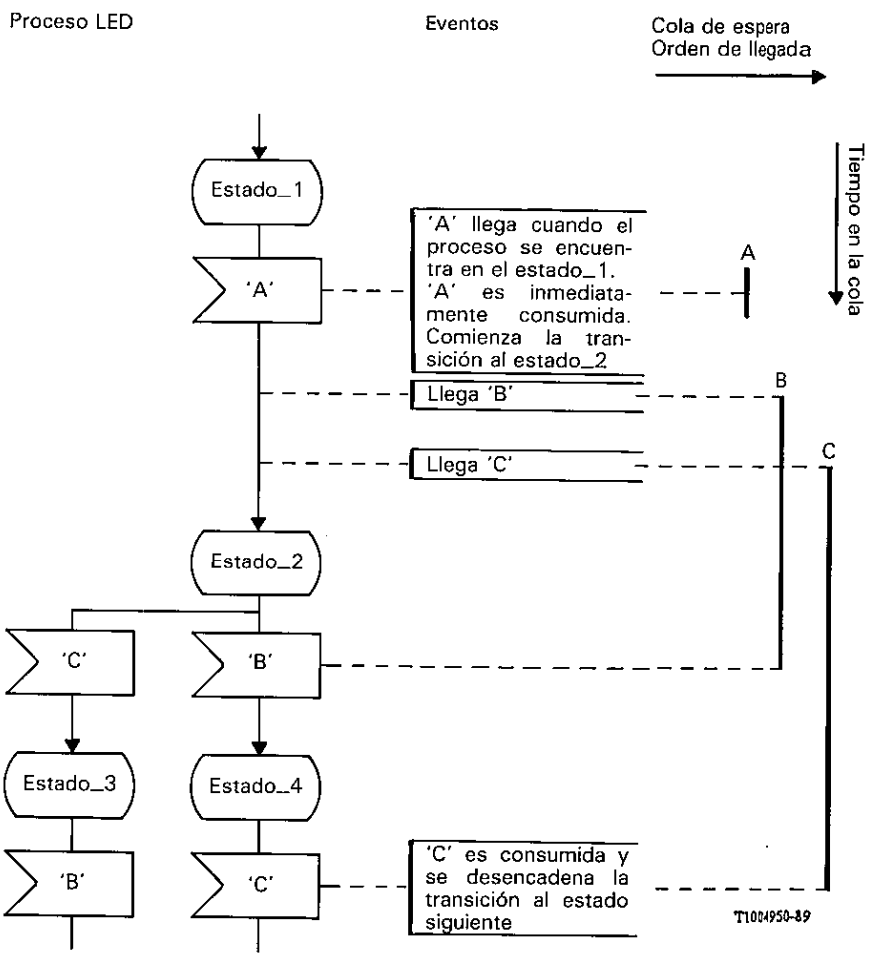


FIGURA D-3.8.23

Ejemplo de la operación del mecanismo implícito de cola de espera

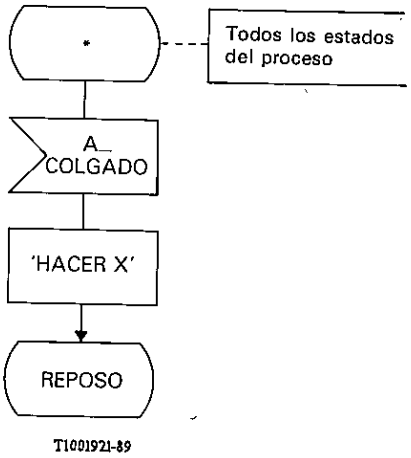


FIGURA D-3.8.24

Ejemplo de tratamiento de señales que pueden aparecer en varios estados

D.3.8.3.5 *Identificación del (proceso) emisor*

Cada señal lleva consigo el valor de instancia de proceso (IPd) del proceso emisor. Cuando se consume una señal, el valor IPd del proceso emisor puede obtenerse por medio de la expresión SENDER (EMISOR). La figura D-3.8.25 muestra un ejemplo de cómo puede utilizarse lo anterior.

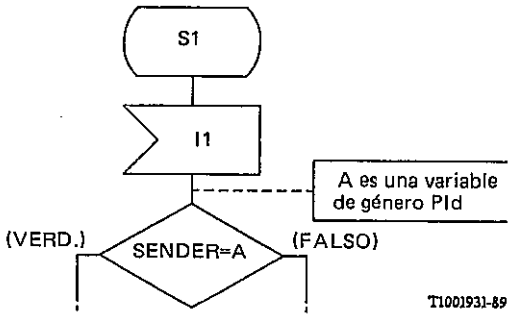


FIGURA D-3.8.25

Uso de la expresión SENDER

D.3.8.4 *Conservaciones*

El concepto de conservación permite aplazar el consumo de una señal hasta que hayan sido consumidas una o más señales llegadas después. Como se indicó en el § D.3.8.3, de no utilizarse el concepto de conservación, las señales son consumidas en el mismo orden en que llegan.

El concepto de conservación puede utilizarse para simplificar procesos cuando el orden relativo de llegada de algunas señales no es importante y el orden de llegada real es indeterminado.

En cada estado, toda señal se trata de una de las tres formas siguientes:

– se indica como una entrada, o

– se indica como una conservación, o

– se trata mediante una entrada implícita que conduce a una transición nula implícita.

La operación del mecanismo implícito de cola de espera presentado en el § D.3.8.3 se aplica también el concepto de conservación. Las señales, al llegar, entran en una cola de espera y, cuando el proceso llega a determinado estado, las señales que están en la cola son examinadas una a una en el mismo orden en que llegaron. Una señal relativa a una entrada explícita o implícita, es consumida y se ejecuta la correspondiente transición. Una señal indicada en una conservación no es consumida y permanece en la cola en la misma posición secuencial que tenía, y se pasa a considerar la señal siguiente en la cola. Ninguna transición sigue a una conservación.

En LED/PR la construcción de conservación se expresa por medio de la palabra clave SAVE seguida por una lista de identificadores de señales. En la figura D-3.8.26 se da un ejemplo sencillo de sentencia SAVE.

```
...
STATE estado_31;
   SAVE s;
   INPUT r;
   NEXTSTATE estado_32;
STATE estado_32;
   INPUT s;
   ...
```

FIGURA D-3.8.26

Ejemplo del uso de la conservación

En LED/GR el concepto de conservación se representa mediante un símbolo de conservación que contiene los identificadores de señales.

De manera similar al caso de las entradas, puede utilizarse una «notación de asterisco» para representar una conservación de todas las señales (válidas para este proceso) no explícitamente mencionadas en ese estado.

La figura D-3.8.27 ilustra un ejemplo de un proceso LED que comprende un símbolo de conservación. Debe observarse que las señales S y R son consumidas en el orden R, S es decir, en el orden inverso a aquél en que se recibieron. Un símbolo único de conservación sólo puede conservar una señal mientras el proceso se encuentra en el estado en que aparece el símbolo, y la conservación durará el tiempo que toma la transición al estado siguiente. En el estado siguiente, la señal será consumida mediante una entrada explícita o implícita (como se indica en la figura D-3.8.27), a menos que se repita el símbolo de conservación con el nombre de la señal, o que haya otra señal conservada disponible para ser consumida antes que ella en la cola implícita (como se ilustra en la figura D-3.8.28).

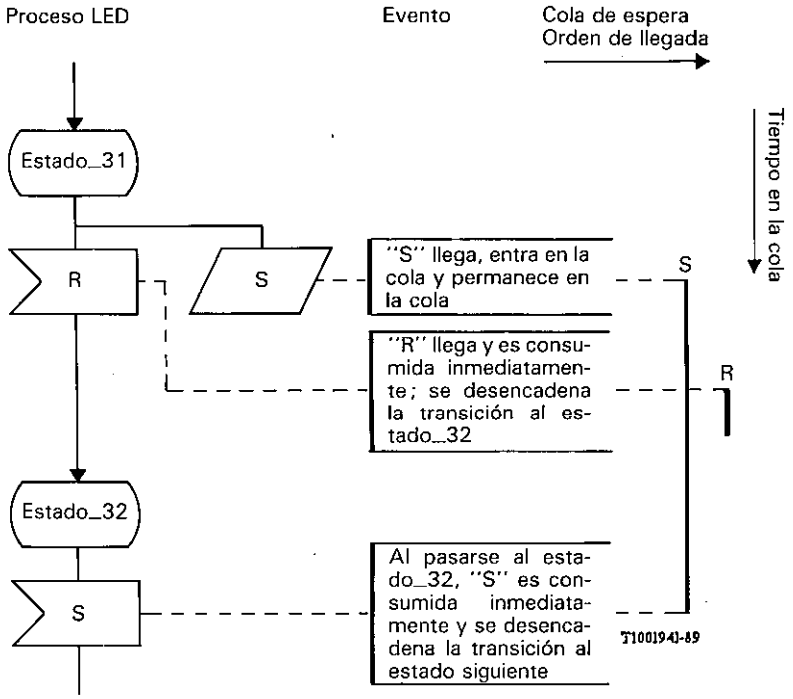


FIGURA D-3.8.27

Ejemplo de un diagrama LED con símbolo de conservación que ilustra la operación del mecanismo de cola de espera implícita

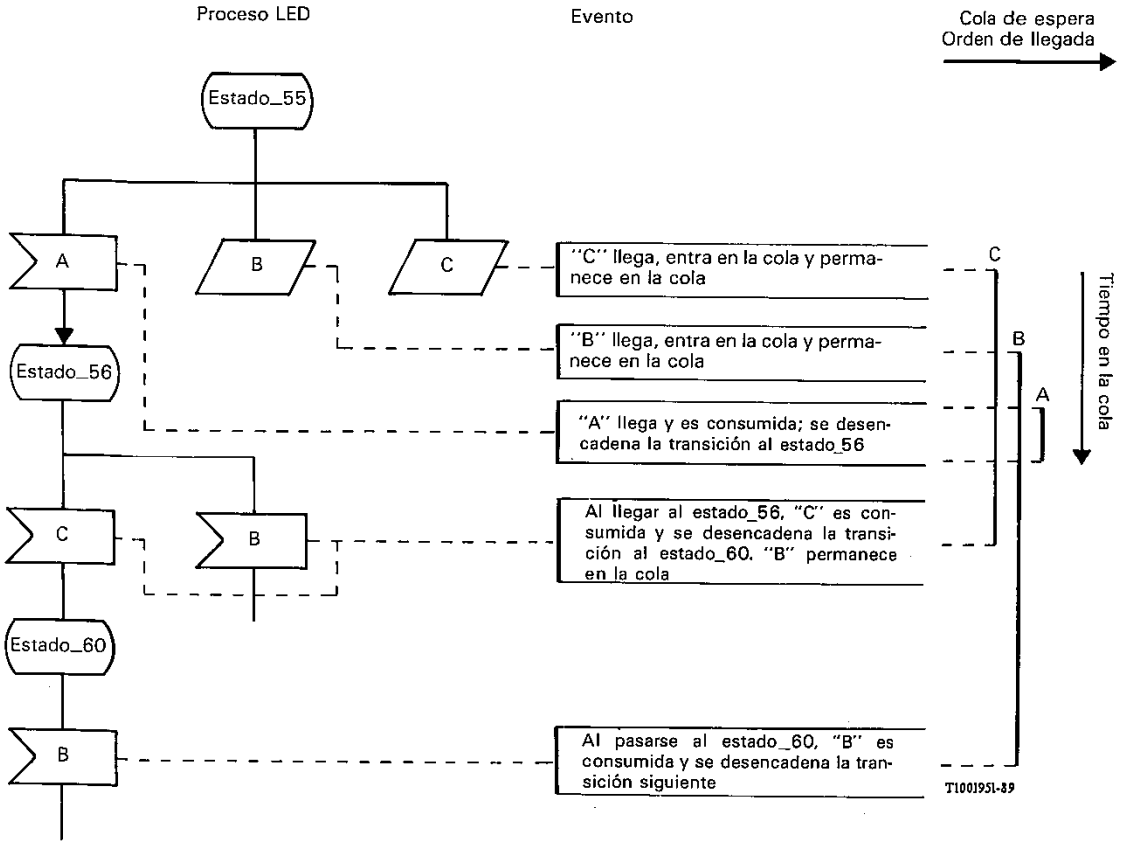


FIGURA D-3.8.28

Segundo ejemplo de utilización del símbolo de conservación

Una señal conservada se pone a disposición de un proceso solamente a través de un símbolo de entrada (explícita o implícita) correspondiente. En particular, en una decisión no pueden hacerse preguntas acerca de una señal conservada antes de su consumo en una entrada, ni tampoco están disponibles los valores asociados.

En un estado en que deba conservarse más de una señal, se puede, o bien emplear un símbolo de conservación para cada señal o inscribir todas las señales en un solo símbolo de conservación. Si deben conservarse varias señales, la semántica del símbolo de conservación implica que se mantiene el orden de llegada.

Un tercer ejemplo del uso de la conservación se ilustra en la figura D-3.8.29; la operación del mecanismo conceptual de cola de espera se describe en la figura D-3.8.30.

La conservación permite simplificar los diagramas. Por ejemplo, conservando una señal es posible evitar su recepción y tener que almacenar sus valores asociados hasta el estado siguiente.

Aunque la conservación puede utilizarse en todos los niveles de especificación, en los niveles inferiores pudiera ser necesario describir el mecanismo real que realiza el concepto de conservación.

Si no se pone cuidado en el uso de la conservación, la fila de señales conservadas puede volverse demasiado grande o mantener señales por demasiado tiempo y podría consumirse entonces una señal antigua cuando lo que se necesitaba era una nueva.

El LED no estipula ningún límite a la longitud de la cola, lo que puede llevar a problemas de realización.

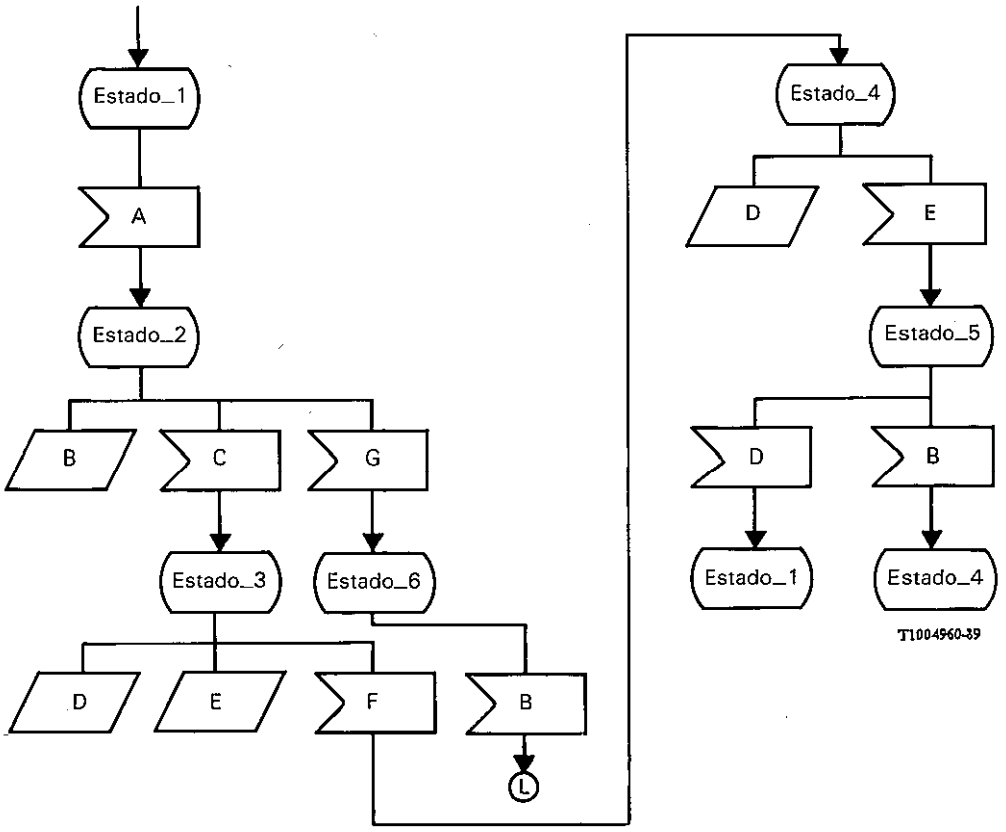


FIGURA D-3.8.29

Ejemplo de un diagrama LED más complejo

Estado vigente	Evento	Cola de espera
Estado_1	(El proceso pasa al estado_1 estando en la cola las señales «A», «B», «C», «D», «E»). La primera señal en la cola, «A», es consumida y se dispara la transición al estado_2	
Estado_2	La primera señal en la cola, «B», aparece en un símbolo de conservación y permanece en la cola	
Estado_2	La segunda señal, «C», es consumida (entrada explícita) y se dispara la transición al estado_3	
Estado_3	La primera señal en la cola, «B», es consumida (entrada implícita)	
	La señal «F» llega y entra en la cola	
Estado_3	(Al pasarse al estado_3 de nuevo) la primera señal en la cola, «D», aparece en un símbolo de conservación y permanece en la cola	
Estado_3	La segunda señal, «E», aparece en un símbolo de conservación y permanece en la cola	
Estado_3	La tercera señal, «F», es consumida (entrada explícita) y se dispara la transición al estado_4	
Estado_4	La primera señal en la cola, «D», aparece en un símbolo de conservación y permanece en la cola	
Estado_4	La segunda señal, «E», es consumida (entrada explícita) y se dispara la transición al estado_5	
Estado_5	La primera (y única) señal en la cola, «D», es consumida (entrada explícita) y se dispara la transición al estado_1	

FIGURA D-3.8.30

Operación del mecanismo de cola de espera

D.3.8.5 *Condiciones habilitadoras y señales continuas*

Las condiciones habilitadoras permiten la recepción condicional de señales, en base a la condición habilitadora especificada. Si la condición es verdadera la señal es consumida y la transición es interpretada. Si la condición es falsa, la señal es conservada y el proceso permanece en el estado hasta que llega otra señal o hasta que la condición cambia de falsa a verdadera. Esto puede ilustrarse mediante el ejemplo de la figura D-3.8.31. Cuando P1 entra en el estado S1 la condición (es decir, si IMPORT(X,P2) es igual a 10, o no) es evaluada. Si la condición es verdadera puede recibirse la señal B. De otra manera, la señal B es conservada. En este ejemplo, llega A y causa una transición a S2. Durante la transición, X ha cambiado a 11 y P2 exporta su nuevo valor; ahora la condición asociada a la señal B, en el estado S2, es verdadera. Ya que B es la primera señal en la cola de espera, la transición que sigue es ejecutada y el proceso termina en el estado S3.

Algunos atributos importantes de las condiciones habilitadoras son los siguientes:

1) la condición habilitadora es probada cuando el proceso llega a un estado y después comprobada continuamente mientras el proceso permanece en el estado. Así, si el valor exportado de X ha cambiado de 9 a 11 y después a 12 durante la ejecución de la transición que sigue a la recepción de A, el proceso habría permanecido en S2;

2) las condiciones habilitadoras pueden basarse en variables locales y/o cualquier construcción de lenguaje que pueda ser incluida en una expresión (por ejemplo IMPORT, VIEW, NOW);

3) aunque es posible utilizar más de una condición habilitadora por estado, no se permite utilizar más de una condición habilitadora para una misma señal. Así pues, la condición mostrada en la figura D-3.8.32 no está permitida. Si hacen falta múltiples condiciones para una determinada señal, las mismas pueden combinarse en una expresión booleana, como se ve en la figura D-3.8.33.

Las condiciones habilitadoras pueden evaluarse varias veces y en cualquier orden por lo que el usuario debe tener cuidado si las expresiones tienen efectos secundarios mutuos (por ejemplo, combinación de IMPORT y SENDER).

Debe hacerse notar que la señal especificada en la condición habilitadora no puede influenciar el valor booleano de la condición porque sus valores transportados no han sido asignados antes del consumo de la señal. Por ejemplo, las sentencias:

```
INPUT x(A) PROVIDED (A=5);...

INPUT y PROVIDED (SENDER=pid1);
```

son engañosas pues los valores de A y de SENDER en las condiciones corresponden a la situación antes del consumo de la señal.

Las señales continuas tienen las mismas propiedades básicas que las condiciones habilitadoras, con la excepción de que no tienen ninguna señal asociada. Así, al entrarse al estado, sin ninguna señal en la cola de espera que pueda causar una transición, se verifican las señales continuas y, si una es verdadera, se ejecuta la transición que le sigue. Esto puede ilustrarse con el ejemplo de la figura D-3.8.34. Inicialmente el proceso está en el estado S1 y el valor exportado de X es 9. Cuando llega la señal A, causa la transición a S2. Durante la transición, X cambió a 11 y su nuevo valor es exportado. Como no había ninguna otra señal en la cola de espera, se ejecuta la transición a S3.

Algunos atributos importantes de las señales continuas son los siguientes:

1) al igual que para las condiciones habilitadoras, el valor de la condición se verifica solamente cuando el proceso llega a un estado o se encuentra en él;

2) se permiten señales continuas múltiples para cada estado. Cuando hay más de una señal continua asociada a un estado, se probará primero la señal continua con prioridad más alta (número de prioridad más bajo). Dos señales continuas, cualesquiera que sean, no pueden tener el mismo número de prioridad. En todos los casos, la prioridad de la señal continua es más baja que la de cualquier otra señal. Una vez más, esto procede del sistema subyacente del LED; no obstante, la manera en que se modelan las señales continuas en el LED, mediante el empleo de las señales enviadas al exportar variables, permite utilizar prioridades para las señales continuas y, de hecho, lo hace necesario a fin de impedir ambigüedades cuando hay más de una señal continua presente. Esto se ilustra en la figura D-3.8.35. Inicialmente el proceso está en el estado S1 y sus expresiones importadas dan los valores 10 para X y 11 para Y. Como ambas señales continuas son verdaderas, se elige la que tiene la prioridad más alta (número de prioridad más bajo) y se ejecuta la transición a S2. En S2, la condición Y ya no es verdadera y por ello, aunque la prioridad de la señal continua para X es menor que la correspondiente a Y, se efectúa la transición que le sigue y el proceso llega al estado S3;

3) cuando se sigue la transición desde una señal continua, la expresión SENDER devuelve el mismo valor de SELF.

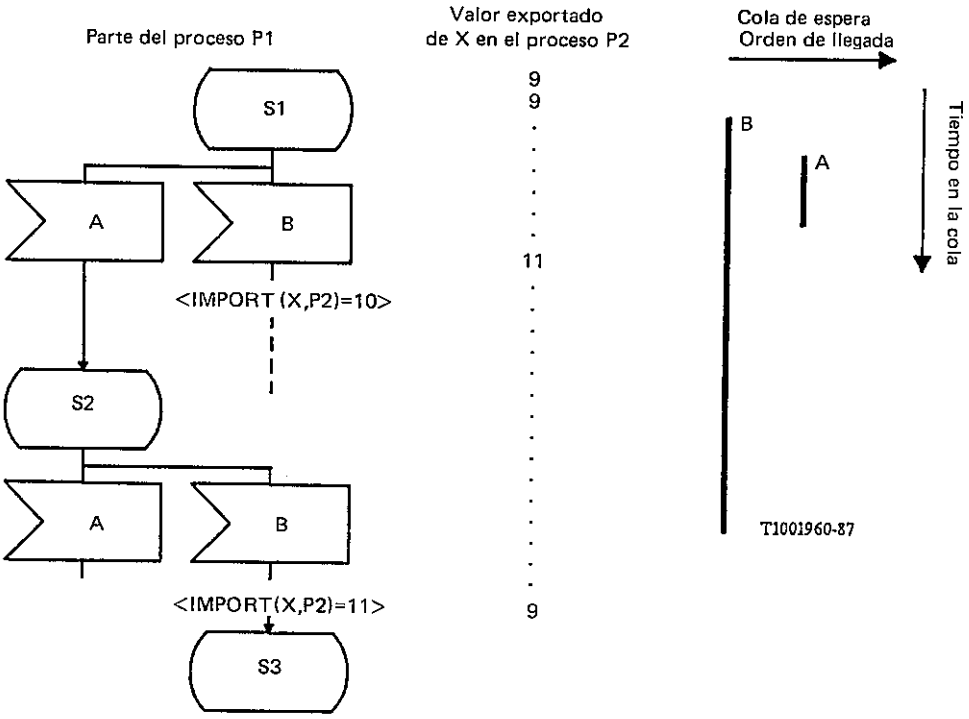


FIGURA D-3.8.31

Condición habilitadora

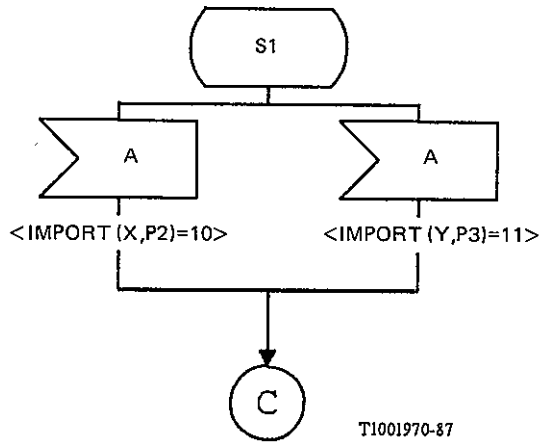


FIGURA D-3.8.32

Condición habilitadora ilegal

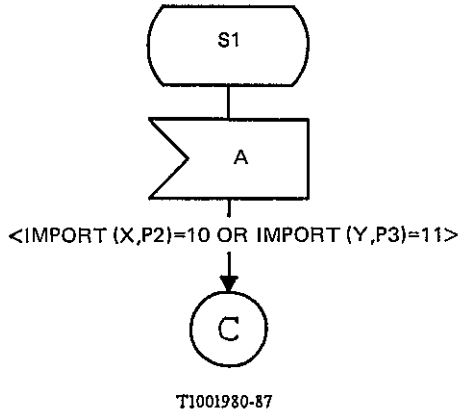


FIGURA D-3.8.33

Solución correcta para la figura D-3.8.32

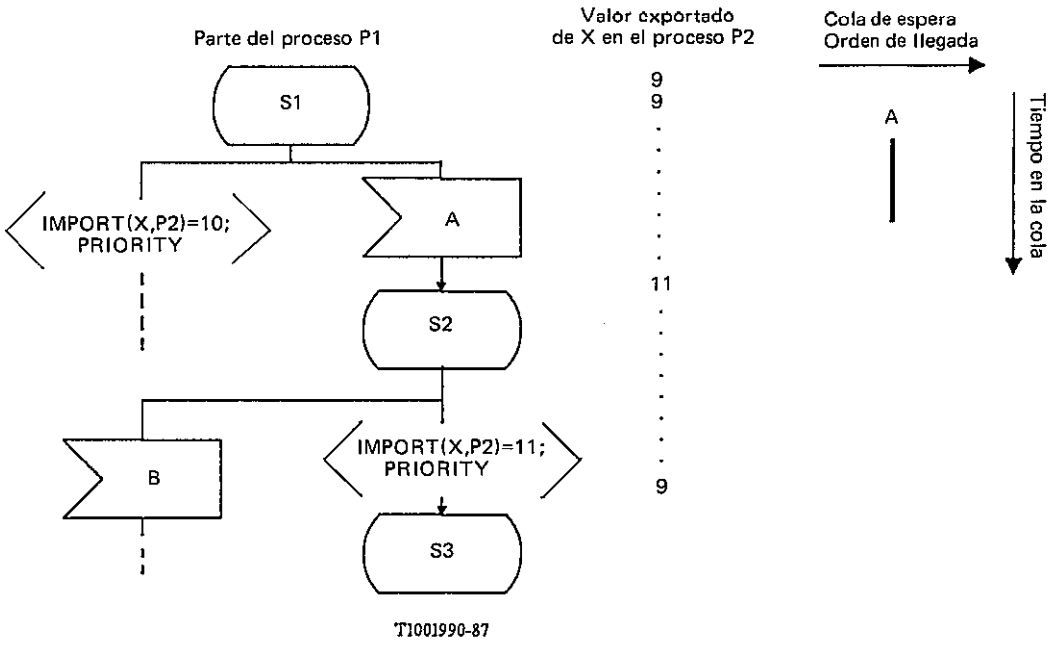


FIGURA D-3.8.34

Señales continuas

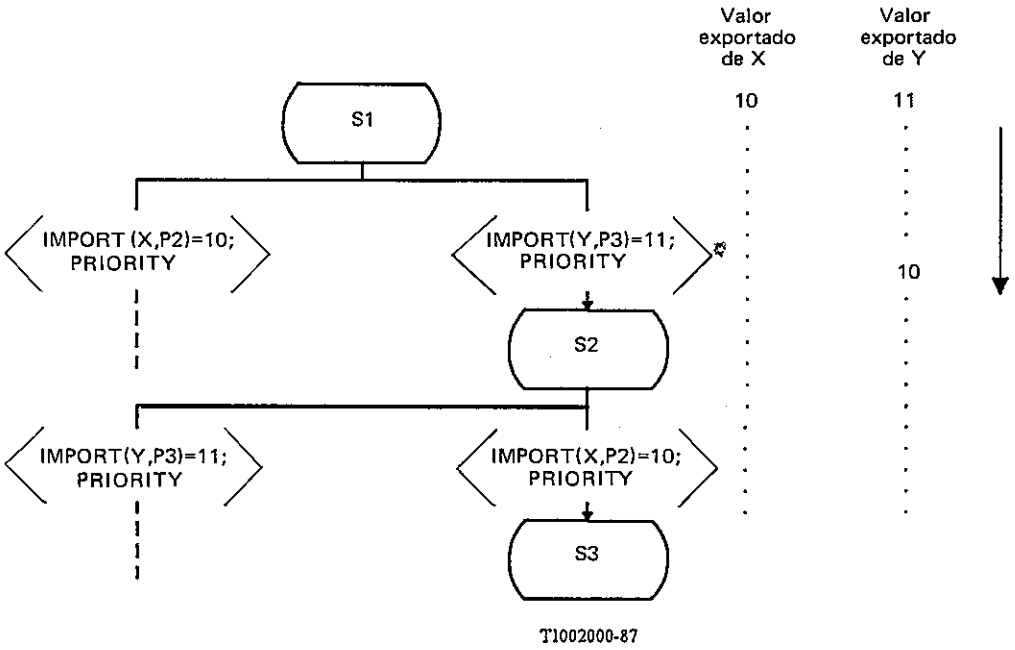


FIGURA D-3.8.35

Señales continuas con prioridad

D.3.8.6 *Salidas*

Una salida es el envío de una señal de un proceso a otro proceso (o a sí mismo). Puesto que el control de la recepción y del consumo de la señal está asociado al proceso receptor (véase el § D.3.8.3), la semántica relacionada directamente con la salida es relativamente sencilla. Desde el punto de vista del proceso emisor, una salida puede considerarse a menudo como una acción instantánea que, una vez completada, no ejercerá influencia ulterior alguna en el proceso emisor, el cual no tendrá por qué conocer directamente el destino de la señal.

Una acción de salida representa el envío de una señal y la asociación de valores, si existen. Los valores pueden asociarse con una señal de salida colocando los valores entre paréntesis o colocando expresiones que tienen valores entre paréntesis (véase la figura D-3.8.37).

En LED/PR una acción de salida se representa por medio de la palabra clave OUTPUT seguida por una lista de identificadores de señal. Puede asociarse una lista de parámetros efectivos entre paréntesis a cada identificador de señal. Si no hay parámetro efectivo en la salida correspondiente a cierto género en la definición de señal, la variable correspondiente en la entrada que recibe tendrá el valor «indefinido».

La instancia de proceso de destino debe expresarse en la sentencia de salida por la palabra clave TO seguida de una expresión de instancia de proceso. Si la instancia de proceso de destino puede determinarse en forma exclusiva por el contexto, la cláusula TO puede omitirse. Puede proporcionarse una condición adicional de direccionamiento en la sentencia de salida por medio de la palabra clave VIA seguida por una lista de rutas de señales o de identificadores de canal.

La figura D-3.8.36 muestra algunos ejemplos válidos de sentencias de salida.

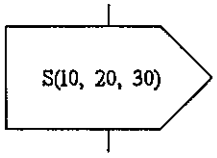
```
OUTPUT señ1(2,verdadero,10);

OUTPUT señ2, señ3(par1,par2) TO proceso_a;

OUTPUT señ4 TO proceso_b VIA canal_x,canal_y;

OUTPUT señ5 VIA ruta_de_señ_z;
```

FIGURA D-3.8.36

Ejemplos de sentencias de salida

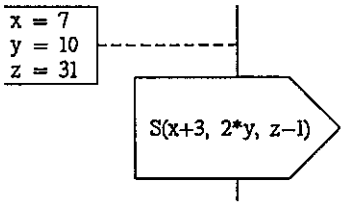
En LED/GR una salida se representa por medio de un símbolo de salida que contiene la especificación de las señales, parámetros efectivos y, facultativamente, destino y/o construcción VIA.

Cada salida debe estar dirigida a una instancia de proceso específica. Como suele ser imposible conocer la instancia de proceso de ningún proceso en el momento de establecer una especificación, el método normal de direccionamiento de señales consiste en utilizar una variable o expresión con la palabra clave TO. Las figuras D-3.8.38, D-3.8.39 y D-3.8.40 muestran algunos ejemplos. En la figura D-3.8.38, el parámetro de proceso salida_a recibe el valor de una instancia de proceso en el momento en que se crea el proceso. Se utiliza entonces salida_a dentro del proceso como enlace entre este proceso y su proceso conectado. Al diseñar el sistema debe ponerse cuidado en garantizar que el tipo de proceso indicado por salida_a sea capaz de recibir las señales enviadas. En la figura D-3.8.39 la expresión incorporada SENDER se usa para retornar una señal al proceso que envió la señal. En la figura D-3.8.40 la señal es direccionada al proceso vástago (denominado también descendiente) más recientemente creado del proceso.



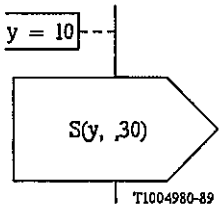
Nota – La señal S tiene tres valores, 10, 20 y 30 asociados a ella.

a)



Nota – Al interpretar la salida: x, y y z tienen (en este ejemplo) los valores 7, 10 y 31 respectivamente. La salida envía los valores 10, 20 y 30.

b)



Nota – Al interpretar la salida, y tiene (en este ejemplo) el valor 10. La salida envía el valor 10, un valor indefinido y 30, respectivamente.

c)

FIGURA D-3.8.37

Salida con valores asociados

```
PROCESS X (0,5);
   FPAR (salida_a PID);
   . . .
   OUTPUT señ TO salida_a;
   . . .
```

FIGURA D-3.8.38

Direccionamiento de señales mediante parámetros formales

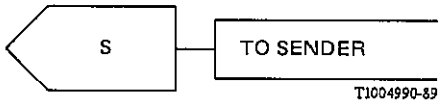


FIGURA D-3.8.39

Direccionamiento de una señal de retorno al proceso emisor

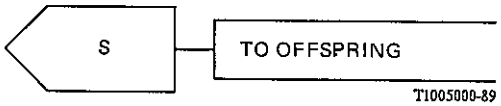


FIGURA D-3.8.40

Direccionamiento de una señal a un proceso vástago

D.3.8.7 *Tarea*

Una tarea se utiliza en una transición para representar operaciones sobre variables o para representar operaciones especiales por medio de texto informal.

En LED/PR una tarea se representa por medio de la palabra clave TASK seguida por una lista de sentencias o textos informales separados por comas y terminados por un punto y coma. Una sentencia en una tarea puede ser solamente una asignación. Un texto informal consiste en una frase delimitada por apóstrofos. En la figura D-3.8.41 se incluyen algunos ejemplos de tareas válidas en LED/PR.

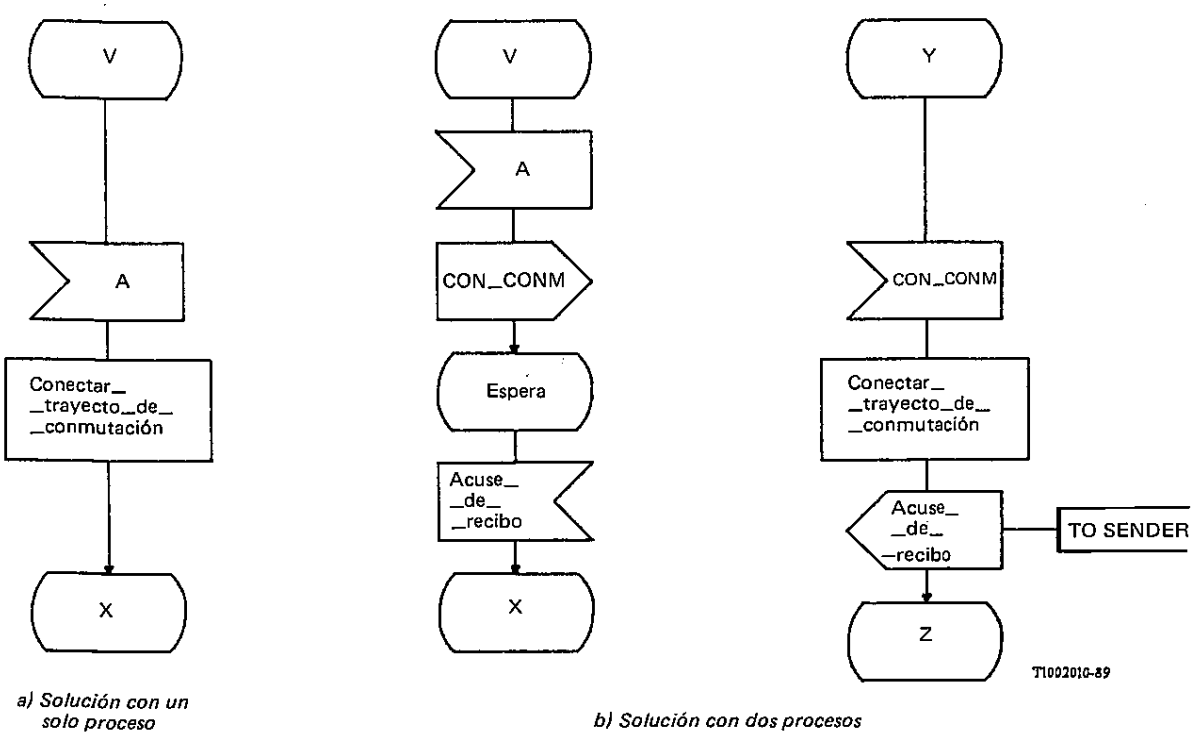
```
TASK a:=b;

TASK 'conectar al abonado';

TASK c:=d+e;

TASK var1:=var2*var3,
     var4:=var5 MOD var6;
```

FIGURA D-3.8.41

Ejemplos de tareas

En LED/GR una tarea consiste en un símbolo de tarea que contiene la lista de sentencias o textos informales.

Los usuarios del LED pueden experimentar dificultades a la hora de decidir si algunos aspectos del sistema que se define deben representarse por una tarea o por un proceso separado. Considérese el proceso mostrado en la figura D-3.8.42; ¿la acción «conectarrayecto_de_conmutación» debe representarse como una tarea o como un proceso separado? Si no se ha identificado un proceso separado de control del trayecto de conmutación, el símbolo de tarea sería apropiado [figura D-3.8.42 a)]. Si se ha identificado tal proceso separado, hay que utilizar señales que comuniquen con el proceso de control [figura D-3.8.42 b)].



a) Solución con un solo proceso

b) Solución con dos procesos

FIGURA D-3.8.42

Dos posibles soluciones para «conectar_trayecto_de_conmutación»

D.3.8.8 *Decisiones*

Una decisión es una acción, dentro de una transición, en virtud de la cual se hace una pregunta respecto al valor de una expresión en el instante de ejecutar la acción. El proceso continúa por uno de los dos o más trayectos que siguen a la decisión, de acuerdo con la respuesta. Los autores de diagramas LED deben cerciorarse de que los procesos queden definidos de tal modo que no puedan tratar de ejecutar decisiones para las cuales no se dispone de respuesta; tales decisiones invalidarían el diagrama y podrían causar una confusión considerable.

La pregunta de una decisión puede ser una expresión o un texto informal. Las respuestas de una decisión están representadas por uno o más valores posibles obtenidos por la evaluación de la expresión en la pregunta o por uno o más textos informales. Si la pregunta, o una de las respuestas, es informal, entonces la decisión completa es informal. Las respuestas diferentes están separadas por comas. Los valores están representados por expresiones constantes, por expresiones constantes con un operador como prefijo, o por gamas cuyos límites superiores e inferiores son expresiones constantes. Los valores de las respuestas tienen que ser del mismo género que la expresión contenida en la pregunta.

Es posible indicar algunas respuestas explícitamente y agrupar todas las otras respuestas posibles utilizando la palabra clave ELSE.

En LED/PR la decisión está representada por la palabra clave DECISIÓN seguida por la especificación de la pregunta y por la lista de respuestas posibles cada una asociada con la transición correspondiente. Las respuestas se indican entre paréntisis. El conjunto de transiciones salientes está delimitado al final por la palabra clave ENDDECISION (véase la figura D-3.8.43).

```
DECISION pregunta;
   (respuesta_a): . . .
   (respuesta_b): . . .
   (respuesta_c): . . .
   ELSE: . . .
ENDDECISION;
```

FIGURA D-3.8.43

Esqueleto de una decisión

En la figura D-3.8.44 se muestran algunos ejemplos de decisiones.

```
. . .
DCL x INTEGER,a BOOLEAN;
. . .
DECISION x;
   (2): . . .;
   (2+5): . . .;
   (6+8,10+9): . . .;
   (=3): . . .;
   (20:30): . . .;
   (> =100): . . .;
   ELSE: . . .;
ENDDECISION;
. . .
DECISION a;
   (TRUE): NEXTSTATE s1;
   (FALSE): NEXTSTATE s2;
ENDDECISION;
. . .
DECISION 'Categoria del abonado':
   ('Internacional', 'Nacional'): . . .
   ('Local'): . . .
ENDDECISION;
. . .
```

FIGURA D-3.8.44

Ejemplos de decisiones en LED/PR

Todas las transiciones terminan con la palabra clave ENDDECISION. Aquellas transiciones que no se terminan con una sentencia de terminación (es decir unión (JOIN), estado-siguiente (NEXTSTATE), parada (STOP)) continúan en la sentencia que sigue a ENDDECISION, como se muestra en las dos bifurcaciones equivalentes de la figura D-3.8.45.

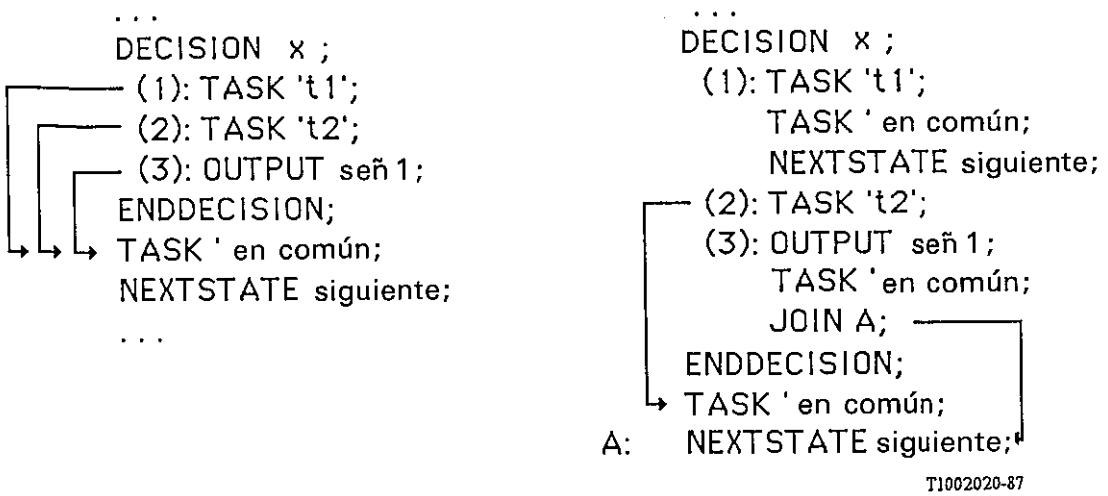


FIGURA D-3.8.45

Bifurcaciones equivalentes de una decisión

La sentencia de decisión también puede utilizarse para modelar las estructuras IF-THEN, DO-WHILE y LOOP-UNTIL como en la programación estructurada.

En LED/GR una decisión se representa por medio de un símbolo de decisión que contiene el texto de la pregunta. El símbolo debe tener dos o más bifurcaciones asociadas con las respuestas correspondientes. Cada respuesta debe estar colocada a la derecha o encima de la rama correspondiente o también sobre la rama interrumpe la línea de flujo. En LED/GR los paréntesis para delimitar las respuestas son facultativos pero se sugiere utilizarlos para evitar malentendidos.

En la figura D-3.8.46 se muestran algunos ejemplos de decisiones en LED/GR.

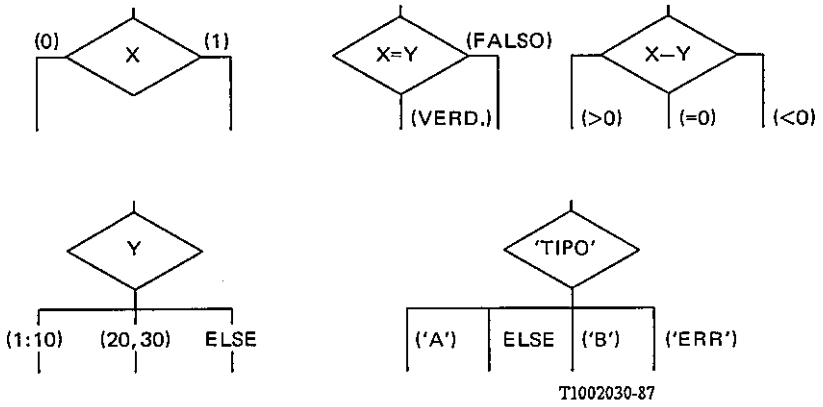


FIGURA D-3.8.46

Ejemplos de decisiones en LED/GR

Si una respuesta hace retornar a la decisión en la misma transición, deben ejecutarse algunas acciones que influencien la pregunta en la decisión. Sin embargo, aun con esta regla podría crearse un número infinito de bucles según se muestra en la figura D-3.8.47. Por lo tanto, siempre debe tenerse especial cuidado cuando hay respuestas que hacen retornar a una decisión en la misma transición.

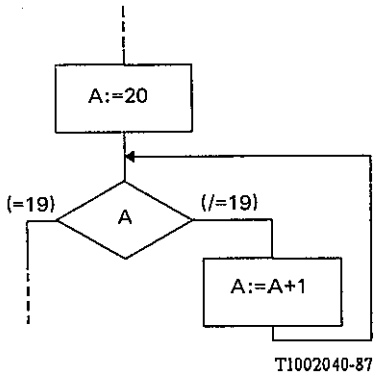


FIGURA D-3.8.47

Ejemplo de uso legal de una decisión que crea un bucle infinito

Las decisiones pueden hacerse utilizando cualquiera de los valores disponibles para el proceso en ese momento, incluyendo:

– valores recibidos por una entrada;

– valores pasados como parámetros efectivos en el momento de creación del proceso;

– valores compartidos.

La expresión en la pregunta puede incluir constantes y cualquiera de las clases de valores arriba indicadas.

D.3.8.9 *Uniones y conectores*

Las uniones permiten transferir el control de un punto a otro de un cuerpo de procedimiento (como también dentro de un cuerpo de procedimiento o dentro de un cuerpo de servicio).

En LED/PR ellas son equivalentes a sentencias «GO TO». Se utilizan etiquetas como puntos de entrada asociados a sentencias como se ilustra en la figura D-3.8.48. Dentro de un cuerpo de proceso (o cuerpo de procedimiento) no es posible transferir el control (y por lo tanto asociar etiquetas) al tipo de sentencias que se muestra en la figura D-3.8.49. Las etiquetas son siempre locales al proceso, por lo tanto, no es posible transferir el control de un proceso a otro por medio de una unión.

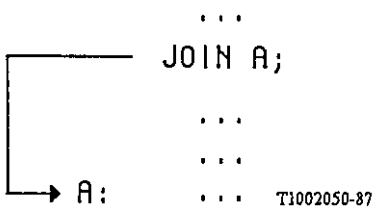


FIGURA D-3.8.48

Etiqueta

STATE
ENDSTATE,
INPUT,
SAVE,
ENDDECISION

FIGURA D-3.8.49

Puntos en que no se admiten etiquetas

En LED/GR las uniones corresponden a conectores (conectores de salida y conectores de entrada). Pueden utilizarse para dividir los diagramas, debido a la falta de espacio, o también para evitar el cruce de las líneas de flujo que podrían hacer los diagramas algo confusos. Además, normalmente es preferible dibujar un diagrama LED en que el flujo circula de la parte superior a la parte inferior de la página.

En GR cualquier línea de flujo puede ser interrumpida por un par de conectores asociados, considerándose que el flujo va del conector de salida al conector de entrada. Cada símbolo de conector contiene un nombre, los conectores asociados tienen el mismo nombre. Para cada nombre existe solamente un conector de entrada pero puede haber uno o más conectores de salida.

En GR es deseable que la referencia de página del conector de entrada apropiado se especifique para el conector de salida, y que la(s) referencia(s) de página del (de los) conector(es) de salida apropiado(s) se especifiquen para el conector de entrada. (Véase ejemplo en la figura D-3.8.50.)

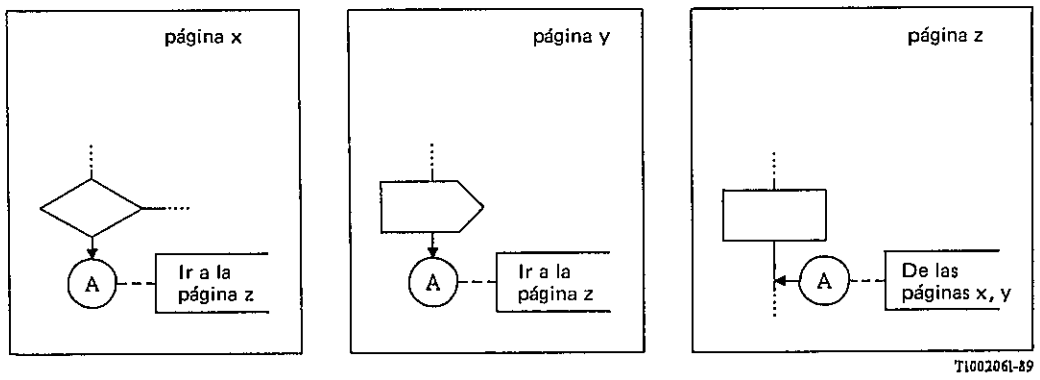


FIGURA D-3.8.50

Referencias de página para conectores

D.3.9 *Procedimientos*

Los procedimientos en LED son similares a los procedimientos en CHILL y otros lenguajes de programación. El propósito de los procedimientos es:

a) permitir la estructuración de un proceso en varios niveles de detalle;

b) mantener la compacidad de las especificaciones permitiendo que una reunión compleja de ítems, que pueden tomarse como aislados, se represente por un solo ítem;

c) permitir que grupos de ítems usados con frecuencia se definan y utilicen repetidamente.

Una definición de procedimiento sólo puede estar contenida en una definición de proceso, en una definición de servicio o en una definición de procedimiento. Por lo tanto, un procedimiento es visible solamente al proceso o procedimiento en el que está definido.

Una definición de procedimiento consiste en los siguientes puntos (algunos de los cuales son facultativos):

– nombre del procedimiento;

– parámetros formales de procedimiento: una lista de nombres de variables asociadas con sus géneros. Se utilizan para transferir información a/desde el procedimiento en el momento de la llamada. Los parámetros de procedimiento pueden pasarse por valor (parámetro IN) o por referencia (parámetro IN/OUT). Si el parámetro es pasado por valor, la especificación del parámetro formal define una variable local al procedimiento; si es pasado por referencia, la especificación define un sinónimo para la variable;

– definiciones de procedimientos: procedimientos que pueden ser llamados solamente por el procedimiento mismo;

– definiciones de datos: la especificación de tipos de datos locales al procedimiento;

– definiciones de variables: variables locales dentro del procedimiento;

– cuerpo del procedimiento: la especificación del comportamiento real del procedimiento en términos de estados y acciones (en forma similar al cuerpo del proceso).

En la figura D-3.9.1 se da un ejemplo parcial de definición de procedimiento en LED/PR (las palabras clave del lenguaje están en mayúsculas). Nótese que los parámetros formales que no tienen atributos explícitos tienen un atributo IN implícito (var5 en la figura).

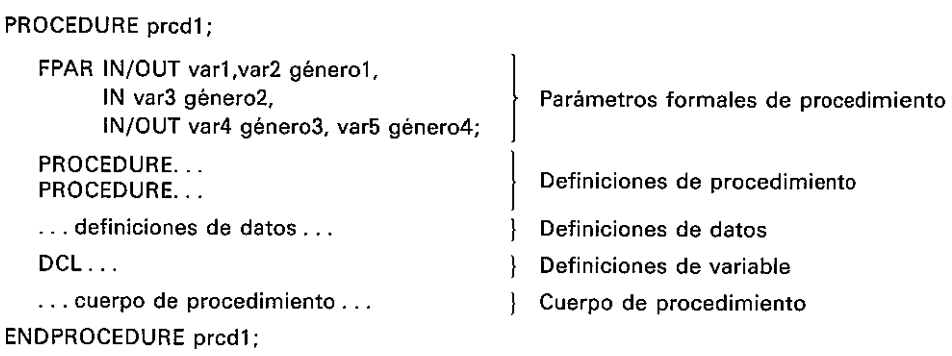


FIGURA D-3.9.1

Ejemplo de definición parcial de procedimiento en LED/PR

D.3.9.1 *Cuerpo del procedimiento*

El cuerpo del procedimiento es muy similar al cuerpo del proceso, con las diferencias siguientes:

– El procedimiento termina su interpretación con un «retorno» en vez de una «parada». En LED/PR la sentencia de retorno está representada por la palabra clave RETURN.

– En LED/GR el símbolo de arranque del procedimiento es ligeramente diferente del símbolo de arranque del proceso.

Los símbolos de arranque y de retorno de procedimiento pueden encontrarse en el resumen de la sintaxis LED/GR.

Un procedimiento puede usar la construcción de unión, pero solamente para referir una etiqueta dentro de sí mismo. La unión no puede ser usada para entrar a un procedimiento desde el exterior, ni para salir de él.

En LED/GR una definición de procedimiento se representa por medio de un diagrama de procedimiento que es muy similar al diagrama de proceso. Un diagrama de procedimiento consiste en los siguientes puntos:

- Un símbolo de casilla facultativo: un símbolo de forma rectangular que contiene todos los otros símbolos.
- El encabezamiento del procedimiento: la palabra clave PROCEDURE seguida por el nombre del procedimiento y por la especificación de los parámetros formales del procedimiento. Generalmente el encabezamiento del procedimiento se coloca en la esquina superior izquierda de la casilla o, si no hay casilla, en la esquina superior izquierda del medio en que el diagrama está dibujado.
- Numeración opcional de páginas. (Colocada en la esquina superior derecha).
- Símbolos de texto: en el caso de un diagrama de procedimiento, el símbolo de texto puede utilizarse para contener la especificación de los parámetros formales, datos y definiciones de variables.
- Referencias de procedimiento: símbolos de procedimiento, que contienen cada uno un nombre de procedimiento que representa un procedimiento local definido separadamente.
- Diagramas de procedimiento: para definir directamente procedimientos locales.
- La zona de gráfico de procedimiento: la especificación del comportamiento del procedimiento en términos de arranque, estados, entradas, salidas, tareas, . . . y arcos dirigidos.

En la figura D-3.9.2 hay un ejemplo de definición de procedimiento en LED/GR. El procedimiento referenciado «P_TERM» en el ejemplo es local al procedimiento llamante.

Como se indicó en el caso de diagramas de proceso (§ D.3.8), si no hay suficiente espacio en una sola página para un diagrama de procedimiento, el diagrama puede representarse en varias páginas repitiendo el símbolo de casilla con el encabezamiento y el número de la página.

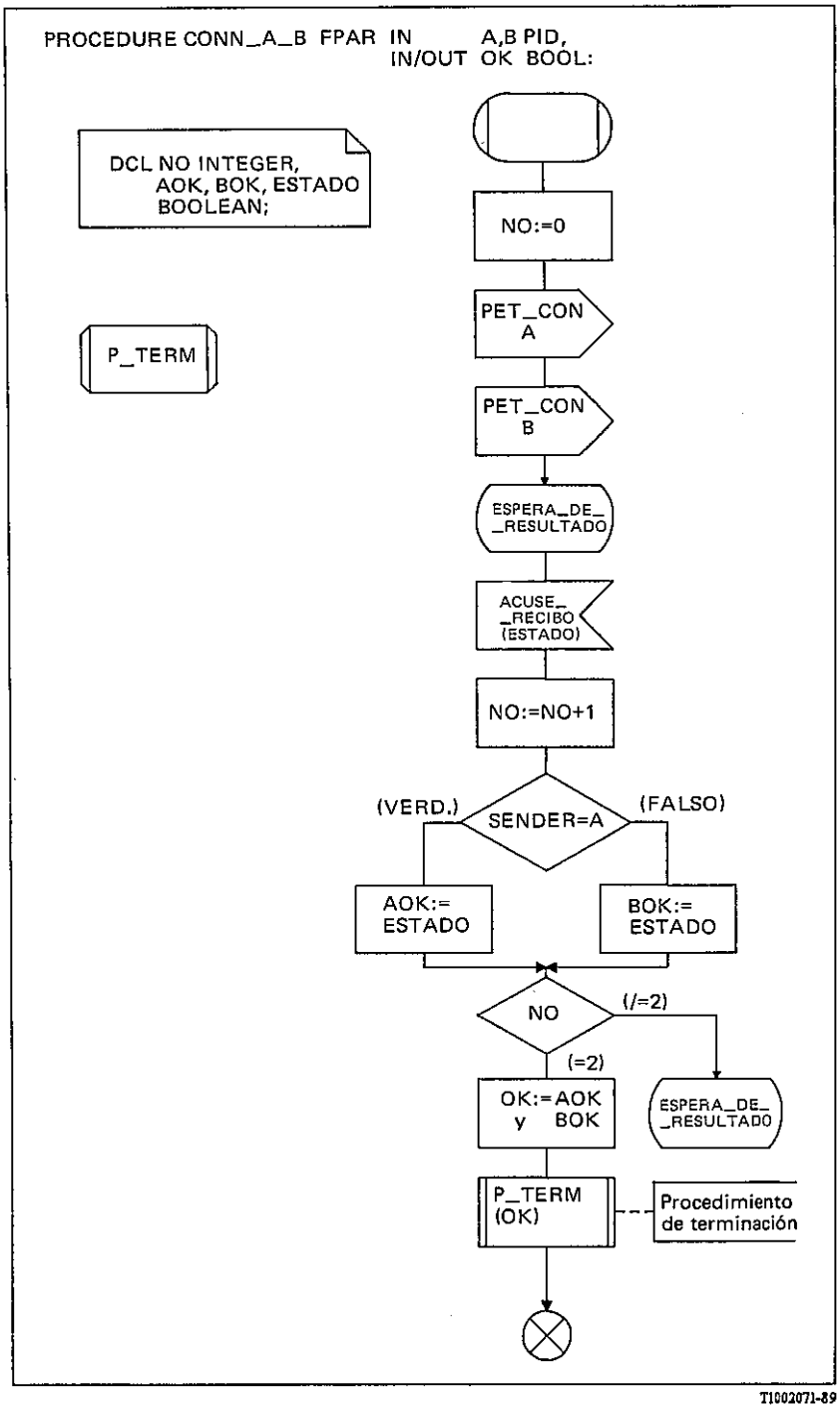


FIGURA D-3.9.2

Ejemplo de diagrama de procedimiento

D.3.9.2 *Llamada de procedimiento*

Las llamadas de procedimiento pueden producirse dondequiera que se admita una tarea, sea en un proceso o en un gráfico de procedimiento. En cierto sentido, un procedimiento puede interpretarse como una tarea con las siguientes excepciones:

1) un procedimiento puede contener estados y, si es así, recibirá señales;

2) un procedimiento puede enviar señales. La instancia de proceso de origen es aquélla que llamó el procedimiento.

Cuando se llama un procedimiento, el entorno del procedimiento es creado y el procedimiento comienza a ser interpretado. La interpretación del procedimiento continúa hasta que se llega al RETURN. Mientras el procedimiento está siendo interpretado, todas las señales dirigidas al proceso son, o conservadas implícitamente, o tratadas explícitamente por el procedimiento. El procedimiento no tiene su propia cola de entrada, sino que utiliza la cola de entrada del proceso que lo llamó.

En LED/PR una llamada de procedimiento se representa por la palabra clave CALL seguida por el identificador de procedimiento y la lista de los parámetros efectivos colocados entre paréntesis. Si un parámetro no se da, debe indicarse por medio de dos comas consecutivas. En este caso, el parámetro formal correspondiente tiene el valor «indefinido». Nótese también que la declaración de IN, o de IN/OUT se hace en la definición del procedimiento, de modo que no debe ser repetida por la sentencia llamante. En la figura D-3.9.3 se ilustran algunos ejemplos de llamada en LED/PR.

```
CALL proced1;
CALL proced2(var1,var2);
CALL proced3 (5,a+b,Pid1);
CALL proced4 (W,X,,Z);
```

FIGURA D-3.9.3

Ejemplo de sentencias de llamada en LED/PR

En LED/GR una llamada de procedimiento se representa por medio de un símbolo de llamada de procedimiento que contiene el nombre del procedimiento y la lista de los parámetros efectivos colocados entre paréntesis. En la figura D-3.9.2 se proporciona un ejemplo de llamada en LED/GR.

D.3.10 *Tratamiento de datos*

D.3.10.1 *Declaraciones de variable*

Las variables son locales con respecto a una instancia de proceso, lo que significa que cada variable pertenece a una y solamente a una instancia de proceso. Solamente la instancia de proceso a que pertenecen puede cambiar el valor de las variables.

Las variables declaradas en la figura D-3.10.1, son locales con respecto a cada instancia del proceso P y, por lo tanto, sólo pueden ser accesibles y modificadas por cada instancia del proceso P (cada instancia del proceso puede tener acceso a su propia copia de variables o modificarla).

```
SYSTEM S;
   ...
   BLOCK B;
   ...
      PROCESS P(3,10);
         DCL
         A Integer,
         D Integer,
         ...
         ...
      ENDPROCESS P;
   ENDBLOCK B;
   ...
ENDSYSTEM S;
```

FIGURA D-3.10.1

Ejemplo de declaración de variable

Una variable puede ser inicializada directamente después de la declaración, según se indica en la figura D-3.10.2.

```
DCL A Integer :=1;
```

FIGURA D-3.10.2

Inicialización de variable

El LED admite dos modos de inicializar las variables. Es posible declarar un valor inicial para todas las variables de cierto género utilizando la sentencia DEFAULT en la definición del tipo de datos (véase el § D.6.4.5). En este caso la inicialización es válida para todas las variables de ese género.

Por otra parte, es posible inicializar cada variable de cierto género con un valor como se muestra en la figura D-3.10.2. Si hay tanto una sentencia DEFAULT como una inicialización en la declaración, entonces esta última prevalece. Si una variable no es inicializada, su valor inicial se considera «indefinido» en el sistema.

Naturalmente, la notación abreviada de inicialización de variables ilustrada en la figura D-3.10.2 es utilizable solamente para variables simples, o para tipos de datos que permiten una notación concreta compacta para las variables (otro ejemplo se muestra en la figura D-3.10.3).

```
PROCESS P1;
   NEWTYPE S Struct
      I Integer;
      B Boolean;
   ENDNEWTYPE;
   DCL
   A S := (. 1,True .);
   ...
ENDPROCESS P1;
```

FIGURA D-3.10.3

Otro ejemplo de inicialización de variable

La figura D-3.10.3 muestra que el género Struct tiene una notación concreta abreviada para indicar un valor de estructura. En el caso de un generador de matrices se sugiere inicializar explícitamente las variables simulando una «construcción while» (construcción «mientras») en la cadena de transición inicial del proceso (véase la figura D-3.10.4).

```
PROCESS P;
        NEWTYPE Arr1 Array (Nat1,Integer)
        ENDNEWTYPE Arr1;
        SYNTYPE Nat1 Natural CONSTANTS 1:3
        ENDSYNTYPE Nat1;
        DCL
          A Arr1,
          I Natural;
        START;
        TASK I:=1;
  Lab1: DECISION I<=3;
           (True): TASK A(I):=I;
                   TASK I:=I+1;
                   JOIN Lab1;
           (False):;
        ENDDECISION;
        ...
ENDPROCESS P;
```

FIGURA D-3.10.4

Un ejemplo más complejo de inicialización de variable

D.3.10.2 *Variables reveladas/vistas (observadas)*

Dos procesos pueden intercambiar información por otros medios que no sean señales. Un proceso puede tener acceso al valor de una variable que pertenece a otro proceso por medio de una operación VIEW. Hay, sin embargo, que hacer notar varias reglas:

- ambos procesos deben pertenecer al mismo bloque;
- el proceso que ejecuta la operación VIEW debe especificar el identificador de la variable vista (observada) en la definición de visión;
- el proceso que revela la variable debe declararla con el atributo REVEALED;
- el identificador de género (o identificador de sintipo) en la declaración de variable y en la definición de visión debe ser el mismo.

El valor que el proceso observador (veedor) obtiene por medio de la operación VIEW es el mismo que el proceso que revela obtiene por medio del acceso ordinario.

Como el proceso observador no posee la variable observada, no puede modificar su valor. Naturalmente, el valor visto puede ser asignado a una variable que pertenece al proceso observador.

El usuario del LED puede encontrar que la definición de variables reveladas/observadas permite un medio fácil de especificar comunicación entre dos procesos. Sin embargo, puede surgir un gran número de problemas al realizar sistemas especificados de esta manera, y esta sección está destinada a guiar a los usuarios para evitar y superar dichos problemas. Es menos difícil describir en LED sistemas que tienen implementados valores revelados/vistos, porque los problemas se habrán superado en la implementación y debería ser posible hacer corresponder la solución escogida con el LED.

En lo que resta de esta sección se supone que el proceso R (revelador) posee y revela variables, y el proceso V (veedor u observador) se refiere a ellas en su definición de visión.

Si se intenta observar variables en el proceso V antes que el proceso R sea creado, el resultado es un error LED. El usuario puede evitar este problema de dos maneras. Ellas son:

– asegurarse que la instancia del proceso revelador R ha sido creada y ha inicializado las variables pertinentes antes que la instancia del proceso observador V, o

– asegurarse que V no entra en transiciones que usen variables reveladas/observadas hasta que R haya sido creada y haya inicializado las variables pertinentes.

Una manera sencilla de llevar a cabo el primer caso es hacer a R el progenitor (o ascendiente) de V (como en el ejemplo de la figura D-3.10.5), o hacer que R sea creado al mismo tiempo que el sistema (creación implícita). En este último caso se puede disponer que la transición pertinente en V pueda ser desencadenada solamente por una señal procedente de R.

Las variables de R no pueden ser observadas después que R haya parado. Cualquier intento de observar datos resultará entonces en un error LED. El usuario puede evitar este problema de dos maneras. Ellas son:

– no usar de ninguna manera una parada en R, o

– asegurarse de que V se haya enterado de que R está por detenerse y no haga ningún intento posterior para observar los datos pertinentes.

La primera solución tiene la desventaja para el implementador que R no ha liberado el almacenamiento de datos que estaba utilizando.

En la figura D-3.10.5 puede encontrarse un ejemplo de variables reveladas/vistas.

D.3.10.3 *Valores exportados/importados*

Un proceso puede declarar una o más de sus variables somo «exportables» lo que tiene el efecto de que todos los otros procesos (cualquiera que sea el bloque a que pertenezcan) pueden importar, a petición, una copia del valor de la variable. El proceso importador debe declarar la variable en su definición de importación.

Cuando el proceso exportador ejecuta una exportación, el valor de la variable se copia en una variable implícita. Un proceso importador obtiene, por medio de una expresión de importación, el valor de esta copia. Así, el valor obtenido por la expresión de importación puede diferir del valor obtenido mediante acceso ordinario por el propietario, aunque ambos se ejecutan al mismo tiempo.

En la figura D-3.10.5 se muestra un ejemplo.

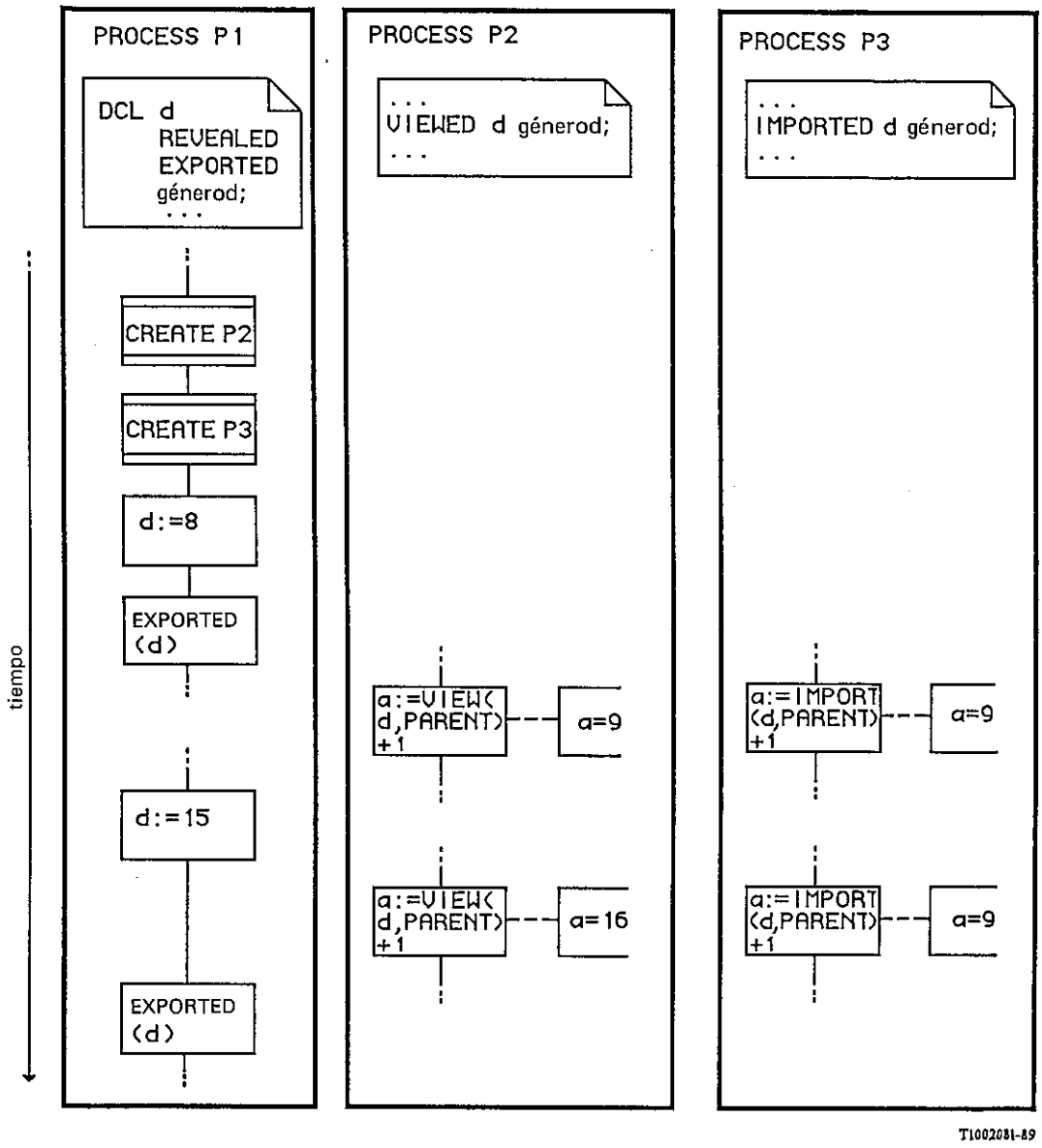


FIGURA D-3.10.5

Ejemplo de diferencias entre el uso de variables reveladas/vistas y variables exportables

En la figura D-3.10.5 el proceso P1 se supone ser el progenitor de los procesos P2 y P3, por lo tanto, el identificador de instancia de proceso en las expresiones IMPORT y VIEW está indicado por el atributo PARENT.

D.3.10.4 *Expresiones*

En un proceso LED las expresiones pueden utilizarse como texto formal en decisiones, alternativas, selecciones, tareas, señales continuas, condiciones habilitantes y construcciones de inicialización. Las expresiones también se utilizan como parámetros efectivos de salida, de llamada de procedimiento y de la construcción crear. Las expresiones PId se utilizan en la parte TO de la construcción de salida. Las expresiones en las construcciones de alternativa y de selección (evaluadas estáticamente) deben ser de géneros de datos predefinidos. En una expresión puede haber términos fundamentales (es decir términos que contienen sólo representaciones de valores constantes) y términos con variables.

El LED tiene un conjunto de operadores infijos predefinidos. Estos operadores pueden ser utilizados para cualquier tipo de datos, y son los únicos operadores infijos permitidos. Para estos operadores las reglas de precedencia son también predefinidas y no pueden cambiarse. Los operadores infijos predefinidos son:

=>, OR, XOR, AND, IN, /=, =, >, <, <=, >=, +, -, //, *, /, MOD, REM

El LED proporciona también los siguientes operadores predefinidos:

–, NOT

que son operadores prefijos unarios.

Cuando se usan los operadores predefinidos, debe tenerse cuidado en lo concerniente a las reglas de precedencia porque estos operadores, siendo predefinidos independientemente del dominio de su aplicación, pueden prestarse a confusión.

Consideremos, por ejemplo, el neotipo y la expresión en la figura D-3.10.6. Pensando en las reglas de precedencia de la multiplicación y de la suma se evalúa la expresión como A => ((B*C) + D). Pero esto es diferente de la precedencia de AND y OR y un cambio como este puede llevar a confusión. Además, este tipo de cambio puede llevar a axiomas inconsistentes e invalidar la especificación LED.

```
NEWTYPE Newbool
   INHERITS Boolean
   OPERATORS ("+" = "AND", "*" = "OR", "=>")
ENDNEWTYPE Newbool;

...

A => B*C+D
```

FIGURA D-3.10.6

Ejemplo de notación engañosa en una expresión

Todos los otros operadores definidos por el usuario son funciones y deben ser utilizados en la notación prefija.

Los dos operadores VIEW e IMPORT tienen una semántica particular explicada en los puntos precedentes. En todo caso, ellos devuelven un valor de cierto género que puede ser otro operando de una expresión.

D.3.11 *Expresión del tiempo en LED*

La necesidad de medir el tiempo y solicitar límites de tiempo en un sistema se satisface con los temporizadores y un conjunto de operaciones realizadas con ellos.

En el modelo LED los «temporizadores» son metaprocesos que pueden enviar señales al proceso a petición. El uso de temporizadores debe ser declarado en la definición de temporizador dentro de las definiciones del proceso. Las operaciones "SET" (INICIALIZACIÓN) y "RESET" (REINICIALIZACIÓN) se utilizan para activar los temporizadores. La operación SET solicita que ocurra un límite de tiempo en un instante especificado, y la operación RESET anula el límite especificado (adviértase que una operación SET incluye implícitamente una operación RESET de cualquier período que no haya expirado de ese temporizador).

La construcción de inicialización contiene la expresión de tiempo del período solicitado, el nombre del temporizador afectado y, facultativamente, una lista de expresiones. La lista de expresiones especifica valores que estarán contenidos en la señal del temporizador en el mismo orden.

La lista de expresiones puede ser especificada en la construcción de reinicialización para reinicializar una instancia particular de la instancia de temporizador que tiene los mismos valores.

La definición de temporizador debe incluir la lista de los identificadores de referencia de género correspondientes para los géneros utilizados en las expresiones en inicializar/reinicializar.

En la figura D-3.11.1 se ilustra un ejemplo de una sentencia de inicialización.

En la construcción de inicialización debe especificarse un tiempo absoluto. El tiempo relativo se transforma en un valor de tiempo absoluto agregando la función primitiva «NOW» (AHORA) que representa el instante actual. La expresión del período (retardo, o demora) solicitado deberá ser una expresión tiempo (time). Tiempo es un género predefinido, heredado del género real.

La posibilidad de recibir un límite de tiempo se especifica por medio del nombre del temporizador en una entrada, como se muestra en la figura D-3.11.1.

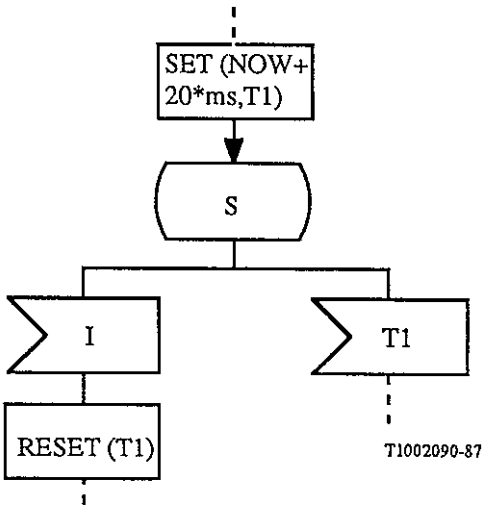


FIGURA D-3.11.1

Ejemplo del uso de temporizadores

Es posible definir sinónimos para indicar las duraciones deseadas. Una vez elegidas las unidades de tiempo y de duración, el usuario puede definir los sinónimos necesarios para representar las duraciones como se muestra en la figura D-3.11.2.

```
PROCESS p;
   TIMER T1 subscriber_id;
   ...
   DCL Sa subscriber_id;
   ...
   SYNONYM
      Sec Duration = 1000.0,
      Min Duration = 60000.0,
   ...
   START;
      ...
      SET (NOW +20*Min +30*Sec , T1 (Sa) );
      ...
      RESET (T1 (Sa));
      ...
ENDPROCESS p;
```

FIGURA D-3.11.2

Sinónimos para representar duraciones

En la Recomendación se establece que se permite la inicialización de un temporizador a un tiempo que ya ha «pasado». Esta decisión se tomó para facilitar la simulación de sistemas. Sin embargo, como esa inicialización podría dar lugar a unas especificaciones poco claras, este uso debe evitarse.

D.3.12 *Uso de calificadores*

En LED los calificadores se utilizan para hacer referencia a ítems en una especificación, cuando el nombre no determina el ítem en forma única. Naturalmente, cuando se define el ítem, solamente el nombre debe ser especificado, pero cuando se hace referencia al mismo fuera de su ocurrencia de definición puede necesitarse un identificador compuesto por un calificador y el nombre.

Esto también se aplica cuando se utilizan definiciones distantes: la ocurrencia de definición utiliza el nombre cuando hace referencia a la definición; la definición distante utiliza un nombre calificado para especificar su contexto.

En la figura D-3.12.1 se presenta un ejemplo del uso de calificadores para un proceso que puede recibir dos señales diferentes que tienen el mismo nombre ¡pero diferentes identificadores! La primera entrada se refiere a un tipo de señal definido a nivel de bloque; la segunda entrada se refiere a un tipo de señal definido en la definición de proceso.

En la segunda entrada, la calificación puede omitirse porque, cuando los identificadores no están calificados, la definición más interna es la referida.

```
SYSTEM s;
   ...
   BLOCK b;
      SIGNAL x;
      ...
      PROCESS p;
         ...
         SIGNAL x;
         ...
         STATE esperar;
            INPUT SYSTEM s/BLOCK b x;
               ...
            INPUT PROCESS p x;
               ...
         ENDSTATE esperar;
         ...
      ENDPROCESS p;
   ENDBLOCK b;
ENDSYSTEM s;
```

FIGURA D-3.12.1

Ejemplo del uso de calificadores

D.3.13 *Sintaxis de nombres*

Los nombres en LED pueden consistir en una sola palabra o en una lista de palabras separadas por delimitadores (espacios o caracteres de control). La segunda posibilidad permite una especificación más legible cuando se usan nombres largos, especialmente cuando se usa el LED/GR ya que los símbolos gráficos tienen un tamaño limitado. En la figura D-3.13.1 se muestran algunos ejemplos de nombres que constan de varias palabras.

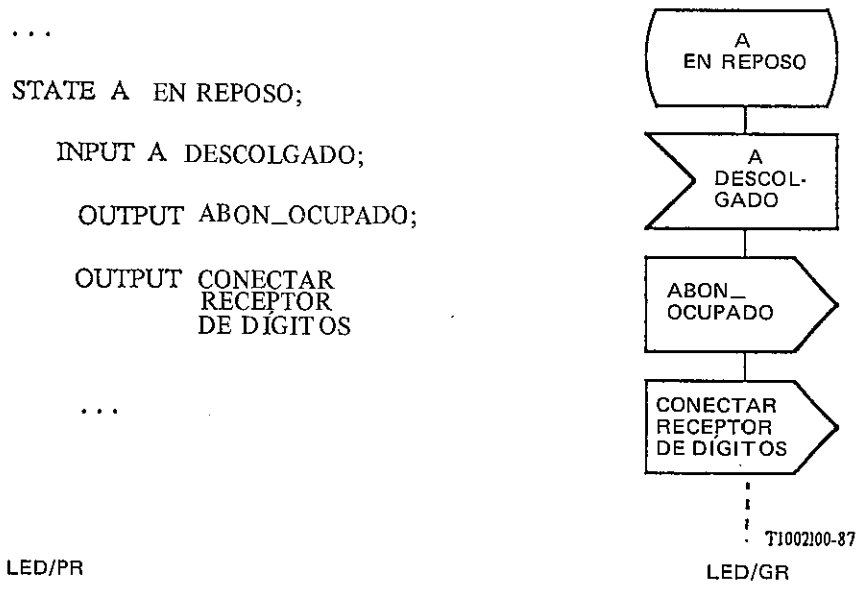


FIGURA D-3.13.1

Ejemplos de nombres que constan de varias palabras

Cuando el uso de varias palabras en un nombre es ambiguo (véanse ejemplos en la figura D-3.13.2), los delimitadores entre dos palabras deben sustituirse por un carácter de subrayado («_»). La cadena de caracteres obtenida transformando los delimitadores en caracteres de subrayado sigue denotando el mismo nombre; así, por ejemplo, ABON OCUPADO denota el mismo nombre que ABON_OCUPADO. En la figura D-3.13.3 encontramos algunos ejemplos de uso no ambiguo de nombres.

```
a)  A := BLOCK B1 B / PROCESS P1 P C;
b)  DCL A B C D;
c)  NEWTYPE X Y Z (PAR) ENDNEWTYPE;
```

FIGURA D-3.13.2

Ejemplos de utilización ambigua de nombres

```
a)  A := BLOCK B1_B/PROCESS P1_P C;
b)  A := (BLOCK B1 B)/(PROCESS P1_P C);
c)  DCL A B_C_D;
d)  DCL A_B C_D;
e)  DCL A_B_C D;
f)  NEWTYPE X Y_Z(PAR) ENDNEWTYPE;
g)  NEWTYPE X_Y Z(PAR) ENDNEWTYPE;
```

FIGURA D-3.13.3

Ejemplos de utilización no ambigua de nombres

Es importante observar que el carácter de subrayado puede también utilizarse en nombres como un carácter de continuación, para permitir la división de nombres en más de una línea. En este caso, un nombre que contiene un carácter de subrayado seguido por uno o más delimitadores puede también indicarse eliminando tanto el carácter de subrayado como los delimitadores.

En la figura D-3.13.4 se dan ejemplos de designaciones diferentes para el mismo nombre.

a) CONECTAR RECEPTOR DE DÍGITOS

b) CONECTAR_RECEPTOR_DE_DÍGITOS

c) CONECTAR RECEPTOR_DE_DÍGITOS

d) CONECTAR
RECEPTOR
DE DÍGITOS

e) CONECTAR RECEPTOR DE DÍ_
GITOS

f) CONECTAR RECEPTOR DE DÍ_ GITOS

g) CONECTAR_
_RECEPTOR_DE_
_DÍGITOS

FIGURA D-3.13.4

Ejemplos de designaciones diferentes para el mismo nombre

D.4 *Estructuración y refinamiento de los sistemas LED*

D.4.1 *Generalidades*

En este capítulo se abordarán algunas técnicas y construcciones del LED que permiten una especificación descendente de grandes sistemas. Generalmente los términos «partición» y «refinamiento» se utilizan para estas técnicas con los siguientes significados:

- partición: subdivisión (fraccionamiento) de una parte de un sistema en partes más pequeñas cuyo comportamiento global es equivalente a la parte no fraccionada. Puede aplicarse a bloques (estructurándolos en nuevos subbloques, canales y subcanales), a canales (estructurándolos en bloques, nuevos canales y subcanales) y a procesos (estructurándolos en servicios);
- refinamiento: adición de nuevos detalles a las funcionalidades del sistema. Visto desde el entorno, el refinamiento de un sistema causa un enriquecimiento de su comportamiento ya que pueden tratarse más clases de señales e información.

Obsérvese que la estructura interna de una parte del sistema da más detalles de la estructura, pero no necesariamente más detalles del comportamiento del sistema. Conceptualmente, es posible distinguir entre el aspecto de una representación más detallada del comportamiento (por ejemplo, el tratamiento de una nueva señal) y el aspecto de una estructura más detallada, pero en la práctica estos dos aspectos suelen combinarse, de manera que al introducir nuevos detalles de la estructura del sistema facilitamos también nuevos detalles sobre el comportamiento de éste.

La estructura mínima de un sistema en LED es la que se describe en el capítulo 2 de la Recomendación, es decir, un sistema consiste en un conjunto de bloques que contienen procesos y están conectados por canales.

Los conceptos para la partición en varios niveles de detalle y el refinamiento de señales se tratan en el capítulo 3 de la Recomendación. Para los sistemas que no necesitan una partición ulterior esos conceptos no son necesarios.

D.4.2 *Criterios de partición*

La técnica que consiste en comenzar con una visión de alto nivel de la representación de un sistema y descomponerla en partes manejables se llama partición. Este proceso de partición añade estructura a un sistema.

Los criterios conducentes a la partición de la representación del sistema son diversos e incluyen los siguientes:

a) definir bloques o procesos de tamaño manejable desde el punto de vista intelectual;

b) crear una correspondencia con las divisiones reales del soporte lógico (software) y/o del equipo físico (hardware);

c) seguir las subdivisiones funcionales naturales;

d) minimizar la interacción entre los bloques;

e) reutilizar las especificaciones ya existentes (por ejemplo, un sistema de señalización);

Los criterios que se adopten de hecho dependerán de cierto número de factores, incluido el grado de detalle requerido.

Como la relación entre los niveles dependerá de los criterios de partición adoptados, es importante indicar éstos claramente para facilitar la comprensión de la representación. Los criterios de partición dependen del usuario pero existen algunas limitaciones a fin de garantizar una representación correcta en LED. Éstas se examinan en los puntos siguientes.

D.4.3 *Partición de un bloque*

Un bloque puede ser subdividido en un conjunto de bloques y canales, casi de la misma manera en que se subdivide un sistema en bloques y canales. En LED/GR esto se representa por medio de un diagrama de subestructura de bloques. En la figura D-4.3.1 se muestra un ejemplo de un diagrama de subestructura de bloques. En el primer diagrama de la figura está el diagrama de bloques del bloque B1 que contiene una referencia a su subestructura. En el segundo diagrama está el diagrama de la subestructura para el bloque B1. El símbolo de bloque unido al canal C2 por una línea de trazo discontinuo en el primer diagrama representa una referencia a la subestructura del canal C2 (véase el § D.4.5).

En LED/PR la partición de un bloque se representa mediante un conjunto de definiciones comprendidas entre las palabras clave SUBSTRUCTURE y ENDSUBSTRUCTURE, dentro de la definición del bloque. En una subestructura de bloque las definiciones son las mismas que existen en una definición de sistema. Además, debe proporcionarse la especificación de las conexiones entre canales y subcanales como se muestra en la figura D-4.3.2.

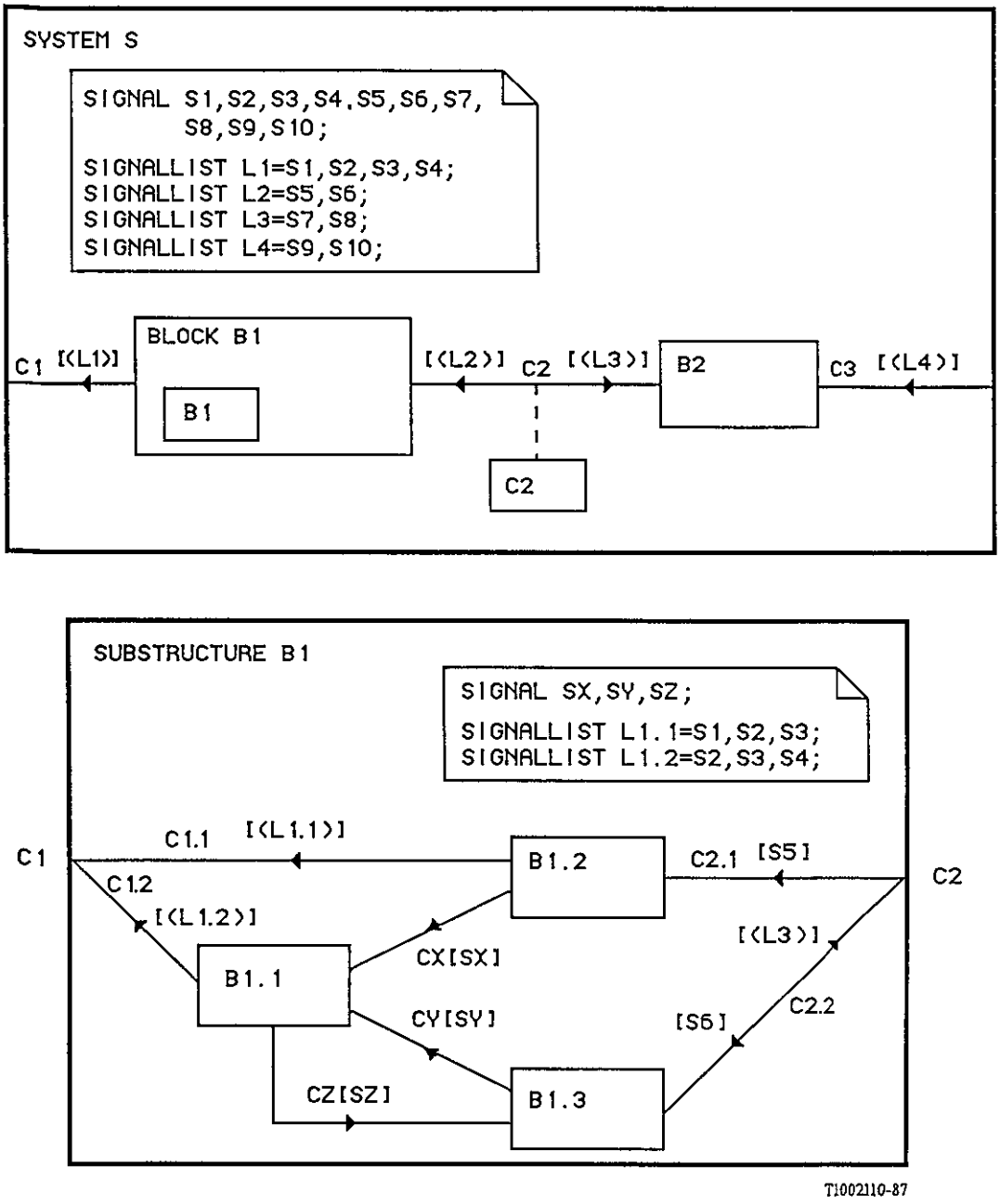


FIGURA D-4.3.1

Partición de bloques

```
SUBSTRUCTURE B1;

   SIGNALLIST L1.1 =S1,S2,S3;
   SIGNALLIST L1.2 =S2,S3,S4;

   SIGNAL SX,SY,SZ;

   CHANNEL C1.1
      FROM B1.2 TO ENV WITH (L1.1);
   ENDCHANNEL C1.1;
   CHANNEL C1.2
      FROM B1.1 TO ENV WITH (L1.2);
   ENDCHANNEL C1.2;
   CHANNEL C2.1
      FROM ENV TO B1.2 WITH S5;
   ENDCHANNEL C2.1;
   CHANNEL C2.2
      FROM ENV TO B1.3 WITH S6;
      FROM B1.3 TO ENV WITH (L3);
   ENDCHANNEL C2.2;
   CHANNEL CX
      FROM B1.2 TO B1.1 WITH SX;
   ENDCHANNEL CX;
   CHANNEL CY
      FROM B1.3 TO B1.1 WITH SY;
   ENDCHANNEL CY;
   CHANNEL CZ
      FROM B1.1 TO B1.3 WITH SZ;
   ENDCHANNEL CZ;

   CONNECT C1 AND C1.1,C1.2;
   CONNECT C2 AND C2.1,C2.2;

   BLOCK B1.1 REFERENCED;
   BLOCK B1.2 REFERENCED;
   BLOCK B1.3 REFERENCED;

ENDSUBSTRUCTURE B1;
```

FIGURA D-4.3.2

LED/PR para el ejemplo de la figura D-4.3.1

Dentro de una definición de bloque, la especificación de los procesos y de una subestructura de bloque pueden coexistir. En este caso los procesos del bloque representan el comportamiento del bloque a cierto nivel de detalle, otros procesos dentro de las definiciones de subbloques representarán el mismo comportamiento de una manera más detallada. En el § D.4.7 figura un ejemplo de bloques descritos tanto en términos de procesos como de subestructuras (figura D-4.7.1).

Para simplificar el diagrama se proporciona en LED/GR una notación abreviada si en un bloque no hay procesos sino solamente la subestructura del bloque, y si la subestructura del bloque no está referenciada. Esa notación permite el anidamiento de bloques considerando que la casilla del bloque exterior implica la casilla de la subestructura. Por medio de esta notación, el ejemplo de la figura D-4.3.1 puede dibujarse como está en la figura D-4.3.3.

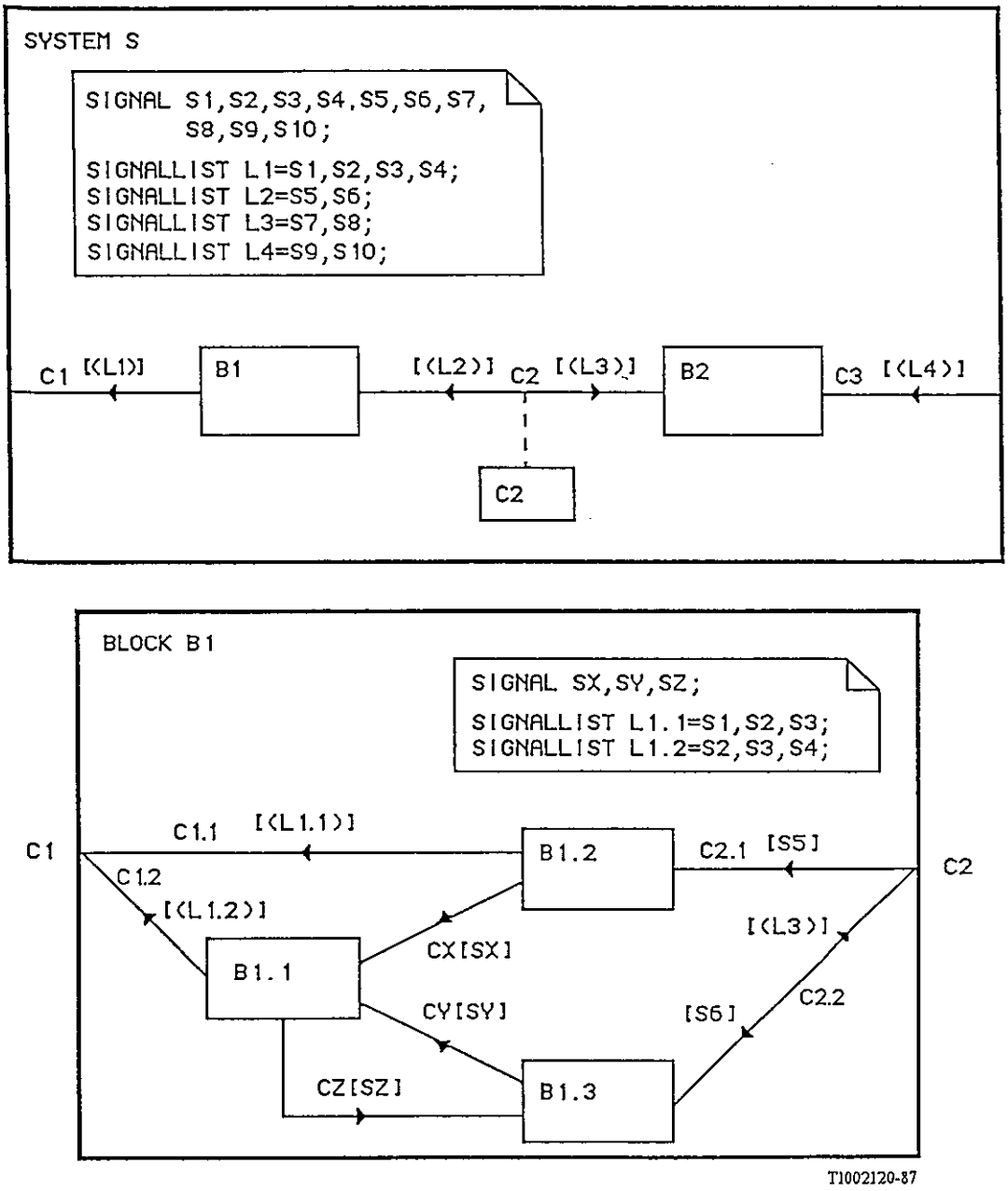


FIGURA D-4.3.3

Notación LED/GR abreviada para partición de bloques

Cada bloque que se deriva de la partición de un bloque puede subdividirse a su vez obteniéndose así una estructura jerárquica de árbol de bloques y subbloques. Un diagrama auxiliar llamado «diagrama árbol de bloques», que muestra una estructura general como esa, se explica en el § D-4.4.

A menos que haya un refinamiento de señal, la partición debe cumplir las reglas siguientes:

1) Los subcanales conectados a un canal entrante no deben contener ninguna señal nueva en sus listas de señales, y sus listas de señales deben contener todas las señales de la lista del canal original (para canales bidireccionales debe considerarse la lista del trayecto entrante). Así, para el ejemplo que se ilustra en la figura D-4.3.1, C2.1 y C2.2 transportan en los trayectos entrantes todas las señales en L2. Además, ninguna señal transportada por C2.1 puede aparecer en el trayecto entrante de C2.2.

2) Los subcanales (por ejemplo, C1.1 y C1.2) conectados a un canal saliente (por ejemplo, C1) no deben contener el nombre de ninguna señal nueva en sus listas de señales, y sus listas de señales deben contener todos los nombres de señal del canal original. Así L1.1 y L1.2 contienen todos los nombres de señal de L1. Las listas L1.1 y L.1.2 pueden contener los mismos identificadores de señal.

3) Si el bloque original contiene procesos, se dispone de dos opciones. Primero, una copia de cada proceso puede ser redefinida en uno u otro de los nuevos subbloques. Segundo, pueden definirse nuevos procesos en los subbloques de manera que el interfaz permanezca sin cambios.

4) Las definiciones de datos en el bloque progenitor están disponibles para sus subbloques, de modo que cada uno de ellos puede usar un tipo de datos definido en el bloque progenitor sin tener que redefinirlo.

5) Si un tipo de datos definido en el bloque progenitor es redefinido con el mismo nombre en un subbloque, la nueva definición se aplica al subbloque de definición mientras que la anterior es válida para los otros subbloques. Se desaconseja hacer una redefinición con el único objeto de caracterizar un subbloque, porque puede ser pasada por alto por un lector que suponga que la definición anterior es válida. Debe cuidarse de hacer resaltar esta redefinición por medio de anotaciones apropiadas.

D.4.4 *Diagrama de árbol de bloques*

Un diagrama de árbol de bloques representa la estructura de un sistema en términos de bloques y subbloques. La finalidad del diagrama es ofrecer al lector una imagen general de la estructura total del sistema.

El diagrama es un árbol jerárquico de símbolos de bloque con líneas de «partición», como se ilustra en la figura D-4.4.1.

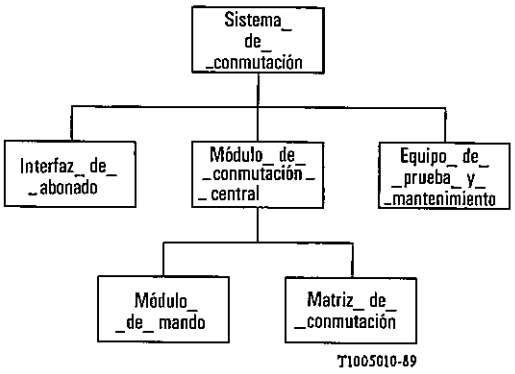


FIGURA D-4.4.1

Ejemplo de un diagrama de árbol de bloques

El diagrama debe dibujarse utilizando de preferencia símbolos de bloque del mismo tamaño. De esta manera, los bloques situados en un mismo nivel de partición aparecen como un nivel uniforme. Si el diagrama es tan grande que hace falta más de una página, debe subdividirse en diagramas de árbol de bloques «parciales» como se indica en la figura D-4.4.2.

A menudo resulta conveniente subdividir un diagrama de árbol de bloques en diagramas parciales.

Para ello, el primer diagrama, que tiene el sistema como raíz, se corta de manera que aparezca un conjunto de bloques separado como si no estuviese subdividido. Los bloques donde se cortó el diagrama original aparecen como raíces en los diagramas que muestran la subdivisión ulterior.

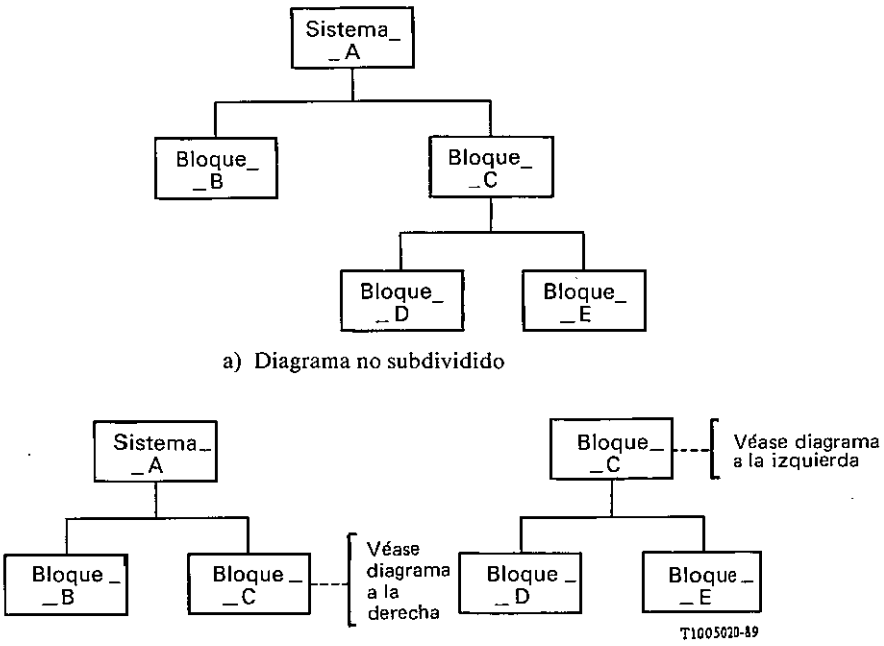


FIGURA D-4.4.2

Ejemplo de diagramas parciales de árbol de bloques

Si se utilizan diagramas parciales y no está claro que un bloque está subdividido aún más y/o dónde aparecen los diagramas de continuación, deben insertarse referencias por medio del símbolo de comentario.

D.4.5 *Partición de un canal*

Un canal puede subdividirse independientemente de los bloques que conecta. Esto permite representar el comportamiento del canal. Para obtener una representación exacta de la manera en que se transmite una señal, a veces puede ser necesario representar el comportamiento del canal. Esto se hace considerando el canal como un elemento en sí mismo, cuyo entorno son los dos bloques que conecta. Al observar el canal de esta manera, podemos mostrar su estructura en términos de bloques, canales y procesos.

En LED/GR la partición de canal se representa por medio de un diagrama de subestructura de canal como se muestra en la figura D-4.5.1. (El ejemplo representa la subestructura del canal C2 de la figura D-4.3.1.)

Un diagrama de subestructura de canal describe cómo un canal se subdivide en subcomponentes. El diagrama se asemeja al diagrama de sistema (con excepción de las conexiones a los bloques). Todas las directrices dadas en el § D.4.3 son válidas también para el diagrama de subestructura de canal.

En el diagrama de interacción de bloques donde aparece el canal subdividido deberá haber una referencia al diagrama de subestructura de canal que describe la partición.

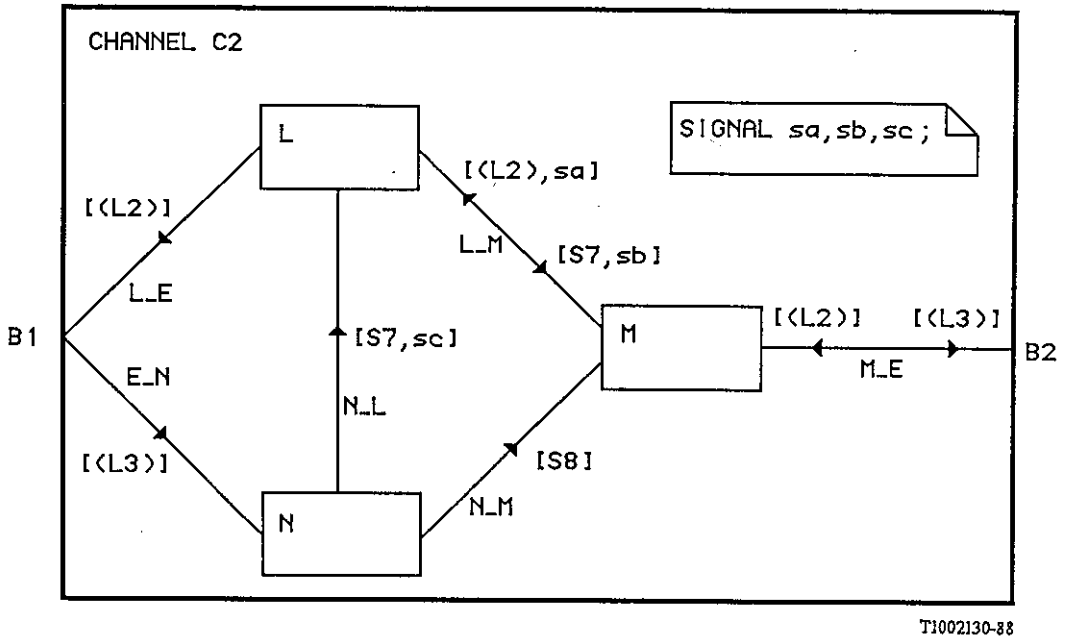


FIGURA D-4.5.1

Ejemplo de diagrama de subestructura de canal

En LED/PR la forma es similar a la forma de la definición de subestructura de bloque, siendo la única diferencia que en la sentencia CONNECT los subcanales de puntos extremos están conectados a los bloques exteriores (B1 y B2 en la figura D-4.5.2) y no a los canales exteriores.

Se permite la importación/exportación de valores entre un bloque y una subestructura de canal. Esto permite una representación sencilla del modelo ISA, en el que la comunicación por capas entre entidades pares se ha modelado mediante intercambio de señales y la comunicación entre capas contiguas mediante valores compartidos.

D.4.6 *Representación del sistema en caso de partición*

En los casos en que un sistema está representado por un conjunto de bloques interconectados por canales y el comportamiento de cada bloque está expresado por uno o varios procesos, tenemos una representación mononivel. Esto significa que podemos ver todos los elementos de la representación en el mismo nivel. Cuando introducimos la partición, insertamos una relación jerárquica entre los diversos documentos. Tendremos un documento con la representación de la estructura del sistema, estando éste compuesto por *n* bloques.

Un documento diferente puede presentar el sistema como compuesto por un conjunto diferente de bloques, algunos de los cuales se derivan de los bloques del documento anterior (algunos bloques del documento anterior han sido sustituidos por los subbloques obtenidos por la partición de los bloques). Este último documento debe estar relacionado con el anterior.

Para obtener una representación completa del sistema no basta simplemente con relacionar los documentos entre sí, sino que los mismos deben estar organizados de manera que sea posible tener acceso a la representación del sistema por niveles, comenzando con una imagen general y pasando a representaciones cada vez más detalladas. Esto hace necesario agrupar los diversos documentos a fin de que formen diversos niveles de representación del sistema.

No todos los niveles deben contener los mismos elementos. En un primer nivel, la representación del sistema puede consistir en representaciones de bloques y canales sin incluir los procesos que describen el comportamiento de cada bloque. En un nivel más bajo podría incluirse la representación del comportamiento de algunos bloques pero no el de otros. El nivel de representación más bajo de todos (el más detallado) debe comprender la representación completa de los comportamientos de todos los bloques, o sea el conjunto completo de procesos que expresan este comportamiento.

```
SUBSTRUCTURE C2;

   SIGNAL sa,sb,sc;

   CHANNEL L_E
      FROM L TO ENV WITH (L2);
   ENDCHANNEL L_E;

   CHANNEL E_N
      FROM ENV TO N WITH (L3);
   ENDCHANNEL E_N;

   CHANNEL M_E
      FROM M TO ENV WITH (L3);
      FROM ENV TO M WITH (L2);
   ENDCHANNEL M_E;

   CHANNEL L_M
      FROM L TO M WITH S7,sb;
      FROM M TO L WITH (L2),sa;
   ENDCHANNEL L_M;

   CHANNEL N_M
      FROM N TO M WITH S8;
   ENDCHANNEL N_M;

   CHANNEL N_L
      FROM N TO L WITH S7,sc;
   ENDCHANNEL N_L;

   CONNECT B1 AND L_E,E_N;
   CONNECT B2 AND M_E;

   BLOCK L REFERENCED;
   BLOCK M REFERENCED;
   BLOCK N REFERENCED;

ENDSUBSTRUCTURE C2;
```

FIGURA D-4.5.2

LED/PR para el ejemplo de la figura D-4.5.1

Observando el árbol de bloques podemos formular varias consideraciones.

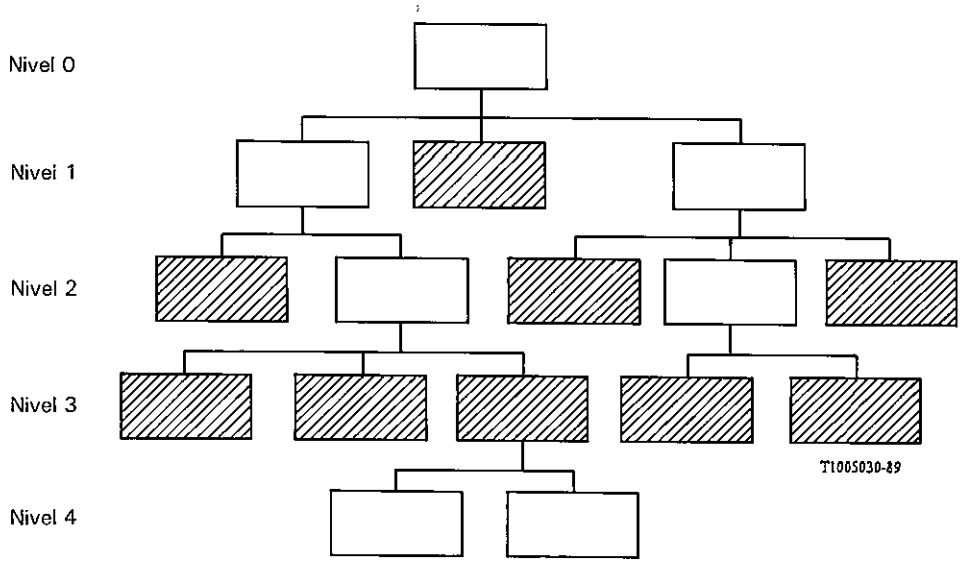
Primeramente, el árbol tiene siempre una sola y única raíz, a saber, el sistema. Tal ocurre incluso en los casos en que un sistema está representado desde el comienzo como compuesto de varios bloques. En el árbol de bloques estos bloques estarán representados, por ejemplo, en el nivel 1. La raíz puede consistir en el nombre del sistema únicamente. Las definiciones de los canales pueden indicarse para los bloques en el nivel 1, aunque esto no es obligatorio a menos que los bloques incluyan procesos.

Surge un segundo punto observando las hojas del árbol: no todas tienen que estar en el mismo nivel. Esto puede ser el resultado de un número diferente de particiones de los bloques del árbol. El número de particiones depende de varios aspectos, la mayoría de los cuales dependen de la apreciación subjetiva del especificador/diseñador.

En LED un bloque hoja (en el árbol de bloques) no necesita más partición solamente si su comportamiento está completamente especificado (es decir, si las definiciones de los procesos asociados son suficientes para representar su comportamiento). En consecuencia, un bloque hoja debe contener por lo menos un proceso asociado.

Cuando la representación de un sistema se expresa en varios niveles de abstracción, podemos seleccionar cualquiera de esos niveles para representar el sistema. La elección de un nivel determinado implica la consideración de los bloques de ese nivel, de sus procesos asociados y de todos los bloques que son bloques hoja en niveles más altos, junto con sus procesos asociados (figura D-4.6.1).

A menudo conviene elegir niveles diferentes para propósitos diferentes, por ejemplo, un nivel global para la presentación y un nivel más detallado para la realización.



Nota – El nivel 3 de representación se muestra mediante casillas sombreadas.

FIGURA D-4.6.1

Nivel 3 de representación

La representación de un sistema en un nivel dado puede ser incompleta en el sentido de que algunos de los bloques de ese nivel no tienen procesos asociados.

Los motivos para que se hable de niveles de representación y de la selección de un nivel de representación dado son los siguientes:

- puede que hayamos llegado en nuestro diseño a cierto «nivel» de detalle, que esté representado por ese nivel de representación (¡en este caso los bloques de ese nivel son bloques hoja y no están completos pues hace falta seguir trabajando!);
- deseamos observar la representación del sistema con cierto nivel de detalle; en consecuencia, elegimos el nivel de representación que mejor corresponda a la abstracción que buscamos. Adviértase que en algunos casos cierto nivel de representación puede consistir en documentos con diferentes niveles de abstracción. Una parte del sistema puede representarse en forma muy detallada en el nivel dos, mientras que otra parte puede ser aún abstracta en el nivel cuatro. Quiere decir que cuando elegimos una representación en el nivel tres podemos tener partes muy detalladas junto con partes que sólo pueden considerarse como una visión general;
- la metodología de representación/diseño puede ser tal que cada nivel tenga un significado preciso. Por ejemplo, el nivel uno puede corresponder a la especificación, el nivel dos presentar la estructura global del sistema, el nivel tres la estructura modular (bastidores, funciones de soporte lógico), el nivel cuatro la estructura detallada (tarjetas de circuito impreso, procedimientos, módulos de soporte lógico). En este caso la elección de un dado nivel corresponde a las necesidades de un determinado lector, y cabe señalar que la metodología evitará una situación de discrepancias en el nivel de detalles de las partes que componen un nivel de representación determinado.

D.4.6.1 *Subconjunto de partición consistente*

Además de la especificación del sistema total y de la expresada por niveles, el LED posee el concepto de «subconjunto de partición consistente». Puede considerarse como una especificación en un solo nivel donde todos los bloques pueden tomarse de cualquier nivel de la estructura del sistema dado que:

– un bloque puede elegirse como parte de la representación consistente del sistema si el mismo puede considerarse como un bloque hoja (o bien es un bloque hoja, o bien tiene asociados todos los procesos necesarios para representar su comportamiento);

– si se elige un bloque todos los bloques obtenidos con la partición de su bloque progenitor deben incluirse, sea directamente o por inclusión de sus vástagos;

– deben facilitarse todos los documentos que definen las señales que circulan por el canal que conecta un bloque de la representación. Esto puede implicar, dependiendo de la estrategia elegida para la partición, que una vez que se ha tomado un bloque, su progenitor deba tomarse también, por lo menos para las partes que definen los datos y las señales.

En los casos en que algunos canales se hayan subdividido considerándose como sistemas, debe facilitarse la representación de cada uno de esos «sistemas». Su especificación consiste en el mismo tipo de documentos que la representación usual del sistema. Deberá agregarse notación que relacione estos sistemas a la especificación del sistema principal a que pertenecen. Así, en un sentido, podemos considerar estos canales divididos como sistemas interiores. Cada uno de estos sistemas puede tener varios niveles de representación y tener también sistemas internos si algunos de los canales contenidos en ellos se subdividen aún más.

D.4.7 *Refinamiento*

El mecanismo de refinamiento ha sido introducido en LED para «ocultar» señales de bajo nivel de niveles más altos de abstracción y permitir una especificación descendente del comportamiento del sistema.

El refinamiento permite al usuario subdividir señales en subseñales resultando esto en una estructura jerárquica como en el caso de bloques y subbloques. Esto es, dentro de una definición de señal es posible definir un conjunto de nuevas señales que se dice son señales refinadas o subseñales de la señal que está siendo definida. Puede dibujarse para una definición de señal, con fines explicativos, un árbol de señales, del mismo modo que un árbol de bloques para una definición de bloques.

El refinamiento está estrechamente relacionado con la partición de bloques porque solamente aquellas señales transportadas por un canal conectado a un bloque subdividido pueden ser refinadas. Esto es, una señal incluida en la lista de un canal puede reemplazarse por sus subseñales cuando el bloque conectado está subdividido. Los subcanales correspondientes generados en la partición del bloque especificarán las subseñales en sus listas de señales.

El canal especificado para llevar una señal será automáticamente el portador de todas las subseñales de la señal, incluso si algunas de las subseñales circulan en sentido opuesto (en ese caso el canal se considera implícitamente bidireccional). En la figura D-4.7.1 se presenta un ejemplo de refinamiento. Este ejemplo representa un sistema en el que un proceso un bloque envía ficheros de texto a un proceso de otro bloque. El nivel de refinamiento más alto se alcanza al enviar señales que representan cada una un fichero de texto (señal sf). En el nivel de refinamiento siguiente se desea especificar que ese fichero de texto consiste en cierto número de registros que se envían uno por uno (señal sr) y que el receptor tiene que responder (señal nr) después de consumir cada registro. El emisor enviará al final una señal de fin-de-fichero. En este ejemplo, al nivel más alto de refinamiento, de los procesos B1 y B2 comunican usando la señal sr; en el nivel inmediatamente inferior, los procesos B11 y B21 comunican usando las señales sr, nr y eof.

Algunos casos reales en que puede aplicarse el concepto de refinamiento son los siguientes:

– transformación de nombre: una señal se refina en otra señal con un nombre diferente. Esta es una transformación biunívoca en que sólo el nombre de la señal cambia. Esta posibilidad se aplica generalmente cuando queremos tener cada nivel completamente comprensible por sí mismo (puede ser conveniente adaptar el nombre de una señal a su contexto);

– transformación por división: es una transformación de uno o varios en que una señal queda dividida en varias señales como resultado de una caracterización. Por ejemplo, una señal genérica «alarma» se divide en «alarma_de_registro», «alarma_de_procesador_central» y «alarma_de_abonado».

– transformación algoritmo: la señal original se transforma en un conjunto de señales que activan un algoritmo para suministrar la información original.

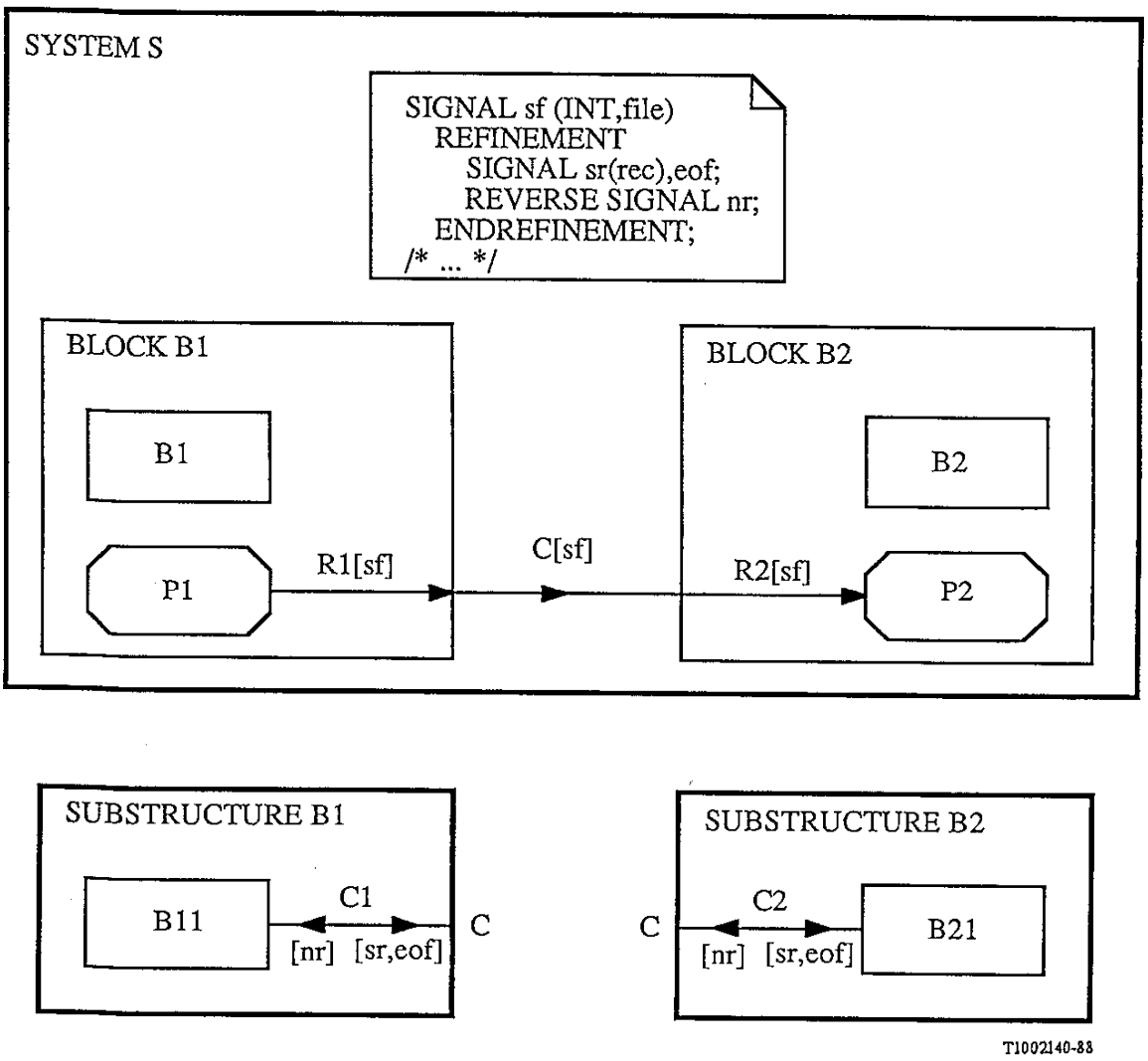


FIGURA D-4.7.1

Ejemplo de refinamiento

D.4.7.1 *Subconjunto de refinamiento consistente*

Cuando se aplica refinamiento en una especificación de sistema, el concepto de subconjunto de partición consistente se limita a evitar comunicaciones con señales diferentes entre niveles de refinamiento diferentes. En este caso, se dice que la definición de sistema contiene varios subconjuntos de refinamiento consistentes. Si un subconjunto de refinamiento consistente contiene un proceso que comunica con subseñales, entonces el proceso no puede comunicar con la señal ascendiente y el otro extremo del enlace de comunicación debe comunicar con las mismas subseñales.

D.4.7.2 *Transformación entre señales y subseñales*

El usuario puede necesitar describir la transformación entre diferentes niveles de refinamiento a efectos de simulación o para verificar el comportamiento del sistema a diferentes niveles de detalle. Esto puede hacerse de manera informal por medio de procesos LED adicionales que describen la transformación dinámica de una señal en subseñales o viceversa.

En la figura D-4.7.2, se han introducido dos procesos para describir la transformación aplicada en la figura D-4.7.1. Un proceso de refinación define cómo la señal de alto nivel se refina en un conjunto de señales en el nivel inmediatamente inferior. Un proceso de recuperación describe la transformación inversa.

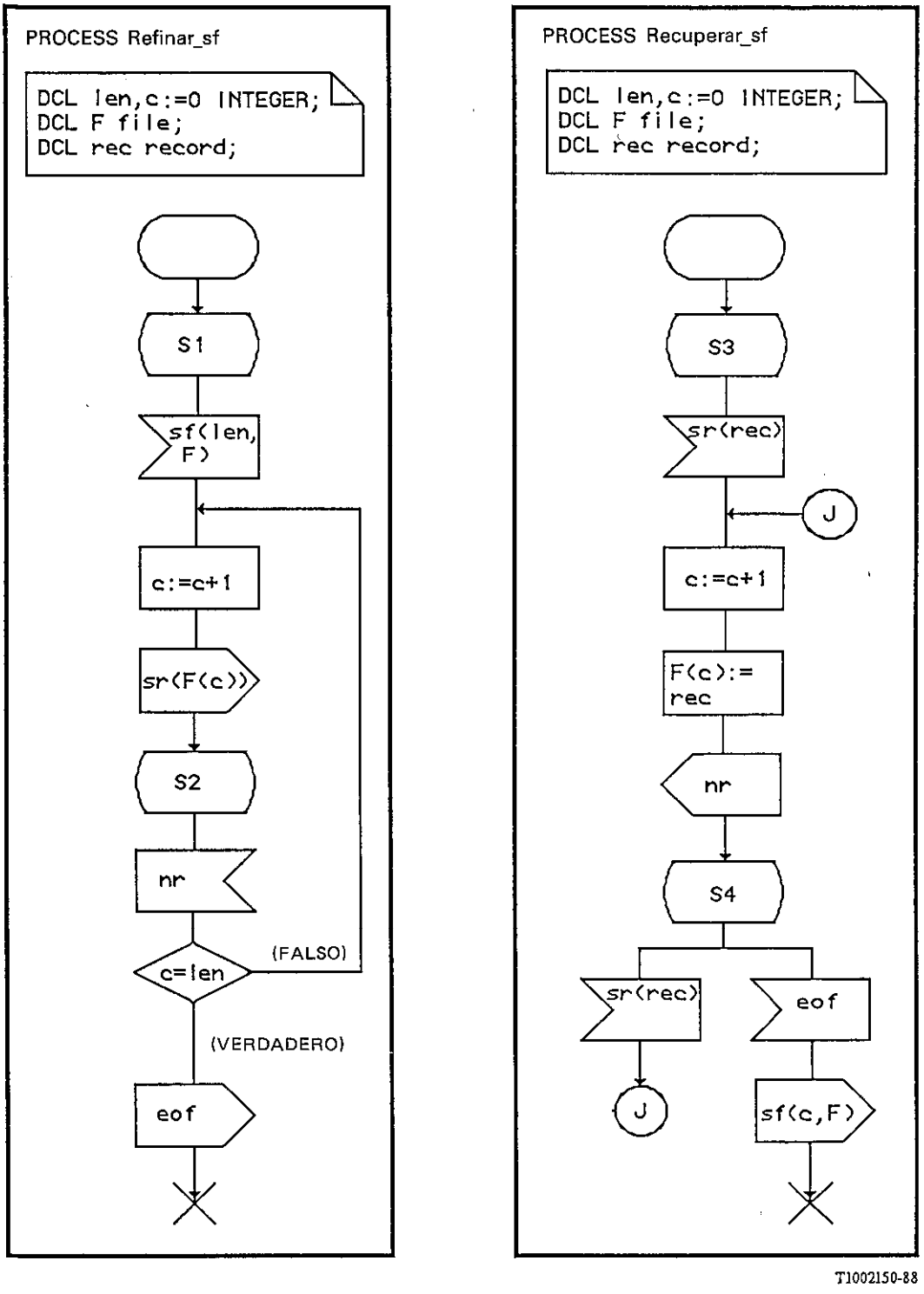


FIGURA D-4.7.2

Ejemplo de especificación de una transformación de señal

D.5 *Conceptos adicionales*

D.5.1 *Macros*

Una macroinstrucción (o macro) es un medio que permite tratar repeticiones y/o estructurar una descripción. Consiste en una definición de macro que contiene una parte de la especificación LED que puede ser referenciada (llamada de macro) en otros sitios de la especificación del sistema.

La definición de macro puede darse en todos los lugares en que se permiten definiciones de datos. Sin embargo el nombre de macro no tiene ámbito. Así, una macro definida dentro de un bloque puede ser referenciada fuera de ese bloque.

En LED/PR es posible tener una definición de macro que reemplace cualquier secuencia de unidades léxicas. Esto difiere de las definiciones de macro LED/GR, que pueden reemplazar sólo colecciones de unidades sintácticas.

Para hacer corresponder documentos LED que contengan macros entre LED/GR y LED/PR, deberán aplicarse las siguientes restricciones sobre el uso de las macros LED/PR.

1) Una macro puede reemplazar solamente a una o más de las siguientes contrucciones sintácticas: (Esto corresponde a símbolos LED/GR.)

   arranque
   estado
   entrada
   condición habilitadora
   señal continua
   conservación
   sentencia de acción
   sentencia de terminación

2) Los parámetros formales de macro no pueden utilizarse en lugares que determinan el tipo de construcción LED/PR. Esto es, no deben utilizarse donde se utilicen las palabras clave del LED/PR. En forma similar los parámetros efectivos no deben contener palabras clave que correspondan a los símbolos LED/GR: START, STATE, PROCEDURE, INPUT, TASK, OUTPUT, DECISION, CREATE, STOP, PROVIDED, CALL, COMMENT, JOIN, RETURN, SAVE u OPTION.

3) Cada sentencia en la definición de macro debe ser accesible desde al menos un acceso de entrada de macro.

En LED/PR, la macro tiene siempre al menos un acceso de entrada y un acceso de salida, de modo que es necesario usar etiquetas y uniones para representar en LED/PR una macro LED/GR con más de un acceso de entrada o de salida, respectivamente. Si la macro ha de invocarse en más de un sitio, las etiquetas tendrán que ser pasadas como parámetros.

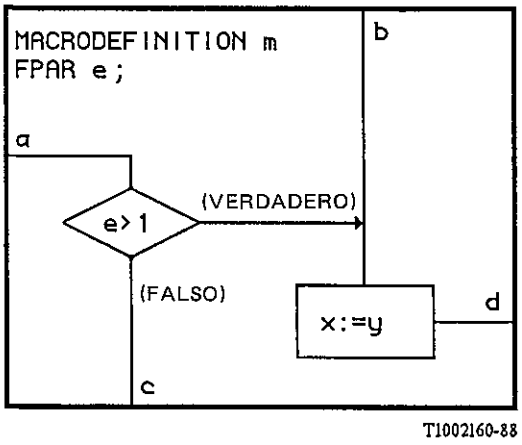
En LED/GR hay dos maneras diferentes de representar accesos de entrada y accesos de salida de una definición de macro. O bien el usuario puede dibujar una casilla para la macro y conectar los accesos de entrada/salida a la casilla, o bien puede usar explícitamente símbolos de acceso de entrada y de salida (la casilla de macro es facultativa).

El ejemplo de la figura D-5.1.1 ilustra el LED/GR y el LED/PR para una macro con dos accesos de entrada y dos accesos de salida (en este ejemplo los accesos de entrada y de salidas están conectados a la casilla de la macro).

Esto significa que en la especificación principal LED/PR (figura D-5.1.2) hay uniones y etiquetas correspondientes. Obsérvese que la etiqueta probablemente será diferente para cada llamada de macro diferente.

La figura D-5.1.3 muestra dos definiciones de macro que definen un mecanismo de sincronización. Obsérvese el uso del parámetro seudoformal MACROID para hacer que los nombres de estado sean únicos. En la figura D-5.1.4 se ilustra una réplica del mismo ejemplo haciendo uso de los símbolos de acceso de entrada y de salida.

En el caso de macros anidadas, es decir, cuando hay una o más llamadas de macro dentro de una definición de macro, deberá observarse que la expansión de las macros situadas más al exterior no afecte la expansión de las situadas más al interior. Más precisamente, la expansión de macros anidadas deberá considerarse como si la expansión de una macro se completara antes de comenzar la expansión de las posibles llamadas de macro interiores.

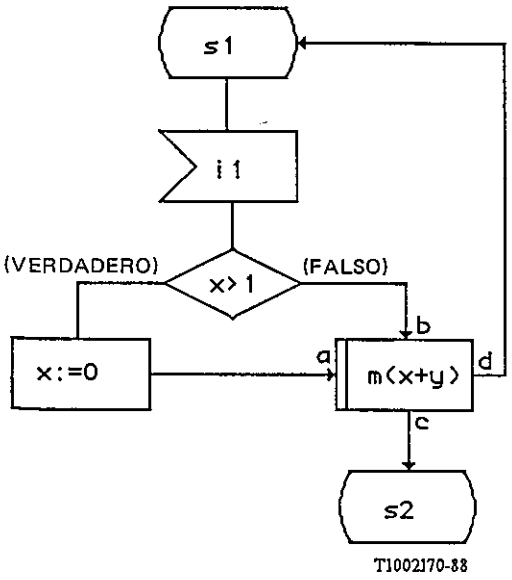



```
MACRODEFINITION m
   FPAR e,a,b,c,d;
   a: DECISION  e>1 ;
    (VERDADERO): JOIN b;
    (FALSO):     JOIN c;
       ENDDECISION;
   b: TASK x:=y;
   JOIN d;
ENDMACRO m;
```

LED/GR LED/PR

FIGURA D-5.1.1

Ejemplo de una definición de macro




```
...
STATE s1;
  INPUT i1;
    DECISION  x>1 ;
      (FALSO): JOIN BB;
 (VERDADERO): TASK x:=0;
              JOIN AA;
    ENDDECISION;
    MACRO m (x+y,AA,BB,CC,DD);
    CC: NEXTSTATE s2;
    DD: NEXTSTATE -;
...
```

LED/GR LED/PR

FIGURA D-5.1.2

Ejemplo de una llamada de macro

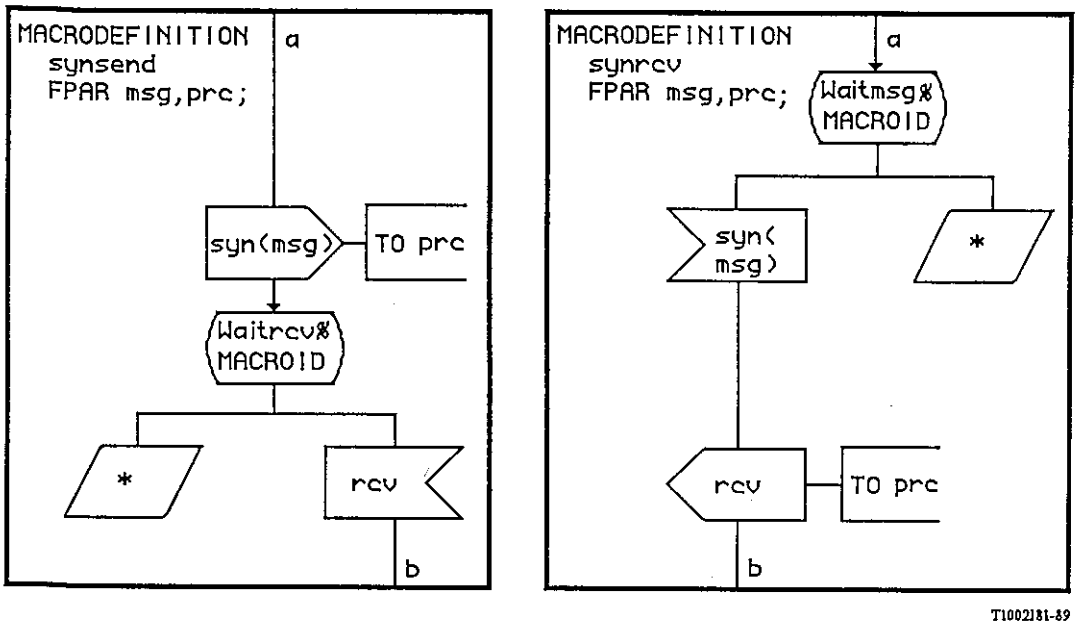


FIGURA D-5.1.3

Ejemplo de dos macros que utilizan MACROID

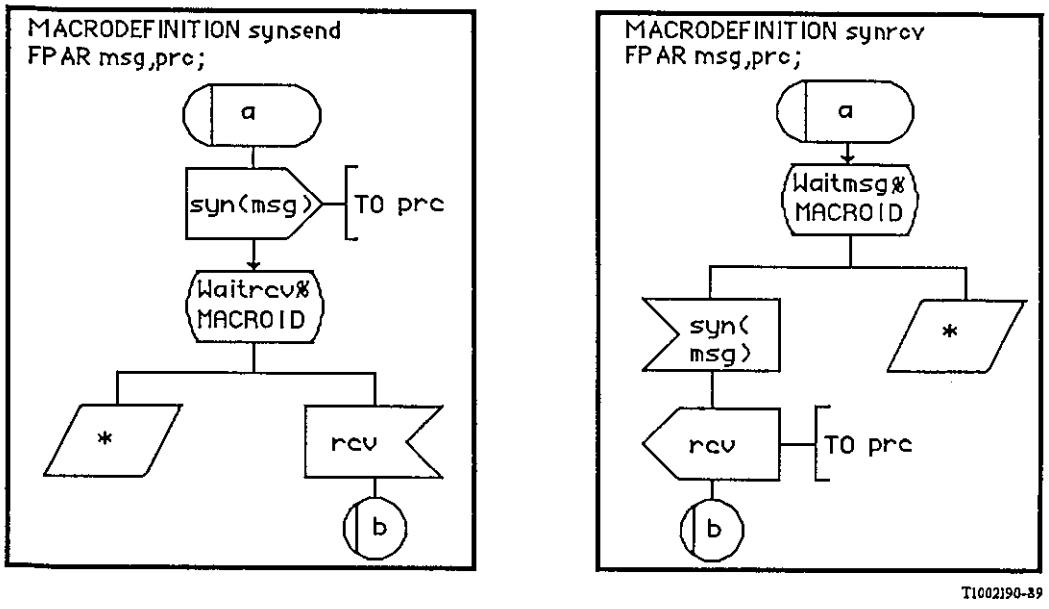


FIGURA D-5.1.4

Ejemplo del uso de símbolos de acceso de entrada y de salida

D.5.2 *Sistemas genéricos*

En LED es posible definir diferentes sistemas con una sola especificación, por medio de los parámetros del sistema. Los parámetros del sistema tienen un valor indefinido que puede ser suministrado desde afuera para obtener la definición de un sistema específico, de acuerdo con las necesidades del cliente.

Los parámetros del sistema son en realidad sinónimos exteriores y pueden utilizarse en todo lugar en que un sinónimo puede ser utilizado. Naturalmente, antes de interpretar un sistema debe asignarse un valor a todos los sinónimos exteriores. La figura D-5.2.1 muestra algunos ejemplos válidos de la utilización de sinónimos exteriores.

```
SYSTEM s;
   ...
   SYN inst.numb INTEGER = EXTERNAL;
   SYN rate.incr REAL = EXTERNAL;
   ...
   BLOCK b;
      PROCESS p (inst.numb); /* número de instancia paramétrico */
         ...
         val:=val+rate.incr; /* incremento paramétrico */
         ...
      ENDPROCESS p;
   ENDBLOCK b;
ENDSYSTEM s;
```

FIGURA D-5.2.1

Ejemplo del uso de parámetros de sistema

El LED proporciona también dos construcciones adicionales para permitir selecciones más eficaces condicionadas por sinónimos externos:

– construcción SELECT: selección condicional de un trozo de especificación. La condición se especifica por medio de una expresión booleana que tiene que poder evaluarse estáticamente, antes de la interpretación del sistema. El trozo de especificación se selecciona cuando la expresión es verdadera (TRUE).

– construcción ALTERNATIVE: permite la selección condicional de un trozo de especificación, entre dos o más trozos alternativos. Puede utilizarse solamente para seleccionar diferentes transiciones en el cuerpo de los procesos, procedimientos o servicios.

D.5.3 *Servicios*

D.5.3.1 *Generalidades*

El concepto de servicio tiene por objeto ofrecer la posibilidad de subdividir una definición de proceso sin introducir paralelismo. Cada servicio puede considerarse como una «función» ofrecida por el proceso. Es una definición de proceso parcial que representa un ′subcomportamiento′ del proceso. Ese ′subcomportamiento′ es una parte del comportamiento del proceso total. En consecuencia, el utilizar el concepto de servicio es una manera de estructurar un proceso.

Hay muchas razones para estructurar un proceso, por ejemplo, gestionar su complejidad y aumentar su facilidad de lectura. Pero subdividir es también un medio de aislar ciertas partes de un proceso y describirlas separadamente. Esas partes pueden ser ′subpartes′ de una función ofrecida por el sistema. De este modo, mediante el concepto de servicio puede aislarse una función del sistema, describiendo un conjunto de servicios subordinados en uno o más procesos.

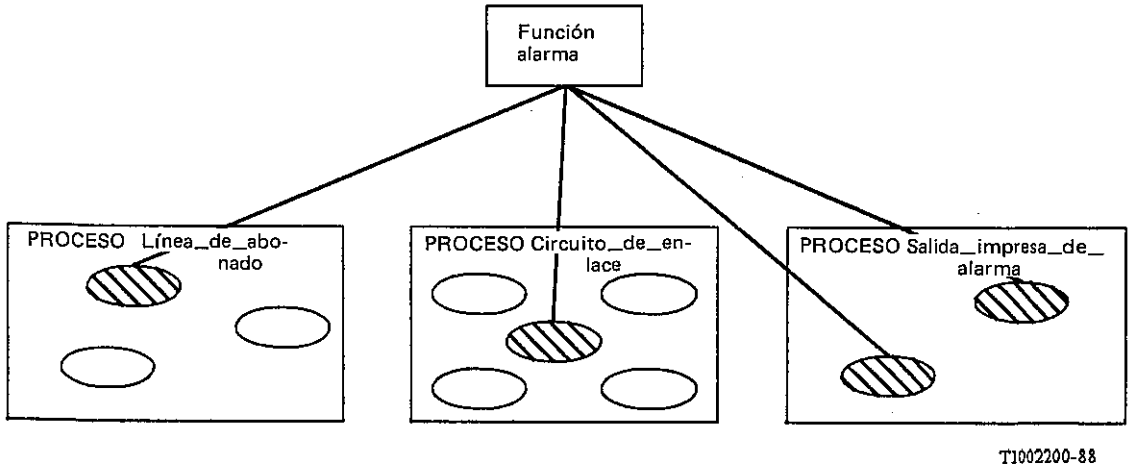


FIGURA D-5.3.1

Ejemplo informal que muestra cómo servicios de procesos diferentes pueden componer una función superior del sistema

Obsérvese que el lenguaje LED no tiene facilidades para componer una función de sistema eligiendo servicios de varios procesos. Esa composición depende del usuario y está enteramente fuera del lenguaje.

Dado que un servicio está aislado de los otros servicios y se describe como una máquina de estado finito con su propio espacio de estados, puede desarrollarse y cambiarse sin afectar los otros servicios. Sin embargo, debe observarse que los servicios en un proceso tienen a menudo datos comunes lo que significa que pueden afectarse unos a otros por manipulación de los datos.

Como el comportamiento de un proceso no cambia cuando se le subdivide en servicios, éstos son solamente una descripción alternativa del mismo comportamiento. Naturalmente, se puede decidir, en el momento del diseño, modelar el comportamiento en varios procesos en vez de uno solo. Sin embargo, esto proporciona un comportamiento diferente, al correr en paralelo varios procesos que no pueden compartir (leer y escribir) datos sin un intercambio complicado de señales.

Obsérvese que estructurar un proceso en servicios no significa que el proceso desaparecerá completamente. Sólo significa que el cuerpo del proceso, al describir el comportamiento del proceso, ha sido reemplazado completamente por cierto número de cuerpos de servicio. Los parámetros formales, las declaraciones y las definiciones a nivel de proceso se mantendrán. Además de ellos, cierto número de definiciones y declaraciones locales puede incluirse en las definiciones de servicio.

Una definición de servicio consiste en los siguientes elementos, algunos de ellos facultativos:

- nombre del servicio;
- conjunto de señales de entrada válidas: una lista de identificadores de señal que define las señales que pueden ser recibidas por el servicio;
- definiciones de procedimiento: la especificación de los procedimientos que son locales al servicio. También puede utilizarse una referencia de procedimiento;
- definiciones de datos: la especificación de los neotipos definidos por el usuario, sintipos y generadores locales al servicio;
- definiciones de variable: las declaraciones de las variables que son locales al servicio. Estas variables no pueden ser reveladas ni exportadas a otros procesos (las palabras clave REVEALED y EXPORTED no son permitidas). Para cada variable declarada, debe especificarse el identificador de género. Facultativamente, puede especificarse un valor inicial;
- definiciones de visión: la declaración de los identificadores de variable que pueden utilizarse para obtener los valores de las variables de propiedad de otros procesos. El género de variable debe especificarse para cada identificador de variable;
- definiciones de importación: la especificación de los identificadores de variable, de propiedad de otros procesos, que el servicio quiere importar. Debe especificarse el género de variable para cada identificador;

– definiciones de temporizador: se explican en el § D.3.11;

– definiciones de macros: se explican en el § D.5.1;

– cuerpo de servicio: la especificación del comportamiento real del servicio en términos de estados, entradas, salidas, tareas, etc. Comparado con un cuerpo de proceso, el cuerpo de servicio puede contener también emisión y recepción de señales prioritarias (§ D.5.3.2).

En LED/GR una definición de servicio se representa por medio de un diagrama de servicio. Un diagrama de servicio consiste en los siguientes elementos:

– un símbolo de casilla facultativo: un símbolo de forma rectangular que contiene todos los otros símbolos;

– el encabezamiento de servicio: la palabra clave SERVICE seguida por el identificador de servicio. El encabezamiento de servicio se sitúa en la esquina superior izquierda de la casilla;

– numeración facultativa de páginas (en la esquina superior derecha);

– símbolos de texto: un símbolo de texto se utiliza para contener definiciones de datos, variables, visión, importación y temporizadores que son locales al servicio;

– referencias de procedimiento: un símbolo de procedimiento que contiene un nombre de procedimiento que representa un procedimiento local al servicio y definido separadamente;

– diagrama de macros: véanse directrices en el § D.5.1;

– gráfico de servicio: la especificación del comportamiento real del servicio en términos de estados, entradas, salidas, tareas, etc. Comparado con un gráfico de proceso, un gráfico de servicio puede contener también emisión y recepción de señales prioritarias (§ D.5.3.2).

En la figura D-5.3.2 se da un ejemplo del concepto de servicio para el proceso sencillo ′Temporizador′. El proceso está estructurado en dos servicios que se referencian y definen en dos diagramas de servicio (figura D-5.3.3).

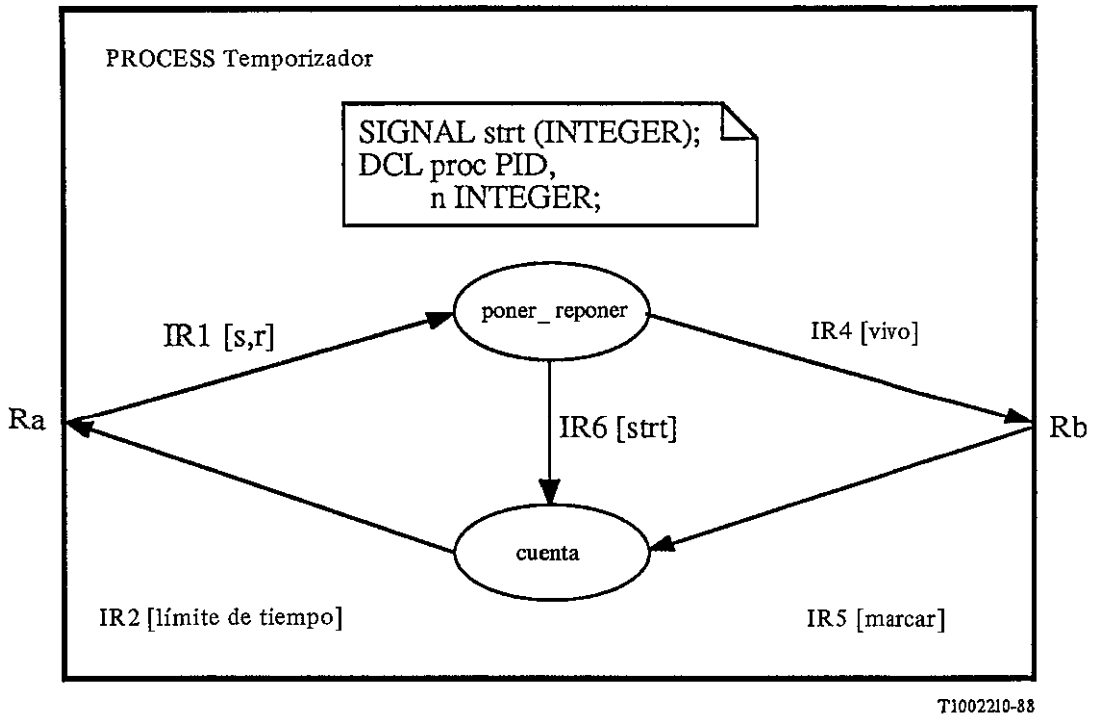


FIGURA D-5.3.2

Diagrama de proceso con servicios referenciados

Hay un interfaz interno entre los dos servicios por medio de la ruta de señales ′IR6′ que transporta una señal «prioritaria» (§ D.5.3.2).

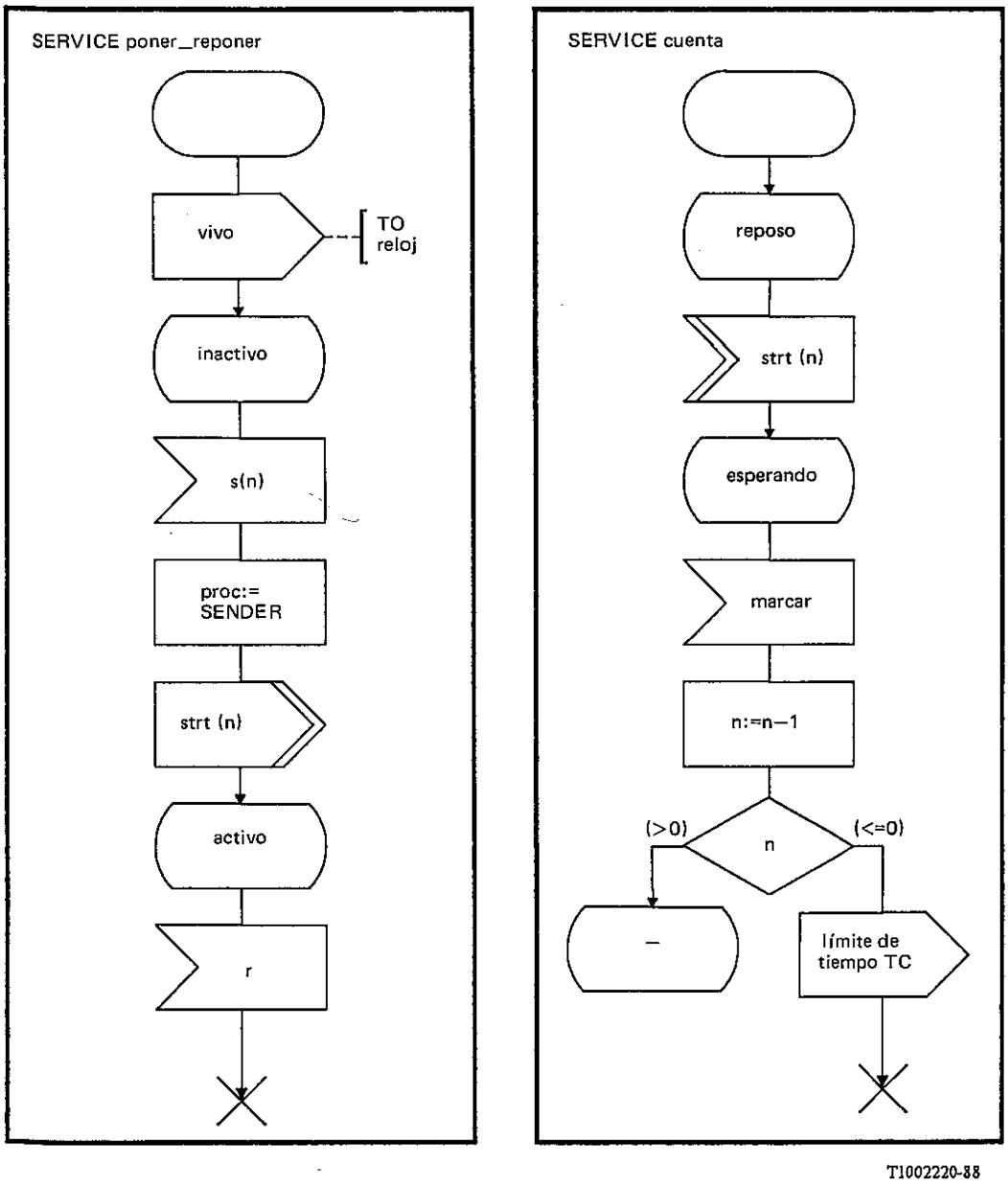


FIGURA D-5.3.3

Diagramas de servicio

Debe observarse que, al estar incluidas acciones (salida) en la transición de arranque del primer servicio, no se permiten acciones en la transición de arranque del segundo servicio.

En el § D.10 se presenta otro ejemplo del concepto de servicio.

D.5.3.2 *Señales prioritarias*

Se utiliza una señal prioritaria para la comunicación entre dos transiciones de servicios diferentes en un proceso cuando no se permite consumir señales exteriores (de otros procesos) en el intervalo de tiempo comprendido entre las transiciones. De esta manera las transiciones están ′concatenadas′.

La figura que sigue (§ D-5.3.4) es una ilustración de concatenación de transiciones cuando se utilizan señales prioritarias.

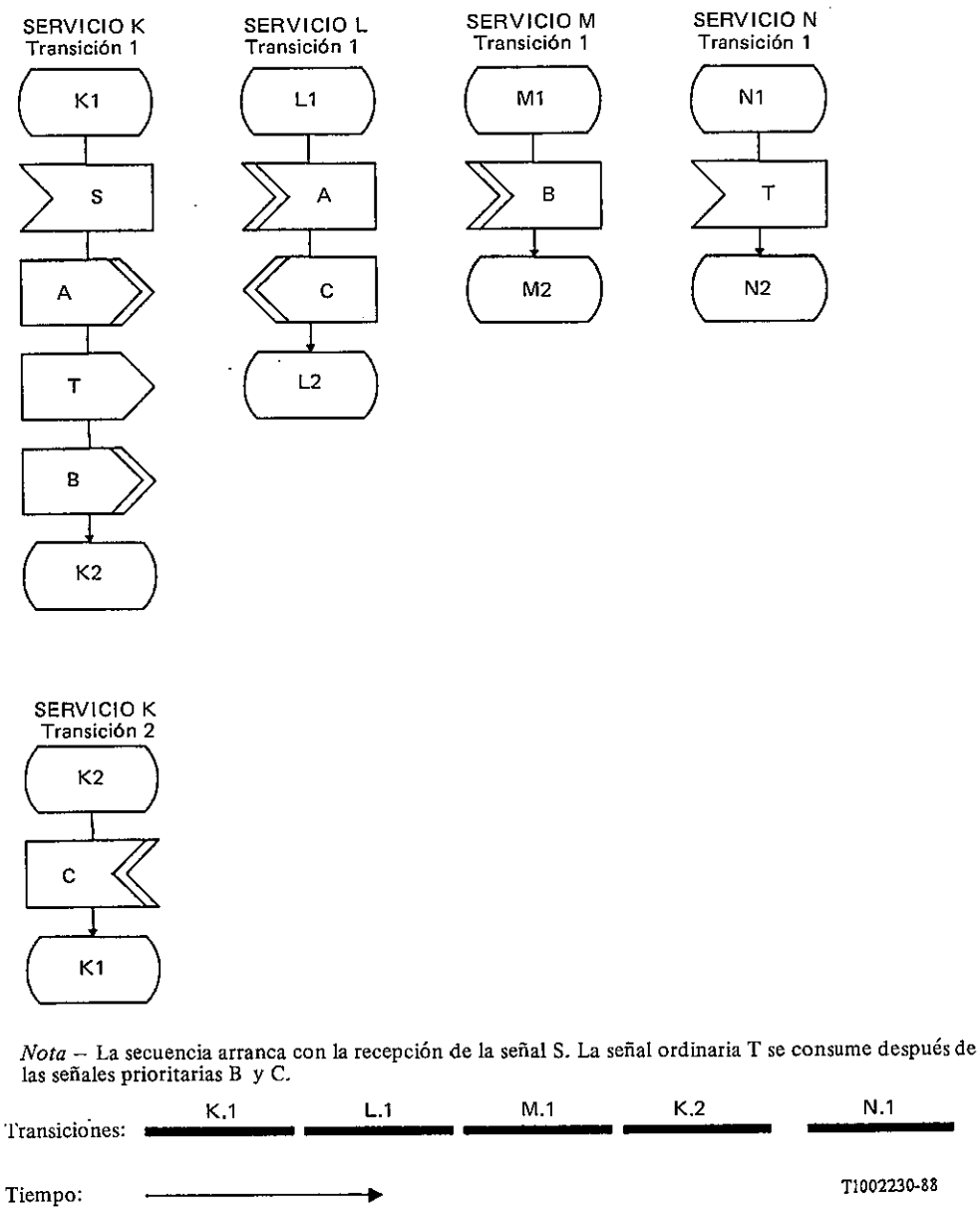


FIGURA D-5.3.4

Concatenación de transiciones de diferentes servicios de un proceso (LED informal)

En la recomendación se explica la construcción para obtener señales prioritarias utilizando la cola de entrada ordinaria. Al enviar las señales prioritarias a un estado preliminar, se pueden tratar como señales ordinarias, que causan transiciones a un estado principal (véase también § D.5.3.3.1).

El ejemplo que sigue es una ilustración de las consecuencias de utilizar señales prioritarias. El ejemplo se explica sin utilizar el modelo de «estado preliminar/principal».

El esquema secuencial (figura D-5.3.5) está tomado de la interacción entre los servicios en el proceso ′LINEA_DE_ABONADO′ descrito en el § D.10.2.

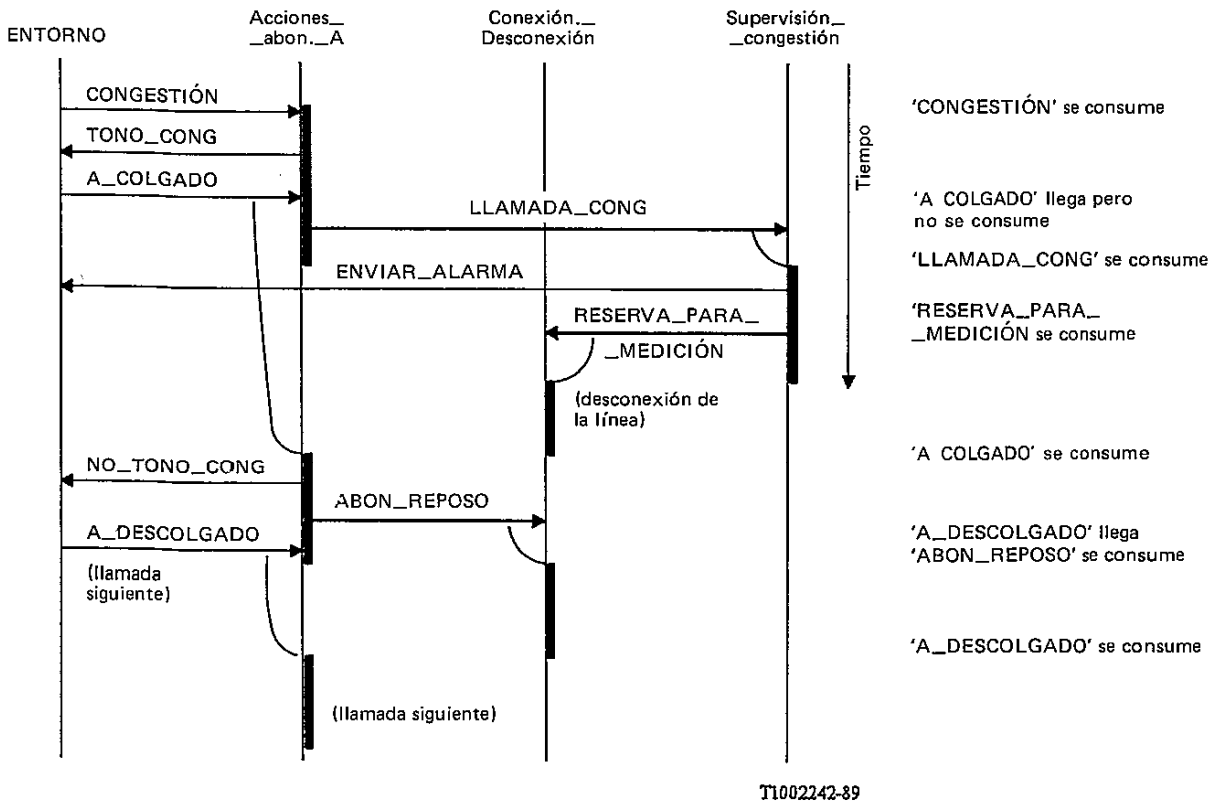


FIGURA D-5.3.5

Esquema secuencial que muestra cómo las señales prioritarias pueden cambiar el orden de las transiciones. Cada línea vertical gruesa indica/ejecución de una transición

El esquema contiene las señales a y desde el entorno del proceso y las señales prioritarias entre los servicios. La flechas de señal prioritaria son más gruesas. El orden (descendente) de las flechas de señal indica cómo están colocadas las señales en la cola de espera. Las líneas verticales gruesas indican cómo se ha ordenado la ejecución de las transiciones en el tiempo.

La secuencia arranca con la señal ′CONGESTIÓN′ del registro, lo que quiere decir que la llamada debe ser rechazada. Significa también que habrá un rechazo inmediato de la llamada siguiente.

La figura muestra que una transición, activada por una señal prioritaria, por ejemplo, ′RESERVA_PARA_MEDICIÓN′, arranca antes que una transición activada por una señal ordinaria, por ejemplo, ′A_COLGADO′, a pesar del orden contrario en la cola de espera. La transición activada por la señal ′RESERVA_PARA_MEDICIÓN′ desconecta la línea del abonado lo que quiere decir que la llamada siguiente (señal ′A_DESCOLGADO′) será rechazada inmediatamente.

En el esquema secuencial siguiente tenemos la misma situación pero el diagrama no usa señales prioritarias. Los servicios interfuncionan con señales ordinarias.

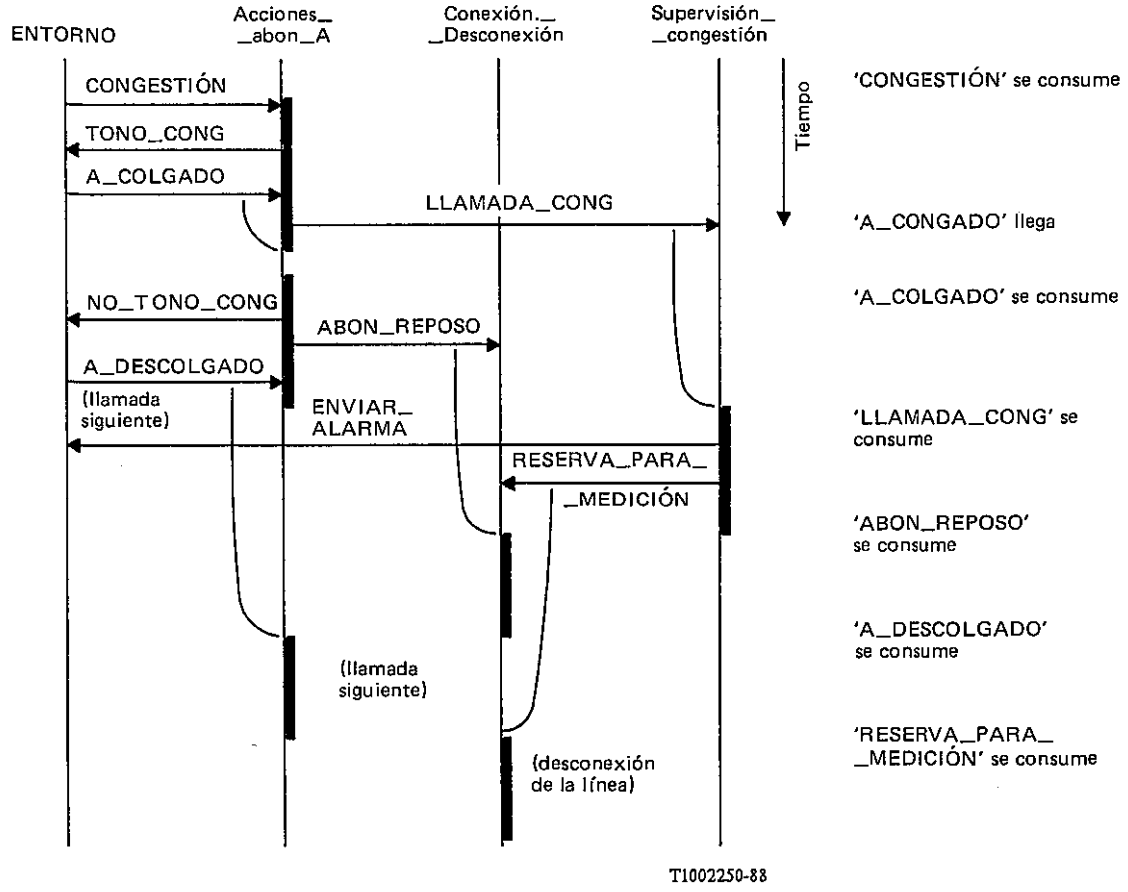


FIGURA D-5.3.6

Esquema secuencial con señales ordinarias y orden normal de las transiciones

Comparando con la figura D-5.3.5 podemos ver que se ha cambiado el orden de las transiciones. La llamada siguiente llega antes que la línea del abonado esté desconectada lo que quiere decir que la llamada no será rechazada inmediatamente.

D.5.3.3 *Transformación*

El servicio y la señal prioritaria son conceptos adicionales porque están compuestos (modelados) a partir de conceptos LED más básicos como proceso y señal. Para poder interpretar el servicio y la señal prioritaria en forma semántica ellos deben volverse a transformar a esos conceptos básicos. Normalmente esa transformación no necesita ser ejecutada por el usuario, por lo menos no manualmente, pero el usuario debe darse cuenta de la transformación. En un instrumento para la comprobación semántica del servicio la transformación podría ejecutarse automáticamente.

Después de la transformación los servicios han desaparecido y se reemplazan por una definición ampliada de proceso que puede ser interpretada en forma semántica. La transformación ha definido el comportamiento.

D.5.3.3.1 *Transformación de estados*

En la recomendación se dan reglas específicas para la transformación de estados.

El resultado de estas reglas es que el número de estados del proceso es el producto del número de estados de los servicios.

Adicionalmente a esto, el uso de las señales prioritarias duplicará este producto porque cada estado transformado se subdivide en un estado preliminar y un estado principal. Esta duplicación del número de estados debido a las señales prioritarias no se considera en el ejemplo de transformación de estados que sigue.

En muchos casos un gran número de estados del proceso «transformado» no son pertinentes y el espacio de estados puede ser reducido. Se habla de esto en el ejemplo que sigue, tomado del § D.10.2.

En el proceso ′LINEA_DE_ABONADO′ tenemos cuatro servicios, ′acciones_abonado_A′, ′acciones_abonado_B′, ′conexión.desconexión′ y ′supervisión_congestión′. El número de estados es 10, 5, 3 y 2, es decir, 20 en total.

Aplicando a estos servicios las reglas para la transformación de estado, tendremos 300 estados (10*5*3*2) en el proceso, lo que significa 300 tuplas de nombres. La dimensión de la tupla es 4 (4 servicios). Las posiciones en la tupla están dispuestas de acuerdo con la figura D-5.3.7.

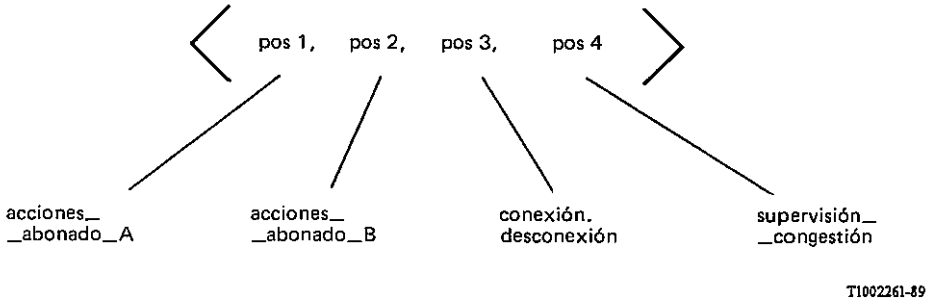


FIGURA D-5.3.7

Configuración de posiciones en una tupla de nombre

Todos los nombres de estado posibles en las diferentes posiciones dan 300 combinaciones.

Ej.<A_REPOSO, B_REPOSO, CONECTADO, SIN_ALARMA>

<ESPERAR_CON, B_REPOSO, CONECTADO, SIN_ALARMA>

<ESPERAR_PRIMERA_CIFRA, B_REPOSO, CONECTADO, SIN_ALARMA>

. . . .

<ESPERAR_A_COLGADO_2, B_REPOSO, CONECTADO, SIN_ALARMA>

<A_REPOSO, B_LLAMANDO, CONECTADO, SIN_ALARMA>

<A_REPOSO, B_CONVERSACIÓN, CONECTADO, SIN_ALARMA>

. . . .

<A_REPOSO, ESPERAR_B_COLGADO, CONECTADO, SIN_ALARMA>

. . . .

. . . .

<ESPERAR_A_COLGADO_2, ESPERAR_B_COLGADO, TOMADO, ALARMA>

Muchas de estas combinaciones no son pertinentes porque una línea de abonado nunca puede ser usada como línea de abonado-A y línea de abonado-B de la misma llamada. Esto significa que los diez estados de ′acciones_abonado A′ pueden solamente combinarse con un estado de ′acciones_abonado_B′ que es ′B_REPOSO′. Significa también que los cinco estados de ′acciones_abonado_B′ pueden combinarse solamente con un estado de ′acciones_abonado_A′ (A_REPOSO). El número de estados relevantes entonces es (10*1*3*2) + (1*5*3*2) = 90.

Una reducción del espacio de estados, como la arriba mostrada, está subordinada al ejemplo presente y no puede formalizarse en reglas generales aplicables a una transformación automática. Sin embargo, si la transformación se ejecuta en forma manual, el espacio de estados puede probablemente reducirse. Así, cuando se compara la complejidad de un proceso diseñado sin servicios con la complejidad del mismo proceso subdividido en servicios, debe utilizarse el proceso con el espacio de estados reducido para obtener una comparación justa. En la mayor parte de los casos la utilización de servicios reducirá apreciablemente el número de estados.

D.5.3.3.2 *Transformación de transiciones*

En la recomendación se dan reglas específicas para la transformación de transiciones. El ejemplo que sigue es una ilustración de transformación. El proceso del ejemplo es una especificación de un protocolo sencillo. El proceso está subdividido en dos servicios enviar y recibir.

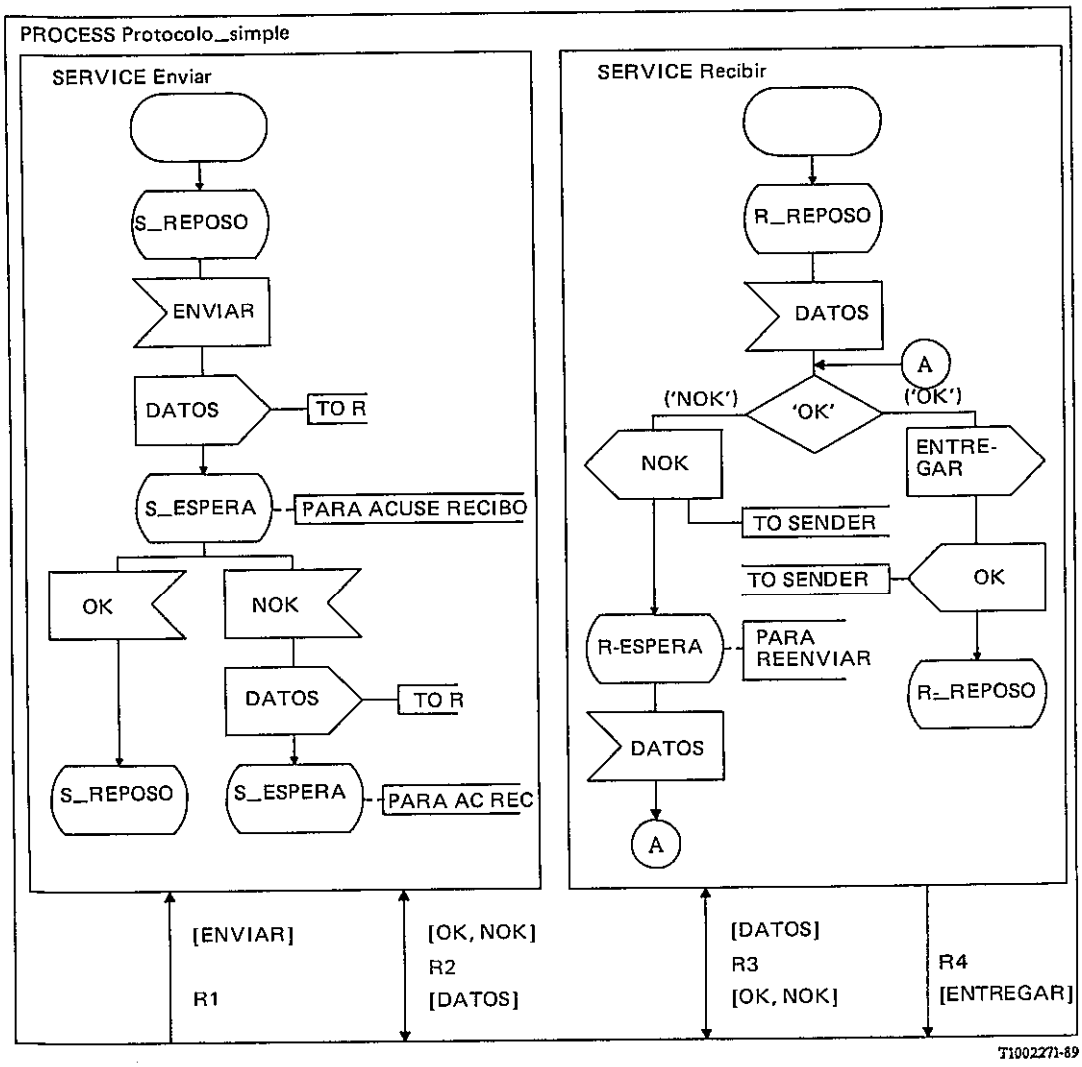


FIGURA D-5.3.8

Diagrama de proceso que incluye dos diagramas de servicio

La figura D-5.3.9 es un diagrama general para los dos servicios. Las transiciones están numeradas 1 a 7.

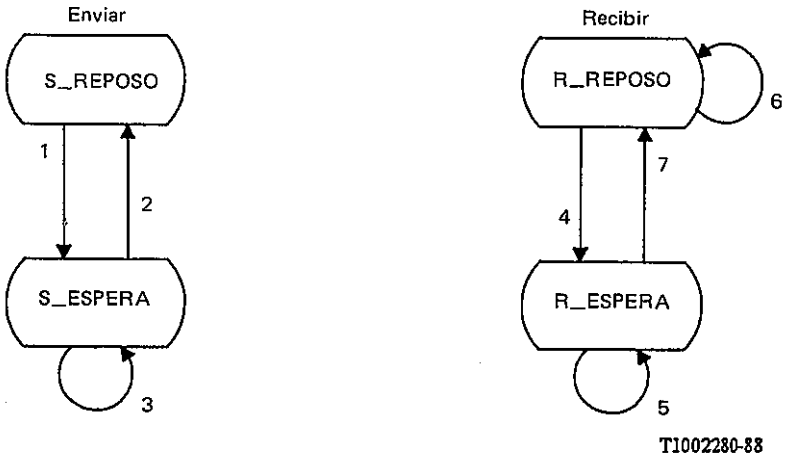


FIGURA D-5.3.9

Diagramas generales de estados con transiciones numeradas

Aplicando las reglas formales para la transformación de estados y transiciones el resultado es el siguiente diagrama general de estados para el proceso. No se han usado señales prioritarias, así que no se requiere, y no se muestra, la subdivisión adicional en estados preliminar y principal.

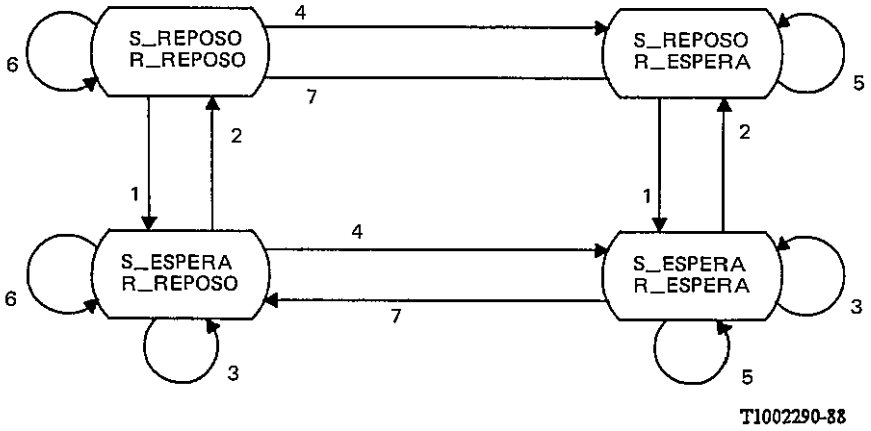


FIGURA D-5.3.10

Diagrama general de estados para los servicios transformados

El número de estados no puede reducirse y el gráfico de proceso se muestra en la figura D-5.3.11. Los nombres de los estados en el gráfico del proceso corresponden a las tuplas de nombres de la figura D-5.3.10.

Ejemplo – SR.RR corresponden a S_REPOSO, R_-REPOSO_S.

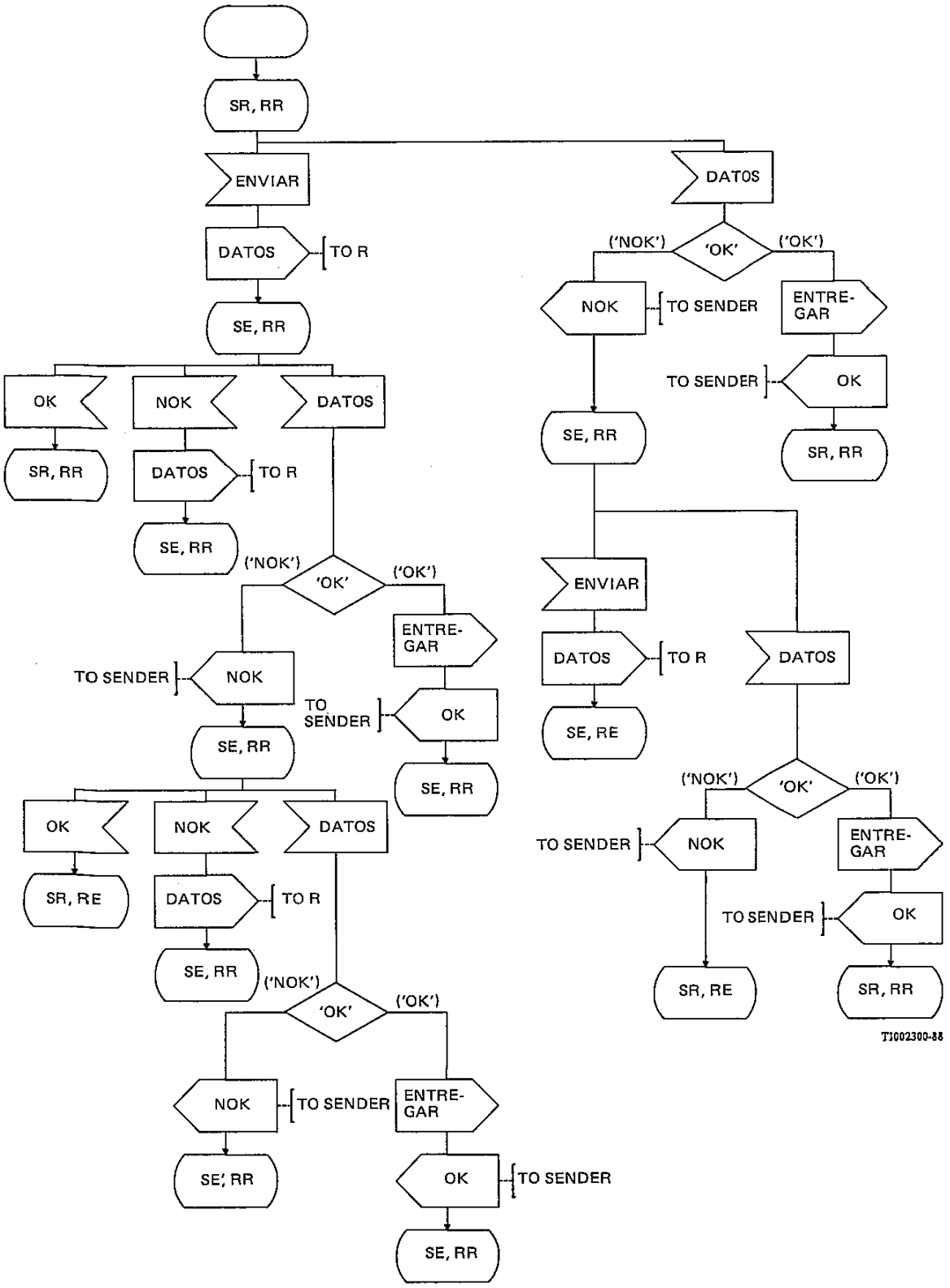


FIGURA D-5.3.11

Gráfico de proceso transformado a partir de dos gráficos de servicio

D.5.4 *Directrices para la representación por medio de estados y elementos pictográficos*

Este capítulo explica la representación por medio de estados del LED/GR y los elementos pictográficos.

La primera sección (§ D.5.4.1) incluye comentarios generales sobre la representación por medio de estados; sobre sus características, aplicaciones adecuadas y variaciones.

La segunda sección (§ D.5.4.2) explica la descripción del estado mediante elementos pictográficos.

D.5.4.1 *Comentarios generales sobre representación por medio de estados*

El LED/GR da a elegir entre tres versiones diferentes para describir un diagrama de proceso.

La primera se llama la versión del LED/GR basada en las transiciones, en la que las transiciones se describen exclusivamente mediante símbolos de acción explícitos.

La segunda se llama la versión del LED/GR basada en los estados o ampliación pictográfica del LED basada en los estados, en la que los estados de un proceso se describen utilizando pictogramas de estado, y las transiciones solamente se deducen de la diferencia entre los estados de origen y de terminación.

La última se llama la versión combinada y es una combinación de las dos anteriores.

En el anexo E de este fascículo se dan ejemplos de estas tres versiones.

La versión del LED/GR basada en las transiciones es adecuada cuando la secuencia de acciones es importante y no lo son las descripciones de estado detalladas.

La versión basada en los estados describe el estado en forma declarativa y en detalle, por lo tanto es adecuada cuando un estado del proceso es más importante que las secuencias de acciones dentro de cada transición, cuando es conveniente tener una explicación pictográfica intuitiva y cuando los recursos y sus relaciones asociados al estado son de interés.

Los pictogramas de estado se describen generalmente con elementos pictográficos que indican recursos pertinentes del estado actual del proceso. Este método es utilizable directamente en aplicaciones para las cuales se han definido elementos pictográficos adecuados, por lo que el usuario puede aplicar esta representación a todas las aplicaciones definiendo, si es necesario, los elementos pictográficos apropiados.

La versión combinada es adecuada cuando entran en consideración tanto la secuencia de acciones dentro de cada transición como descripciones de estado detalladas.

D.5.4.2 *Pictograma de estado y elemento pictográfico*

D.5.4.2.1 *Elemento pictográfico y texto calificador*

Si se elige la opción de pictograma de estado, un pictograma de estado consta de elementos pictográficos y texto calificador como se ilustra en la figura D-5.4.1 a) a D-5.4.1 d).

Esta combinación proporciona pictogramas de estado que pueden entenderse. Por ejemplo, la figura D-5.4.1 a) da el significado de un receptor de dígitos de marcación manipulado por el proceso; el ejemplo b) da el significado de un emisor de tono de invitación a marcar que está enviando una señal permanente al entorno.

Obsérvese que las señalas de salida (señales no permanentes) y los recursos pertinentes no se describen en los pictogramas de estado; las señales de salida pueden describirse en diagramas de transición.

El ejemplo c) muestra un temporizador cuya expiración siempre está representada por una entrada. Obsérvese que el elemento pictográfico recomendado para el temporizador incorpora la señal de entrada pertinente t1.

El último ejemplo d) significa que un registrador de mensajes vocales está ahora registrando.

La identidad del recurso puede ser muy abreviada, y cuando sea posible deberá colocarse dentro del elemento pictográfico apropiado. De este modo se distinguen claramente los elementos pictográficos que se califican.

D.5.4.2.2 *Compleción de los pictogramas de estado*

A cada pictograma de estado debe asignarse un número suficiente de elementos pictográficos que permitan indicar:

a) los objetos o recursos que está considerando el proceso en el estado vigente. Ejemplos de recursos son: trayectos de conmutación, receptores de señalización, emisores de señales permanentes y módulos de conmutación;

b) si el proceso está supervisado en ese momento por uno o más temporizadores;

c) en caso de que el proceso en cuestión concierna al tratamiento de la llamada, si hay o no tasación en curso y los abonados afectados por la tasación en este estado de la llamada;

d) los objetos de propiedad de otro proceso (entorno) que se considera que tienen relaciones con recursos del proceso durante el estado vigente;

e) las señales de salida permanentes que salen durante este estado;

f) la relación entre señales y recursos en el estado;

g) el estado de los recursos relevantes para el estado vigente del proceso.

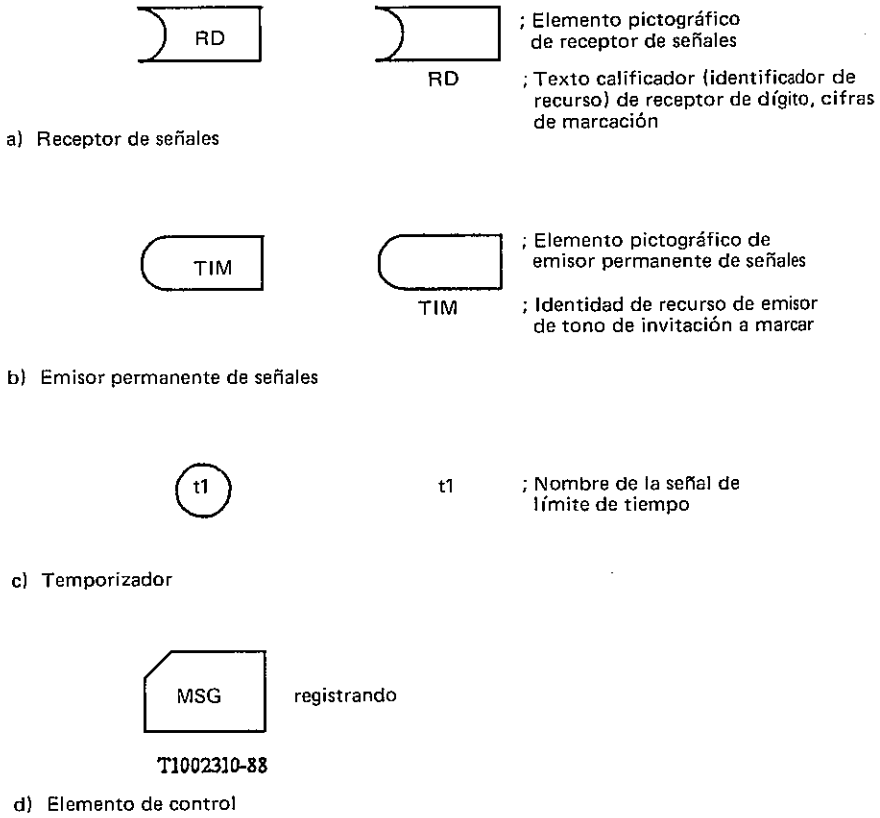


FIGURA D-5.4.1

Ejemplos de elemento pictográfico con texto calificador

D.5.4.2.3 *Ejemplo*

Un ejemplo de la aplicación del principio mencionado se muestra en el pictograma de estado de la figura D-5.4.2. Puede observarse que durante este estado:

a) los recursos considerados por el proceso consisten en un receptor de dígitos de selección (marcación), un emisor de tono de invitación a marcar, un aparato telefónico de abonado perteneciente al entorno y trayectos de conmutación que conectan estos elementos;

b) un temporizador t0 está supervisando este proceso en ese momento;

c) no hay tasación de comunicación en curso;

d) el abonado ha sido identificado como abonado A, pero no se ha considerado otra información acerca de su categoría;

e) se esperan las señales de entrada siguientes: A_colgado (es decir, la señal de teléfono colgado), cifra (es decir, la cifra marcada) y t0 (es decir, el temporizador de supervisión t0 está funcionando);

f) la señal permanente de salida TIM (es decir, el tono de invitación a marcar) ha sido presentada a la salida antes de este estado y durante el mismo.

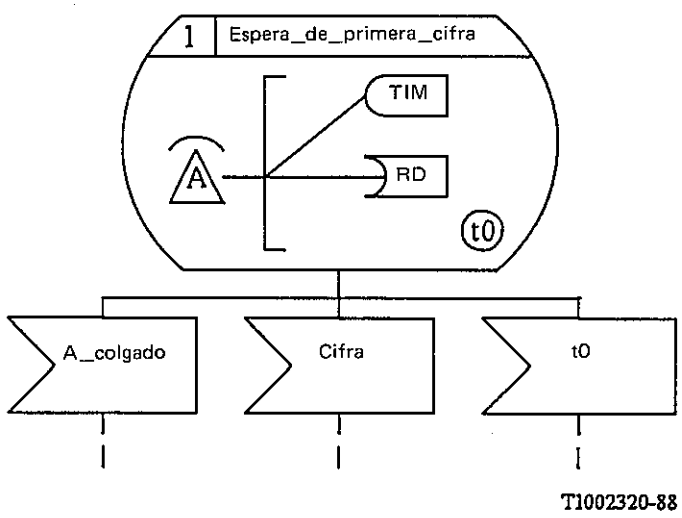


FIGURA D-5.4.2

Ejemplo de un estado de un proceso de tratamiento de llamada

D.5.4.2.4 *Verificación de la coherencia de los diagramas LED con elementos pictográficos*

No cabe duda de que un pictograma de estado es más conciso ni de que, en cierto modo, pone más información ante los ojos del lector, pero al mismo tiempo uno tiene que estudiar detenidamente el recurso para deducir el conjunto exacto de operaciones que tienen lugar en la transición.

Además, no es posible decir, mirando solamente el pictograma de estado, el orden de las acciones en la transición.

Los elementos pictográficos indicados fuera del símbolo de frontera de bloque son elementos que no están directamente controlados por el proceso dado, y los elementos pictográficos indicados dentro del símbolo de frontera de bloque son elementos que están controlados directamente por el proceso dado. Por ejemplo, el proceso de llamada especificado parcialmente en la figura D-5.4.3 puede atribuir o suprimir la atribución del emisor de tono de llamada, el emisor de señal de llamada y los trayectos de conversación, así como arrancar o parar el temporizador t4, pero no puede cambiar ninguna de las condiciones del teléfono del abonado.

En el diseño de la lógica a partir de una especificación LED con elementos pictográficos, sólo los elementos pictográficos indicados dentro de la frontera de bloque influirán en las acciones del proceso que tienen lugar durante las transiciones. Normalmente, los elementos pictográficos que aparecen fuera de la frontera de bloque se incluyen en un pictograma de estado por las razones siguientes:

a) porque indican recursos y estado del entorno que tienen relación con señales de entrada del proceso durante el estado en cuestión;

b) para mejorar la inteligibilidad del diagrama.

D.5.4.2.5 *Utilización del símbolo de temporizador*

Empléense o no elementos pictográficos, una expiración de temporizador se representa siempre por una entrada.

La presencia de un símbolo de temporizador en un pictograma de estado implica que un temporizador está funcionando durante ese estado.

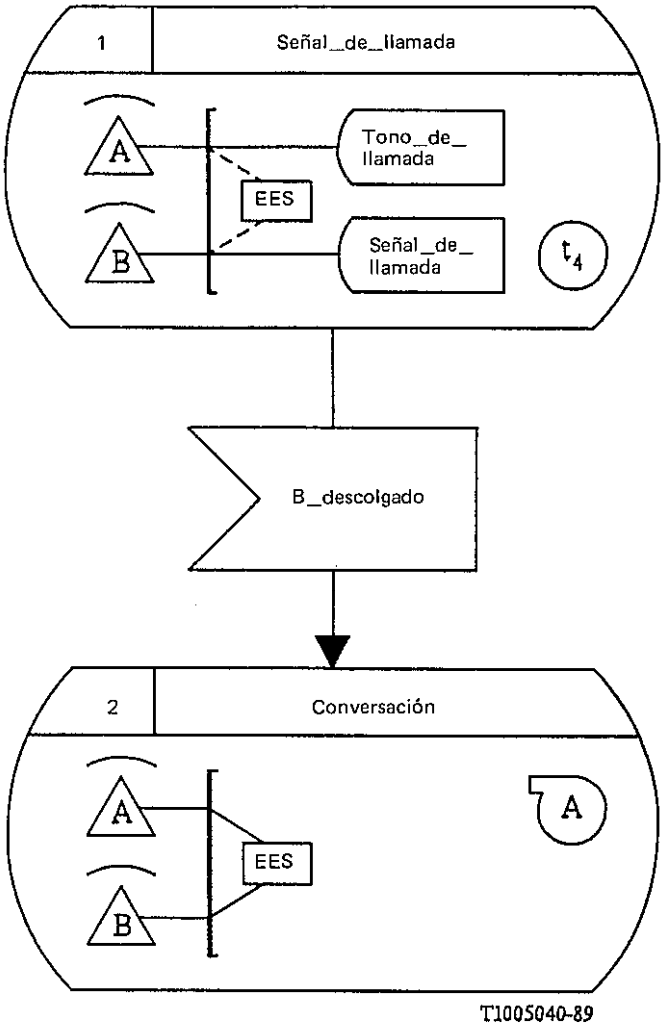


FIGURA D-5.4.3

Ejemplo de transición entre dos estados en que todas las acciones de proceso se deducen de las diferencias entre los pictogramas de estado

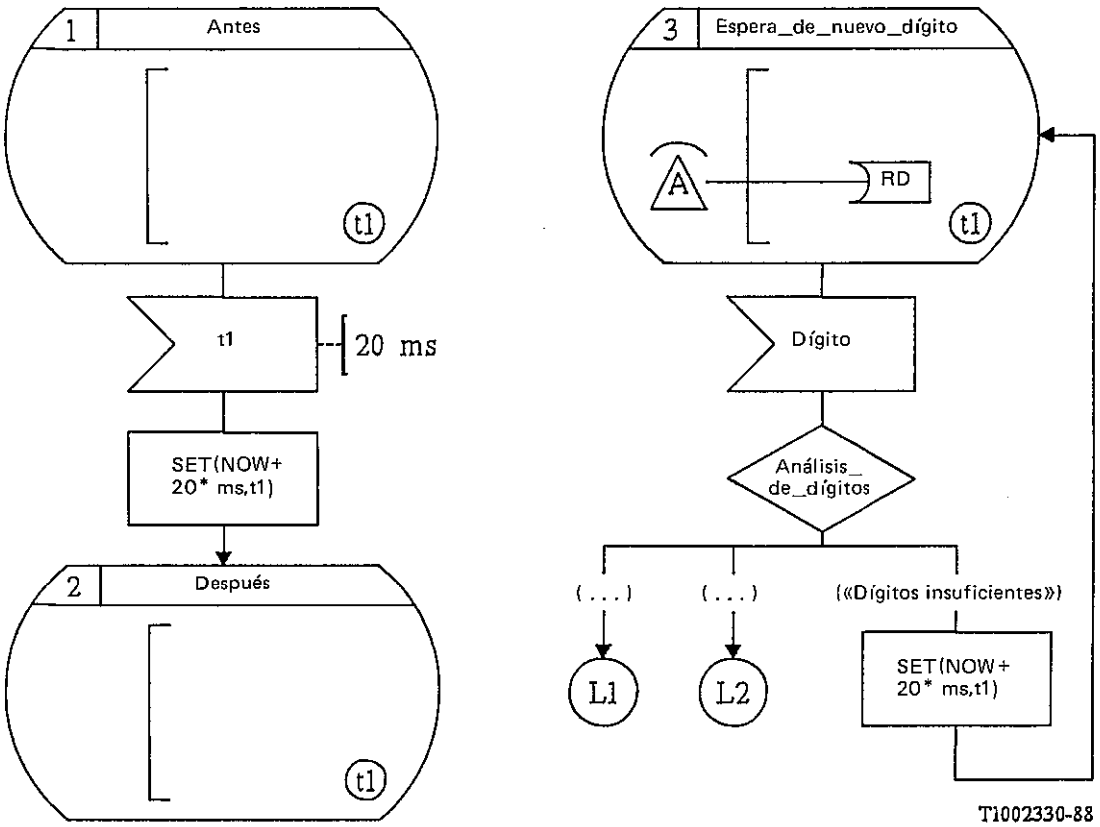
De acuerdo con el principio general enunciado en las Recomendaciones, el arranque, parada, rearranque y expiración de temporizadores se indican utilizando elementos pictográficos de la manera siguiente:

a) Para indicar que un temporizador arranca durante una transición dada, el símbolo de temporizador debe aparecer en el pictograma de estado que corresponde al fin de la transición, pero no en el pictograma de estado que corresponde al comienzo de la transición.

b) Para indicar que un temporizador ha parado en el curso de una transición, el símbolo de temporizador debe aparecer en el pictograma de estado que corresponde al comienzo de la transición, pero no en el pictograma de estado que corresponde al fin de la transición.

c) Para indicar que un temporizador ha rearrancado durante una transición, debe figurar en esta transición un símbolo explícito de tarea (en la figura D-5.4.4 se presentan dos ejemplos).

d) La expiración del periodo de un temporizador determinado se indica mediante un símbolo de entrada asociado con un estado en que el pictograma de estado incluye el símbolo de temporizador correspondiente. Desde luego, si es preciso, más de un temporizador puede estar supervisando el mismo proceso en un momento dado (véase la figura D-5.4.5).



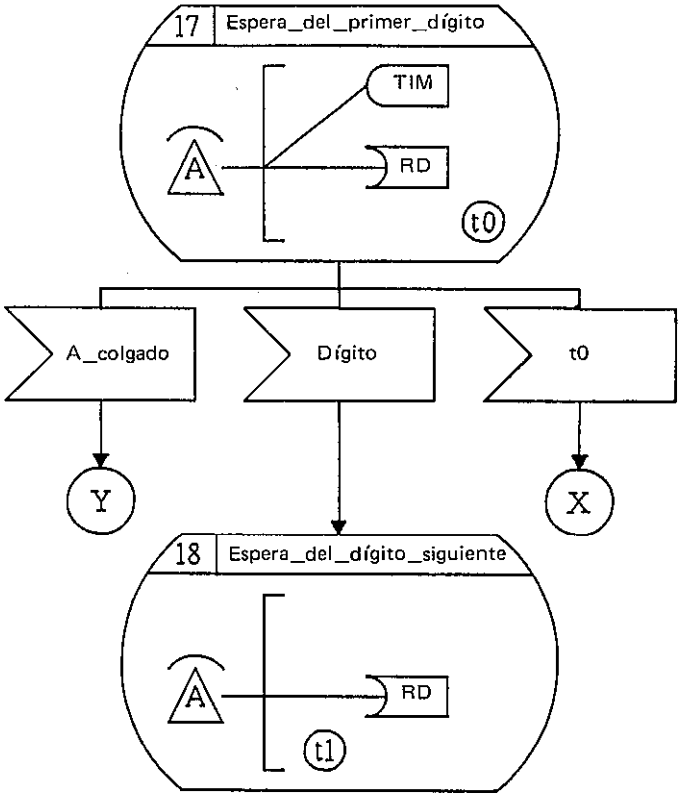
a) Rearranque de un temporizador después de expirar el periodo del temporizador

b) Rearranque de un temporizador cuando el periodo del temporizador no ha expirado todavía

Nota – Cada temporizador t1 tiene dos condiciones que se excluyen mutuamente: activo e inactivo.

FIGURA D-5.4.4

Rearranque de un temporizador



Nota – El temporizador t0 supervisa la llegada del primer dígito, se suprime el tono de invitación a marcar y se para el temporizador t0. El temporizador t1 sigue supervisando la llegada de un número suficiente de dígitos que permita el encaminamiento de la llamada.

FIGURA D-5.4.5

Ejemplo de utilización de dos temporizadores de supervisión en un mismo estado

D.5.5 *Diagramas auxiliares*

Para facilitar la lectura y comprensión de los diagramas de proceso grandes o complejos el autor puede proporcionar diagramas auxiliares informales. El objeto de esos documentos es dar un diagrama general o una descripción simplificada del comportamiento del proceso (o de parte de él). Los documentos auxiliares no reemplazan los documentos LED, pero sirven como introducción a ellos.

En esta sección se ofrecen ejemplos de algunos diagramas auxiliares usados corrientemente, a saber, diagrama general de estados, matriz de estados/señales, esquema de secuencias. (El diagrama de árbol de bloques descrito en el § D.4.4 también es un diagrama auxiliar.)

D.5.5.1 *Diagrama general de estados*

La finalidad del diagrama general de estados es dar una idea general de los estados de un proceso y de las transiciones que son posibles entre los mismos. Como su objeto es presentar una imagen general, pueden omitirse los estados o transiciones que «no son importantes».

Los diagramas se componen de símbolos de estado, arcos dirigidos que representan las transiciones y, facultativamente, símbolos de comienzo y parada.

El símbolo de estado debe contener el nombre del estado referenciado. Puede contener varios nombres de estado, y puede utilizarse un asterisco (*) como notación que indica todos los estados.

A cada uno de los arcos dirigidos puede asociarse el nombre de la señal o conjunto de señales que causan la transición lo mismo que las posibles salidas emitidas durante la transición.

En la figura D-5.5.1 se ofrece un ejemplo de diagrama general de estados.

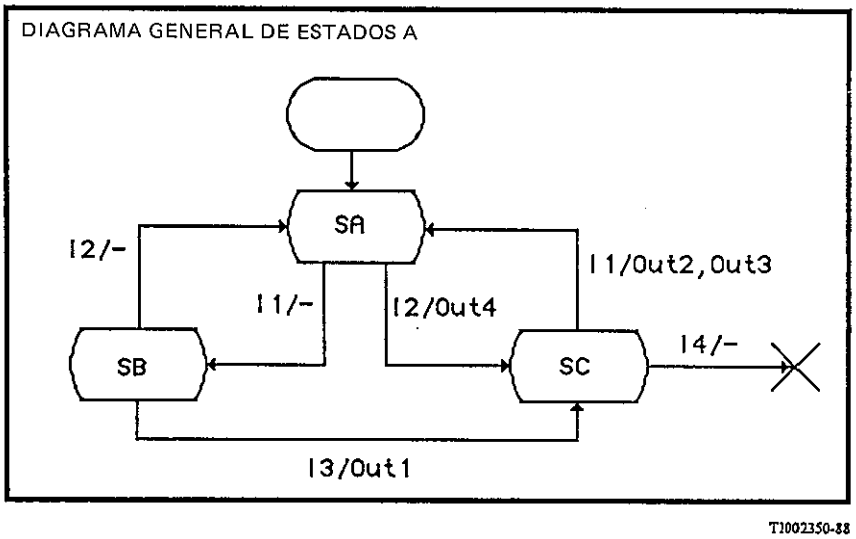


FIGURA D-5.5.1

Ejemplo de un diagrama general de estados

El diagrama general de estados de un proceso puede dividirse en varios diagramas relativos a diferentes aspectos, como por ejemplo «caso normal», «tratamiento-en-caso-de-fallo», etc.

FIGURA D-5.5.2

Ejemplo de un diagrama general de estados dividido

D.5.5.2 *Matriz de estados/señales*

La matriz de estados/señales es una alternativa al diagrama general de estados y contiene exactamente la misma información.

El diagrama consiste en una matriz bidimensional, en uno de cuyos ejes están indizados todos los estados de un proceso, y en el otro todas las señales de entrada válidas para ese proceso. En cada elemento de la matriz se da el estado siguiente junto con las salidas posibles durante la transición. Puede ofrecerse una referencia acerca de dónde se encuentra la combinación determinada por los índices, en el caso de que exista.

El elemento que corresponde al estado ficticio «START» y a la señal ficticia «CREATE» se utiliza para mostrar cuál es el estado inicial del proceso.

ESTADO/SEÑAL A

ESTADO / SEÑAL	"START"	SA	SB	SC
"CREATE"	SA/-	-	-	-
I1	-	SB/-	-	SA/Out2,Out3
I2	-	SC/Out4	SA/-	-
I3	-	-	SC/Out1	-
I4	-	-	-	"STOP"/-

FIGURA D-5.5.3

Ejemplo de una matriz de estados/señales

La matriz puede dividirse en subpartes que aparezcan en páginas diferentes. Las referencias son las que normalmente utiliza el usuario en la documentación.

Es preferible agrupar las señales y estados de manera tal que cada parte de la matriz cubra un aspecto del comportamiento del proceso.

D.5.5.3 *Esquema secuencial*

El esquema secuencial puede proporcionarse para mostrar las secuencias permitidas de señales intercambiadas entre servicios, procesos, bloques y su entorno.

Este esquema tiene por objeto dar una idea general del intercambio de señales entre las partes del sistema. Puede representar el intercambio completo de señales o sólo partes de él, según los aspectos que se desee resaltar.

Las columnas del esquema indican las entidades que se comunican (servicios, procesos, bloques o el entorno).

Sus interacciones se representan mediante un conjunto de líneas dirigidas, cada una de las cuales representa una señal o un conjunto de señales.

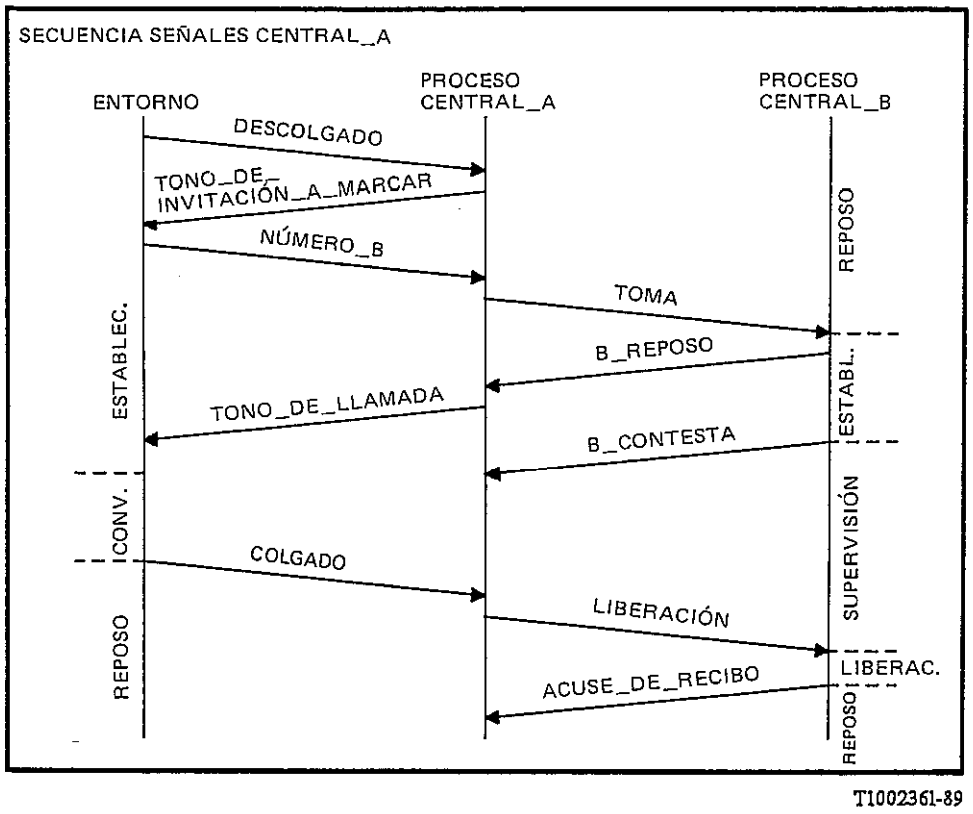


FIGURA D-5.5.4

Ejemplo de un esquema secuencial

Se puede anotar cada secuencia para que quede claro qué conjunto de informaciones se intercambia. Cada línea se anota con la información pertinente (nombres de señales o procedimientos).

En las columnas se puede colocar un símbolo de decisión para indicar que la secuencia siguiente es válida si la condición indicada es cierta. En este caso el símbolo de decisión suele aparecer varias veces, indicando las diferentes secuencias que resultan de los diferentes valores de la condición.

Este diagrama puede mostrar completamente todas las secuencias de señales intercambiadas o solamente un subconjunto significativo de ellas.

Una aplicación útil de este diagrama es la representación de la interacción entre servicios que resulta de la partición de un proceso.

Por lo general, los esquemas de secuencia no cubren todas las secuencias posibles; ellos son a menudo preliminares a la definición completa.

D.6 *Definición de datos en el LED*

D.6.1 *Directrices sobre datos en LED*

Esta sección da más información sobre los conceptos definidos en el § 5 de la Recomendación sobre el LED. La principal diferencia entre esta sección y la anterior Z.104 es que se ha refinado, aclarándola y armonizándola con la ISO. Se han cambiado algunas de las palabras clave y hay algunas ampliaciones, pero las semánticas son consistentes con el significado previsto en el Libro Rojo. Se hace observar que el uso de los datos en § 2, § 3 y § 4 de la nueva Recomendación (antes Z.101-Z.103) no ha cambiado.

D.6.1.1 *Introducción general*

El método en que se basan los tipos de datos del LED es el «tipo de datos abstractos». Este es un método donde no se describe COMO puede implementarse un tipo, sino que solamente se dice CUAL será el resultado de los operadores cuando se aplican a valores.

Cuando se definen datos abstractos, cada segmento de la definición, denominado «definición parcial de tipo», se introduce mediante la palabra clave NEWTYPE (NEOTIPO). Cada definición parcial de tipo afecta a las otras de modo que todas las definiciones parciales de tipo a nivel de sistema constituyen una sola definición de tipo de datos. Si se introducen más definiciones parciales de tipo a un nivel más bajo (por ejemplo, a nivel de bloque) ellas constituyen, junto con las definiciones de nivel más alto, una definición de tipo de datos. Esto es, en cualquier punto de la especificación hay solamente *una* definición de tipo de datos.

Esencialmente, la definición de tipo de datos consiste en tres partes:

a) definiciones de género;

b) definiciones de operador;

c) ecuaciones.

Cada una de estas partes se explica más adelante. La definición de tipo de datos está estructurada en definiciones parciales de tipo de datos y cada una de ellas introduce un género. Las definiciones de operador y las ecuaciones están repartidas entre las definiciones parciales de tipo de datos.

D.6.1.2 *Géneros*

Un género («sort») es un conjunto de valores. Este conjunto puede tener un número finito o infinito de elementos, pero no puede ser un conjunto vacío.

Ejemplos:

a) El conjunto de valores de género booleano es {verdadero, falso}.

b) El conjunto de valores de género natural es el conjunto infinito de números naturales {0, 1, 2, . . .}.

c) El conjunto de valores de género color_primario es {verde, rojo, azul}.

Cada elemento de un género no necesita ser proporcionado directamente por el usuario (tomaría una cantidad infinita de tiempo en el caso de los números naturales) pero hay que dar el nombre del género. En la sintaxis concreta la palabra clave NEWTYPE (NEOTIPO) es seguida directamente por el nombre del género (algunas otras posibilidades se examinarán más adelante). Este nombre se usa principalmente en definiciones de operador, como se explica en el § D.6.1.3 y en declaraciones de variable.

D.6.1.3 *Operadores, literales y términos*

Los valores de un género pueden definirse de tres maneras:

a) por enumeración; los valores se definen en la sección literales,

b) por operadores; los valores se producen como resultados de «aplicaciones» de operadores,

c) por una combinación de enumeración y operadores.

La combinación de literales y operadores da términos. Las relaciones entre términos se expresan utilizando ecuaciones. Las secciones siguientes se refieren a los literales, operadores y términos; el § D.6.1.4 trata de las ecuaciones.

D.6.1.3.1 *Literales*

Los literales son valores enumerados de un género. Una definición parcial de tipo no necesita contener literales; todos los elementos del género podrían definirse por medio de operadores. Los literales pueden considerarse como operadores sin argumentos. Una relación entre literales puede expresarse en ecuaciones. En la sintaxis concreta los literales se introducen después de la palabra clave LITERALS (LITERALES).

Ejemplos:

a) La definición del género booleano contiene dos literales, a saber, verdadero (true) y falso (false). De modo que la definición del tipo booleano tiene la forma:

```
NEWTYPE Booleano
    LITERALS Verdadero, Falso
    …
ENDNEWTYPE Booleano;
```

b) El género natural puede definirse utilizando un literal, el cero. Los otros valores pueden generarse utilizando este literal y operadores.

c) Los valores del género color_primario pueden definirse en forma similar a los literales booleanos:

```
NEWTYPE Color_primario
    LITERALS Rojo, Verde, Azul
    …
ENDNEWTYPE Color_primario;
```

d) En el § D.6.1.3.2 el tercer ejemplo c) muestra una definición parcial de tipo sin literales.

D.6.1.3.2 *Operadores*

Un operador es una función matemática que hace corresponder uno o más valores (posiblemente de géneros diferentes) con un valor resultado. Los operadores se introducen después de la palabra clave OPERATORS (OPERADORES), el/los género(s) de su(s) argumento(s) y el género del resultado se indican también (es o se llama la signatura de los operadores).

Ejemplos:

a) En el tipo booleano puede definirse un operador llamado «not» («no»), que tiene un argumento de género booleano y un resultado también de género booleano. En la definición del tipo booleano, aparece como sigue:

```
NEWTYPE Booleano
    LITERALS Verdadero, Falso
    OPERATORS «Not»: Booleano->Booleano;
    …
ENDNEWTYPE Booleano;
```

b) El operador mencionado en la sección precedente que se necesita para construir todos los números naturales es Next (Siguiente). Este operador toma un argumento de género natural y da un valor natural (el valor superior siguiente) como resultado.

c) Puede definirse un nuevo tipo para colores que no tiene literales pero usa literales de género color_primario y algunos operadores:

```
NEWTYPE Color
    OPERATORS
      Tomar: Color_primario->Color;
      Mezclar: Color_primario, Color->Color;
    …
ENDNEWTYPE Color;
```

La intención es tomar un primario y considerarlo como un color, y después comenzar a mezclar con éste más colores primarios para obtener más colores.

D.6.1.3.3 *Términos*

Utilizando literales y operadores puede construirse el conjunto de términos para cada género, como sigue:

1) Reunir todos los literales en un conjunto del género en el que están definidos – cada literal es un término.

2) Para cada operador se crea un nuevo conjunto de términos donde el operador es aplicado a todas las posibles combinaciones de términos de los conjuntos previamente creados del género correcto.

   a) Para el género booleano el conjunto de literales es {True, False}. El resultado de este paso es {Not(True), Not(False)} porque sólo tenemos el operador Not.

   b) Para el género natural el resultado de este paso es {Next(0)}.

   c) Para el género color el conjunto de literales es un conjunto vacío, pero el resultado de este paso es {tomar(rojo), tomar(verde), tomar(azul)}.

3) Los términos de los conjuntos creados en el paso anterior son todos del género del resultado del operador aplicado, por ejemplo, todos los resultados del operador Not son del género booleano. Se toma ahora la unión de todos los conjuntos del mismo género, tanto los conjuntos originales como los creados posteriormente.

   a) Para el género booleano esta unión proporciona el conjunto {True, False, Not(True), Not(False)}.

   b) Para los números naturales este paso da el conjunto {0, Next(0)}.

4) Los dos últimos pasos se repiten una y otra vez, definiendo en general un conjunto infinito de términos.

   a) El conjunto de términos booleanos generados por los literales True y False y el operador Not es {True, False, Not(True), Not(False), Not(Not(True)), Not(Not(False)), Not(Not(Not(True))), . . .}.

   b) El conjunto de términos naturales generados por el literal 0 y el operador Siguiente (Next) es {0, Next(0), Next(Next(0)), Next(Next(Next(0))), . . .}.

   c) El conjunto de términos de color generado por los literales rojo, verde y azul del género color_primario y los operadores tomar y mezclar es {tomar(rojo), tomar(verde), tomar(azul), mezclar(rojo, tomar(rojo)), mezclar(rojo, tomar(verde)), mezclar(rojo, tomar(azul)), mezclar(verde, tomar(rojo)), mezclar(verde, tomar(verde)), mezclar(verde, tomar(azul)), mezclar(azul, tomar(rojo)), mezclar(azul, tomar(verde)), mezclar(azul, tomar(azul)), . . .}.

D.6.1.4 *Ecuaciones y axiomas*

En general, el número de términos generados como se indica en la sección precedente es mayor que el número de valores del género. Por ejemplo, hay dos valores booleanos, pero el conjunto de términos booleanos tiene un número infinito de elementos. Existe, sin embargo, una posibilidad de proporcionar reglas que establezcan qué términos se considera que indican el mismo valor. Estas reglas se llaman ecuaciones, y se explicarán en la subsección siguiente. En los § D.6.1.4.2 y D.6.1.4.3 se tratan dos clases especiales de ecuaciones, los axiomas y las ecuaciones condicionales.

Los axiomas, ecuaciones y ecuaciones condicionales se dan todos, en la sintaxis concreta, después de la palabra clave AXIOMS. Esta palabra clave se mantiene por razones históricas.

D.6.1.4.1 *Ecuaciones*

Una ecuación establece cuáles términos se considera que denotan el mismo valor. Una ecuación relaciona dos términos que están separados por el símbolo de equivalencia = =.

Por ejemplo, «Not(True) = = False» establece que los términos Not(True) y False son equivalentes; en todo sitio donde aparezca Not(True), False puede sustituirlo sin cambiar el significado y viceversa.

Al dar ecuaciones el conjunto de términos se divide en subconjuntos disjuntos de términos que denotan el mismo valor. Estos subconjuntos se llaman clases de equivalencia. En el hablar cotidiano, las clases de equivalencia se identifican con los valores.

Ejemplos:

a) El conjunto de términos de género booleano está dividido en dos clases de equivalencia de términos por los dos axiomas siguientes:

   Not(True) = = False;

   Not(False) = = True;

   Las clases de equivalencia resultantes son:

   {True, Not(False), Not(Not(True)), Not(Not(Not(False))),

   Not(Not(Not(Not(True)))), Not(Not(Not(Not(Not(False))))), . . .}

   y

   {False, Not(True), Not(Not(False)), Not(Not(Not(True))),

   Not(Not(Not(Not(False)))), Not(Not(Not(Not(Not(True))))),} . . .

   Estas dos clases de equivalencia se identifican con los valores True y False.

b) En el caso de los colores se quiere especificar que da lo mismo mezclar un color primario con un color que contiene el primario. Además, el orden en que los colores primarios se mezclan no tiene importancia. Esto puede establecerse en ecuaciones, como:

```
mezclar(rojo, tomar(rojo))                    = =  tomar(rojo);
mezclar(rojo, mezclar(rojo, tomar(verde)))    = =  mezclar(rojo, tomar(verde));
mezclar(rojo, mezclar(rojo, tomar(azul)))     = =  mezclar(rojo, tomar(azul));
mezclar(rojo, mezclar(verde, tomar(rojo)))    = =  mezclar(verde, tomar(rojo));
mezclar(rojo, mezclar(azul, tomar(rojo)))     = =  mezclar(azul, tomar(rojo)); etc.
```

Esto representa un trabajo excesivo porque, para todas las permutaciones de rojo, verde y azul, aparecen ecuaciones similares. Por lo tanto, el LED tiene la construcción FOR ALL que introduce nombres de valores que representan una clase de equivalencia arbitraria (o el valor asociado con esta clase de equivalencia). En la situación arriba indicada esto puede ser muy provechoso; todas las ecuaciones que se establecieron anteriormente y aquéllas que están indicadas por etc. pueden escribirse en unas pocas líneas, como sigue:

```
FOR ALL p1, p2 IN color_primario
/*1*/    (  mezclar(p1, tomar(p1)) = = tomar(p1);
/*2*/    mezclar(p1, mezclar(p1, tomar(p2))) = = mezclar(p1, tomar(p2));
/*3*/    mezclar(p1, mezclar(p2, tomar(p1))) = = mezclar(p2, tomar(p1));
/*4*/    mezclar(p1, tomar(p2)) = = mezclar(p2, tomar(p1));
         FOR ALL c IN Color
/*5*/    (  mezclar(p1, mezclar(p2, c)) = = mezclar(p2, mezclar(p1, c));
/*6*/    mezclar(p1, mezclar(p2, c)) = = mezclar(mezclar (p1, tomar(p2)), c))
         )
```

En estas ecuaciones hay una superposición pero esto no es un problema mientras las ecuaciones no se contradigan entre sí.

Las ecuaciones antes indicadas crean 7 clases de equivalencia en el conjunto de términos del género color, de modo que con estas ecuaciones hay 7 valores de color. Los términos siguientes están en clases de equivalencia diferentes:

```
tomar(rojo), tomar(verde), tomar(azul),
mezclar(rojo, tomar(verde)),
mezclar(verde, tomar(azul)),
mezclar(azul, tomar(rojo)),
mezclar(azul, mezclar(verde, tomar(rojo))).
```

Todos los otros términos del género color son equivalentes a uno de los términos que se acaban de indicar.

En los ejemplos de ecuaciones con la construcción FOR ALL, llamadas ecuaciones cuantificadas explícitamente, la información que p1 y p2 son identificadores de valor del género color_primario es redundante; el argumento del operador tomar y el primer argumento del operador mezclar pueden ser solamente del género color_primario. En general, cuando las ecuaciones están cuantificadas explícitamente, esto las hace de más fácil lectura, pero se permite omitir la cuantificación si el género de los identificadores de valor puede deducirse del contexto. En ese caso, se dice que la ecuación está cuantificada implícitamente.

Ejemplos:

Las ecuaciones 4 y 5 son las mismas que:

```
mezclar(p1, tomar(p2)) , c))      = =  mezclar(p2, tomar(p1));
mezclar(p1, mezclar(p2, c))       = =  mezclar(p2, mezclar(p1, c));
```

D.6.1.4.2 *Axiomas*

Los axiomas no son más que una clase especial de ecuaciones, introducida porque muchas de las ecuaciones en situaciones prácticas se refieren a booleanos. En ese caso las ecuaciones tienden a ser de la forma . . . = = True; es decir, establecen que algún término es equivalente a True.

Ejemplo:

Supongamos que para el tipo color se define un operador: contiene: color, color_primario-> booleano destinado a dar True si el primario está contenido en el color y False de no ser así. Algunas de las ecuaciones incluidas son:

```
FOR ALL p IN Color_primario
( Contiene(Tomar(p),p)        = =  True;
FOR ALL c IN Color
( Contiene(Mezclar(p,c), p)   = =  True)
)
```

La parte ′= = True′ de estas ecuaciones puede omitirse, y los resultados se llaman axiomas. Los axiomas pueden reconocerse por la ausencia del símbolo de equivalencia = =, e indican términos que son equivalentes al valor True del género booleano.

La construcción de la segunda ecuación puede parecer un poco forzada. Una mejor manera de escribir estas ecuaciones se muestra después de la introducción de algunas construcciones auxiliares.

D.6.1.4.3 *Ecuaciones condicionales*

Las ecuaciones condicionales son un medio de escribir ecuaciones que se cumplen solamente si algunas condiciones se cumplen. Las condiciones se indican con la misma sintaxis que las ecuaciones incondicionales y están separadas por un símbolo = => de la ecuación que se cumple si la condición se cumple.

Ejemplo:

El ejemplo típico de una ecuación condicional es la definición de división en el tipo real donde:

```
FOR ALL x, z IN Real
(   z/= 0 = = True          = =>        (x/z) * z = = x)
```

establece que si se cumple la condición «z no es igual a 0», la división por z seguida por la multiplicación por z da el valor original. Esta ecuación condicional no establece nada sobre lo que sucedería si un valor de género real se divide por 0. Si se quiere especificar lo que sucede en caso de división por 0 debe darse una ecuación condicional de la forma:

```
FOR ALL x, z IN Real
(   z = 0 = = True          = =>        (x/z) * z = = . . .).
```

Sin embargo en estos casos se recomienda por razones de legibilidad un denominado «término condicional». En el caso anterior la ecuación sería:

```
FOR ALL x, z IN Real
(   (x/z) * z          = =      IF z/=0
                                THEN x
                                ELSE . . .
                                FI
)
```

D.6.1.5 *Más sobre ecuaciones y axiomas*

Las dos secciones siguientes explican algunas dificultades que se pueden encontrar cuando los operadores contienen resultados de un género ya definido. El § D.6.1.5.3 explica el concepto de error como término de una ecuación.

D.6.1.5.1 *Coherencia jerárquica*

En todo punto de una especificación LED hay una definición de tipo de datos y solamente una. Esta definición de tipo de datos contiene los géneros, operadores y ecuaciones predefinidos y todos los géneros, operadores y ecuaciones definidos por el usuario en las definiciones parciales de tipo visibles en ese punto. (Esta es la razón por la que un texto NEWTYPE . . . ENDNEWTYPE se llama definición *parcial* de tipo.)

Esto tiene algunas consecuencias para las definiciones de tipo en niveles inferiores. Ellas pueden influenciar el tipo de una manera indeseada. Por ejemplo, se podría especificar erróneamente que dos términos son equivalentes, haciéndolos así equivalentes cuando no son equivalentes en un ámbito circundante.

No se permite dar ecuaciones que resulten en que:

a) se hagan equivalentes valores de un género que son diferentes en un ámbito de nivel superior;

b) se agreguen nuevos valores a un género definido en un ámbito de nivel superior.

Esto significa, por ejemplo, que en un bloque a nivel de sistema las definiciones parciales de tipo especificadas por el usuario que contienen un operador con un género de resultado predefinido deben relacionar todos los términos producidos por este operador con valores del género de este resultado.

Ejemplos:

a) Si, por algún motivo, se da el axioma:

```
FOR ALL n, m IN Integer
(  (Fact(n) = Fact(m))      =>          (n = m))
```

con la intención de especificar que si los resultados del operador fact son los mismos, los argumentos son los mismos (obsérvese que => es la implicación booleana; ésta tiene poco que ver con el signo de ecuación condicional = =>), entonces, por accidente, se unifican valores. De las ecuaciones del ejemplo anterior puede derivarse que fact(0) – fact (1), y esta última ecuación establece que 0 y 1 son diferentes denotaciones del mismo valor. De esta última ecuación puede probarse que el número de elementos del género entero (integer) se reduce a uno.

Con la ayuda de una ecuación condicional puede establecerse que, siempre que n y m no sean iguales a 0, el mismo resultado del operador fact sobre n y m implica que n = m. En LED:

```
FOR ALL n, m IN Integer
(n/=0, m/=0 = => (Fact(n) = Fact(m))          =>          (n = m))
```

Obsérvese que esta última ecuación no agrega nada a la semántica de los enteros; es un teorema que puede derivarse de otras ecuaciones. Por otro lado, la adición de una ecuación que puede probarse no afecta.

b) Supongamos que se descubre la necesidad de un operador para factoriales cuando se está especificando algún tipo. En la definición parcial de tipo de este tipo se introduce el operador fact:

```
Fact: Integer -> Integer;
```

y, para definir este operador, se dan las siguientes ecuaciones:

```
Fact(0) = = 1;
FOR ALL n IN Integer
( n > 0 = => Fact(n) = = n * Fact(n–1))
```

Estas ecuaciones no definen fact(–1) y, por lo tanto, es un término del género entero que no tiene relación con otros términos de este género. Por consiguiente, fact(–1) es un valor nuevo de género entero (y lo mismo se cumple para fact(–2), fact(–3), etc.). Esto no está permitido. El ejemplo b) del § D.6.1.5.3 muestra una definición correcta de fact.

D.6.1.5.2 *Igualdad y desigualdad*

Los operadores de igualdad y desigualdad son implícitos para cada tipo. Por lo tanto, si una definición parcial de tipo introduce el género S, existen las siguientes definiciones implícitas de operadores:

```
"=": S, S -> Booleano;

"/=": S, S -> Booleano;
```

(*Nota* – Las comillas especifican que = y /= se utilizan como operadores infijos.)

El operador de igualdad tiene las propiedades que se esperan:

a = a,

a = b => b = a,

a = b AND b = c => a = c,

a = b => a = = b,

a = b => op(a) = op(b) para todos los operadores op.

Las propiedades arriba indicadas no están escritas en sintaxis LED y no deben establecerse como axiomas o ecuaciones porque son implícitas. El valor booleano obtenido cuando se aplica este operador es True cuando los términos del lado izquierdo y los del lado derecho están en la misma clase de equivalencia; de no ser así, el valor obtenido es False. Si no se especifica explícitamente que el valor True es True o False, la especificación es incompleta.

Para el operador de desigualdad, la semántica se explica mejor mediante una ecuación LED:

```
FOR ALL a, b IN S
(   a/= b = = Not(a = b))
```

No hay diferencia entre igualdad y equivalencia. Dos términos que son equivalentes denotan el mismo valor y el operador de igualdad entre ellos da el resultado True.

D.6.1.5.3 *Error*

En los ejemplos anteriores se sintió la necesidad de especificar que la aplicación de operadores a algunos valores se considera como un error. El LED tiene un medio para especificar esto formalmente: el Error. El Error deberá utilizarse para expresar:

> «la aplicación de este operador a este valor no está autorizada y, cuando se le encuentra, el comportamiento futuro es indefinido».

En la sintaxis concreta esto se denota por el término Error!, que no puede utilizarse como argumento de un operador.

Cuando Error es el resultado de una aplicación de operador, y esta aplicación es un argumento de otro operador, la aplicación del operador exterior tiene como resultado también Error (propagación del error). En un término condicional se evalúa la parte THEN *o* la parte ELSE, así que una de ellas puede ser Error sin que Error sea evaluado (cuando se evalúa la otra alternativa).

Ejemplos:

a) En el ejemplo de la división de valores de género real los puntos suspensivos pueden llenarse:

```
FOR ALL x, z IN Real
(  (x/z) * z      = =    IF z/=0
                         THEN      x
                         ELSE      Error!
                         FI
)
```

Para mayor claridad, se puede agregar:

```
FOR ALL x IN Real
(  x/0 = = Error!)
```

b) En el ejemplo del operador fact se puede especificar que la aplicación de este operador sobre enteros negativos se considera como un Error. Esto evita que fact(–1), fact(-2), . . . se conviertan en valores del género entero (integer). Una buena definición del operador fact, sería:

```
n < 0       = =>    Fact(n)  = =  Error!;
                    Fact(0)  = =  1;
n > 0       = =>    Fact(n)  = =  Fact(n-1) * n;
```

Estas tres líneas son mucho más claras que el estilo de programación de la ecuación de más abajo. En general, debería utilizarse el término condicional si hay dos casos complementarios; el anidado de términos condicionales arruina la legibilidad, como se puede ver de:

```
Fact(n)     = =     IF n > 0
                    THEN Fact(n – 1) * n
                    ELSE    IF n  =  0
                            THEN 1
                            ELSE Error!
                            FI
                    FI
```

D.6.2 *Generadores y herencia*

Esta sección se refiere a dos construcciones que pueden usarse para especificar tipos que tienen partes comunes. El generador especifica, no un tipo, sino un esquema que se convierte en un tipo cuando se reemplazan los géneros, operadores, literales y constantes formales por los efectivos.

La herencia ofrece la posibilidad de hacer un tipo nuevo partiendo de un tipo ya existente. Se puede asignar nuevos nombres a los literales y operadores y se puede especificar literales, operadores y ecuaciones adicionales.

D.6.2.1 *Generadores*

Una definición de generador define un esquema cuyos parámetros son nombres formales de géneros, literales, constantes y operadores. Los generadores se destinan a tipos que son ′variaciones sobre un tema′, como conjuntos de elementos, cadenas de elementos, ficheros de registros, tablas de búsqueda, matrices.

Esto se explicará utilizando un ejemplo para el cual se pueden imaginar variaciones. Supongamos que se necesita un tipo que se asemeje a la noción matemática de un conjunto de enteros. El texto que sigue es parte de la definición de tipo de este conjunto de enteros.

```
NEWTYPE Int_set
   LITERALS empty_int_set
   OPERATORS
      Add : Int_set, Integer -> Int_set;
      Delete : Int_set, Integer -> Int_set;
      Is_in : Int_set, Integer -> Boolean
   AXIOMS
   /* 1 */    Delete(empty_int_set, int)
                    ==  empty_int_set;
   /* 2 */    Is_in(set,int1) = false ==>
              Delete(Add(set, int1), int2)
                    ==  IF int1 = int2
                        THEN  set
                        ELSE  Add(Delete(set, int2), int1)
                        FI;
   /* 3 */    Is_in(empty_int_set, int)
                    ==  False;
   /* 4 */    Is_in(Add(set, int1), int2)
                    ==  int1 = int2 OR Is_in(set, int2);
   /* 5 */    Add(Add(set, int1), int2)
                    ==  IF int1 = int2
                        THEN  Add(set, int1)
                        ELSE  Add(Add(set, int2), int1)
                        FI
ENDNEWTYPE Int_set;
```

FIGURA D-6.2.1

Neotipo Int_set

Todas las ecuaciones tienen cuantificación implícita. La primera ecuación establece que la supresión de un elemento del conjunto vacío da como resultado el conjunto vacío. La segunda ecuación dice que la supresión después de la inserción del mismo elemento da como resultado el conjunto de antes de la inserción (siempre que el conjunto no contenga el elemento), de otro modo, el orden inserción y supresión puede intercambiarse. La tercera ecuación establece que el conjunto vacío no contiene ningún elemento. La cuarta ecuación dice que un elemento está en un conjunto si es el último elemento agregado o si estaba en el conjunto antes que se agregase el último elemento. La última ecuación establece que no importa el orden en que se agregan elementos al conjunto.

En el ejemplo de la figura D-6.2.1 Int_set es sólo un ejemplo de un conjunto, y si se necesita también un PId_set, un Subscriber_set y un Exchange_name_set en la misma especificación no es de sorprenderse que todos contendrán los operadores Add, Delete e Is_in y un literal para el conjunto vacío. Las ecuaciones dadas para estos operadores se generalizan fácilmente para otros conjuntos.

Aquí es donde el concepto de generador prueba su utilidad; el texto común puede darse una vez y puede utilizarse varias veces. La figura D-6.2.2 muestra el generador. (Obsérvese que los nombres formales de género se introducen con la palabra clave TYPE. Esto sólo por razones históricas.)

En vez de usar Integer, se utiliza el tipo formal Item, y para que sea posible dar diferentes nombres al conjunto entero vacío y a los conjuntos vacíos de otros tipos este literal se hace también un parámetro formal.

```
GENERATOR Set (TYPE Item, LITERAL empty_set)
   LITERALS empty_set
   OPERATORS
      Add : Set, Item -> Set;
      Delete : Set, Item -> Set;
      Is_in : Set, Item -> Boolean
   AXIOMS
   /* 1 */    Delete(empty_set, itm)
                  ==  empty_set;
   /* 2 */    Is_in(st,itm1) = false ==>
              Delete(Add(st, itm1), itm2)
                  ==  IF itm1 = itm2
                      THEN  st
                      ELSE  Add(Delete(st, itm2), itm1)
                      FI;
   /* 3 */    Is_in(empty_set, itm)
                  ==  False;
   /* 4 */    Is_in(Add(st, itm1), itm2)
                  ==  itm1 = itm2 OR Is_in(st, itm2);
   /* 5 */    Add(Add(st, itm1), itm2)
                  ==  IF itm1 = itm2
                      THEN  Add(st, itm1)
                      ELSE  Add(Add(st, itm2), itm1)
                      FI
ENDGENERATOR Set;
```

FIGURA D-6.2.2

Conjunto Generador

Con este generador el tipo Int_set puede construirse como sigue:

NEWTYPE Int_set Set (Integer, empty_int_set)

ENDNEWTYPE Int_set;

Si comparamos la figura D-6.2.1 y la figura D-6.2.2 podemos ver que:

a) se ha reemplazado GENERATOR y ENDGENERATOR por NEWTYPE y ENDNEWTYPE respectivamente,

b) los parámetros formales del generador (es decir, el texto entre paréntesis después del nombre del generador) han sido suprimidos,

c) Set, Item y empty_set se reemplazan en todo el generador por Int_set, Integer, y empty_int_set, respectivamente.

Así, no hay absolutamente ninguna diferencia entre este Int_set y el de la figura D-6.2.1, pero . . .

– si se necesita un conjunto de valores PId el tipo puede crearse mediante:

WTYPE PId-set Set(PId, empty_pid_set)

DNEWTYPE PId_set;

– si se necesita un conjunto de abonados, donde los abonados estén representados por un tipo que introduce el género Subscr, el conjunto de abonados puede crearse mediante:

```
NEWTYPE Subscr_set Set(Subscr, empty_subscr_set)
ENDNEWTYPE Subscr_set;
```

Esto no solamente economiza papel, sino que también facilita las cosas debido a que sólo se tiene que pensar una vez en los conjuntos o puede delegarse este trabajo a especialistas expertos en tipos de datos abstractos.

Ejemplo:

Este ejemplo muestra un generador que usa género, operador, literal y constante formales. Describe una fila (row) de elementos con una longitud máxima max_length. El género tiene un literal para denotar la fila vacía y operadores para inserción y supresión de elementos en/desde una fila, concatenación de filas, selección de una subfila y determinación de la longitud de una fila. Este último operador se hace formal para poder darle otros nombres.

```
GENERATOR Row (TYPE Item, OPERATOR Length, LITERAL Empty,
                                CONSTANT max_length)
    LITERALS Empty
    OPERATORS
        Length: Row                 -> Integer;
        Insert: Row, Item, Integer  -> Row;
        Delete: Row, Integer, Integer -> Row;
        "//": Row, Row              -> Row;
        Select: Row, Integer, Integer -> Row
        /* y otros operadores pertinentes para filas de ítems */
    AXIOMS
        /* Las ecuaciones de los operadores arriba mencionados, entre *
        /* ellos los dos siguientes (o sus equivalentes)              */
        Length(r) = max_length = => Insert(r, itm, int)  = =  Error!;
        Length(r1) + Length(r2) > max_length = => r1//r2  = =  Error!
    ENDGENERATOR Row;
```

Obsérvese que el operador formal Length y el literal Empty se dan de nuevo en el cuerpo del generador porque se les da otro nombre cuando son instanciados. En el caso del operador, los argumentos y el género del resultado se dan en el cuerpo solamente. El generador Row puede utilizarse para hacer líneas, páginas y libros como sigue:

```
NEWTYPE Line Row(Character, Width, Empty_line, 80)
ENDNEWTYPE Line;
NEWTYPE Page Row(Line, Length, Empty_page, 66)
ENDNEWTYPE Page;
NEWTYPE Book Row(Page, Nrf_pages, Empty_book, 10000)
ENDNEWTYPE Book;
```

D.6.2.2 *Herencia*

La herencia es una manera de obtener todos los valores del llamado género progenitor (o ascendiente), algunos o todos los operadores del tipo progenitor y todas las ecuaciones del tipo progenitor. Existe la posibilidad de poner nuevos nombres a los literales y operadores. Esta es en general una buena práctica porque en ese caso el lector puede deducir del contexto que hay otro tipo implicado aun si los literales son los mismos.

Si un operador no es legado, se le cambia sistemáticamente el nombre por uno que no es accesible al usuario. El hecho de que los operadores estén todavía presentes significa que todas las ecuaciones del tipo progenitor están presentes todavía (con operadores denominados de nuevo). Esto asegura que los valores de progenitor sean heredados.

Junto con la posibilidad de impedir el uso de un operador (no legándolo), se suministra la posibilidad de añadir nuevos operadores. Después de la palabra clave ADDING, pueden darse literales, operadores y ecuaciones como en un tipo ordinario. Sin embargo, se debe tener especial cuidado con los literales nuevos y la interferencia entre operadores heredados y añadidos.

Cuando se añaden literales, el resultado de los operadores heredados cuando se aplican a estos nuevos literales debe definirse (mediante ecuaciones). Cuando se añaden operadores se debe recordar los operadores denominados de nuevo en forma invisible y sus ecuaciones asociadas. Las ecuaciones para definición de los operadores añadidos deberán ser consistentes con las ecuaciones que implican operadores heredados y con las que implican operadores no heredados.

Después de esta lista de advertencias, veamos algunos ejemplos.

a) Supongamos que el neotipo color está completo y disponible. Este tipo está basado en tomar y mezclar rayos de luz de colores primarios. Definir algo similar para tomar y mezclar pintura significaría perder mucho tiempo en pensar y escribir y/o copiar.

   Una buena solución a este problema es transformar el neotipo color en un generador con sólo dos reemplazos:

   1) la primera línea

      NEWTYPE color

      se convierte en

      GENERATOR color (TYPE color_primario)

   2) la palabra clave ENDNEWTYPE se convierte en ENDGENERATOR.

   Ahora se puede poner otro nombre al generador cuando se le instancie. Supongamos que el género anterior color_primario se llame primario_luz, y que el género primario_pintura se defina como:

```
NEWTYPE primario_pintura
    LITERALS rojo, amarillo, azul
ENDNEWTYPE primario_pintura;
```

   Es ahora muy fácil definir dos tipos similares, uno para la luz y uno para la pintura:

```
NEWTYPE colores_luz color (primario_luz) ENDNEWTYPE;
NEWTYPE colores_pintura color (primario_pintura) ENDNEWTYPE;
```

Hasta aquí no hay problema pero ¿cómo se puede ver la diferencia entre colores_luz tomar (rojo) y colores_pintura con la misma sintaxis? Si se siente la necesidad de distinguir entre los dos, la herencia puede ayudar. En vez de colores_luz y colores_pintura, se definen por medio de la herencia los tipos luz y Paleta dando otro nombre al operador tomar:

```
NEWTYPE luz
    INHERITS colores_luz
    OPERATORS (haz=tomar, mezclar, contiene)
    ADDING
        LITERALS blanco
        AXIOMS
          blanco = = mezclar(rojo, mezclar(amarillo, haz(azul)))
ENDNEWTYPE luz;
```

El neotipo luz tiene ahora los literales de colores_luz más el literal blanco. Colores_luz no tiene literales propios (porque utiliza los literales de primario_luz), de modo que blanco es el único literal de luz. Los operadores y ecuaciones de luz son los mismos que los de colores_luz excepto que el nombre de operador tomar se reemplaza por haz, y se añade la ecuación dada para blanco. El axioma añadido establece que el literal añadido se convierte en un elemento del conjunto de términos donde los tres primarios están mezclados.

El neotipo Paleta tiene los literales de colores_pintura y el operador tomar se reemplaza por pintar:

```
NEWTYPE Paleta
    INHERITS colores_pintura
    OPERATORS (pintar = tomar, mezclar, contiene)
ENDNEWTYPE Paleta;
```

b) Supongamos que se quiere ampliar el tipo «conjunto de enteros» (género Int_set), introducido en la sección anterior, por medio de un operador que encuentre el entero más pequeño del conjunto. En primer lugar, debería uno preguntarse si este operador puede introducirse en la definición de generador para hacerlo asequible también para conjuntos de otras cosas.

Aunque esto puede hacerse, restringiría Item a tener definidos > y < . Esto no se ajusta a todos los ítems (por ejemplo a PId) y puede ser mejor hacer un neotipo de género New_int_set que suministre un operador Min.

```
NEWTYPE New_int_set
    INHERITS Int_set
    OPERATORS ALL
    ADDING
        OPERATORS
            Min: New_int_set-> Integer
        AXIOMS
            Min(Empty_int_set) = = Error!;
            Min(Add(Empty_int_set, x))= =x;
            Min(Add(Add(nis,x),y))= =
                IF y <Min(Add(nis,x))
                THEN y
                ELSE Min(Add(nis,x))
                FI
ENDNEWTYPE New_int_set;
```

Debido a que es bastante común una adición después de una instanciación de generador, el texto que comienza con ADDING y termina inmediatamente antes de ENDNEWTYPE puede darse dentro de la instanciación del generador. Se da un ejemplo en el § 5.4.1.12 de la Recomendación.

D.6.3 *Consideraciones sobre las ecuaciones*

Cuando se introduce un nuevo tipo de datos, es esencial introducir un número suficiente de ecuaciones. A continuación se dan tres consideraciones sobre las ecuaciones, que ayudarán a construirlas.

D.6.3.1 *Requisitos generales*

De cualquier modo que se construyan las ecuaciones, debe cumplirse lo siguiente:

a) cada operador aparece por lo menos una vez en el conjunto de ecuaciones (excepto en casos patológicos);

b) todas las sentencias verdaderas pueden derivarse de las ecuaciones. O se establecen como axiomas o pueden derivarse sustituyendo términos equivalentes en las ecuaciones;

c) no se puede detectar inconsistencias; por ejemplo, no puede derivarse de las ecuaciones que Verdadero=Falso.

Se puede expresar en LED informal un procedimiento para encontrar ecuaciones, como en la figura D-6.3.1.

D.6.3.2 *Aplicación de funciones en constructores*

En general, el conjunto de operadores tiene un subconjunto de operadores conocidos como «constructores» y «funciones». Los constructores pueden utilizarse para generar todos los valores (clases de equivalencia) del género. En este método los literales se consideran como operadores sin argumentos.

Ejemplos:

a) El tipo booleano tiene sus literales como constructores.

b) El tipo natural tiene el literal 0 y el operador Next(Siguiente) como sus constructores; todo natural puede construirse con 0 y Next solamente.

c) El generador de conjuntos tiene el literal empty_set (conjunto_vacío) y el operador_add como sus constructores; todo conjunto puede construirse usando solamente empty_set y add.

d) El tipo entero puede construirse por medio de los literales 0 y 1, el operador + y el menos unario.

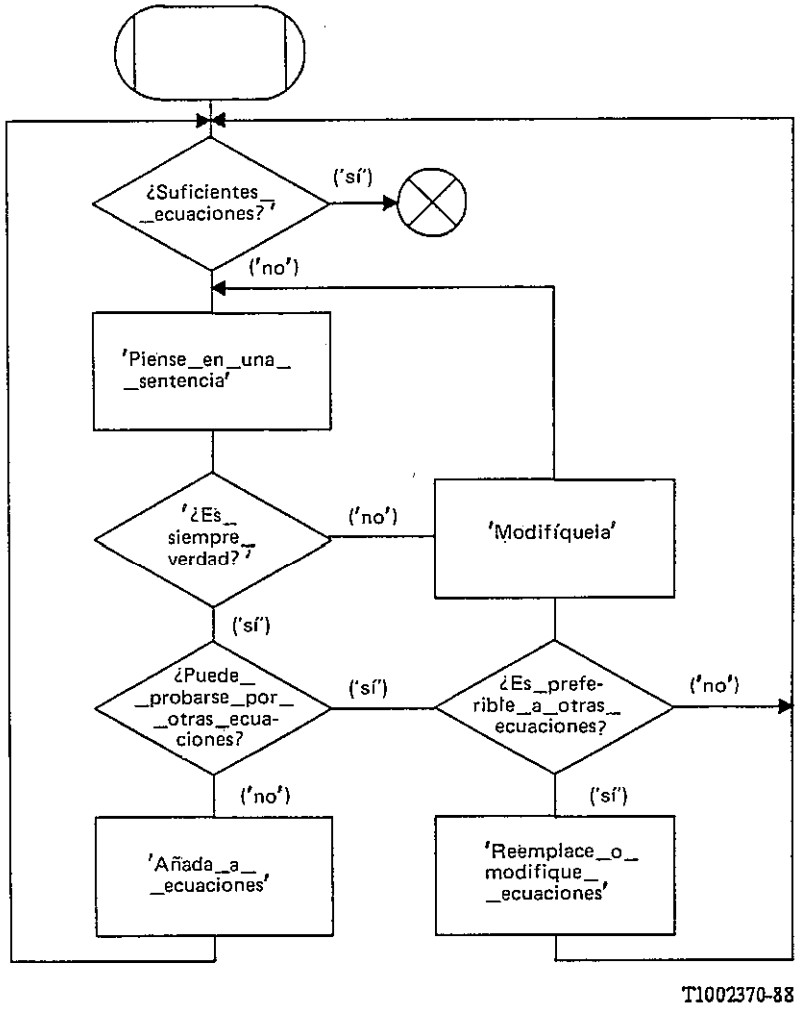


FIGURA D-6.3.1

Procedimiento para encontrar ecuaciones en LED informal

Debe observarse que hay a veces varias opciones posibles para el conjunto de constructores. Cualquier opción se aceptará en el resto de esta sección, pero los conjuntos pequeños son en general los mejores.

Todas las funciones se tratan ahora una por una. Para cada argumento de una función se enumeran todos los posibles términos que consisten en constructores solamente. Para evitar problemas con números infinitos de términos debe utilizarse la cuantificación.

Ejemplos:

a) Para los números naturales la lista puede reducirse a:

0

Next (n) donde n es cualquier número natural.

b) Una posible lista para los conjuntos es

empty_set

Add (s,i) donde s es cualquier conjunto e i es cualquier ítem.

Si en el término del lado derecho de una ecuación que tiene (s,i) en el lado izquierdo hay una diferencia entre que s sea vacío o no vacío, la lista se puede reescribir como sigue:

empty_set

Add (empty_set,i) donde i es cualquier ítem

Add (Add (s,i),j) donde s es cualquier conjunto e i, j son ítems cualesquiera.

Después de la creación de esta lista, los lados izquierdos de las ecuaciones se obtienen aplicando cada función a cualquier combinación de argumentos de la lista. A los identificadores de valor en los diferentes argumentos se les da nombres diferentes. El mismo procedimiento indicado anteriormente para funciones puede seguirse para los constructores; en ese caso da relaciones entre términos donde los constructores se utilizan en órdenes diferentes.

Ejemplos:

a) Para el operador de multiplicación de números naturales con signatura

«*»:Natural,Natural->Natural

este procedimiento da el lado izquierdo de las ecuaciones siguientes (incompletas). El usuario deberá suministrar el lado derecho.

0*0 = = . . .;

0*Next (n) = = . . .;

Next (n)0* = = . . .;

Next (n)*Next (m) = = . . .;

b) Para los operadores Is_in y Delete, en el generador Set (§ D.6.2.1), este método ya ha sido usado.

c) Para el género color los constructores son tomar y mezclar. Deberá definirse un operador similar a contiene (§ D.6.1.4.2) para los argumentos

tomar (p) donde p es cualquier primario,

mezclar (p,c) donde p es cualquier primario y c es cualquier color.

Como se prometió en el § D.6.1.4.2 dar ecuaciones claras para este operador, se dan completas:

contiene (tomar (p),q) = =p=q;

contiene (mezclar (p,c),q) = =(p=q) OR contiene (c,q);

Este procedimiento para construcción de ecuaciones puede producir más ecuaciones que las necesarias, pero es muy seguro. Por ejemplo, en el ejemplo de la multiplicación de números naturales arriba indicado, es posible que se haya establecido la propiedad conmutativa de la multiplicación y por lo tanto solamente será necesaria la última (o la segunda) ecuación de las tres primeras.

El procedimiento descrito en esta sección puede utilizarse en combinación con el procedimiento descrito en la sección anterior donde puede ser útil en la tarea «Piense_en_una_sentencia».

D.6.3.3 *Especificación del conjunto de verificación*

Las ecuaciones pueden considerarse también desde el punto de vista de la implementación. Si los operadores se implementan como funciones de un lenguaje de programación, las ecuaciones describen cómo tienen que ser verificadas estas funciones.

Se evalúa la expresión correspondiente al lado izquierdo de una ecuación, se hace lo mismo para el lado derecho de esa ecuación y se ve si son equivalentes. Lo que puede causar problemas es la construcción FOR ALL. Un método muy pragmático resuelve a menudo el problema:

En vez de or puede usar FOR ALL i IN Integer

el verificador puede usar FOR ALL i IN {–10, –1,0,1,10} y esto bastará en la mayoría de los casos.

Pensar en ecuaciones como requisitos para la implementación puede ser útil en la tarea «Piense_en_una_sentencia» que figura en el procedimiento del § D.6.3.1.

D.6.4 *Características*

Esta sección describe algunas de las propiedades accesorias del LED, es decir, características que se necesitan muy rara vez o de las cuales se puede prescindir casi siempre, pero que a veces facilitan mucho las cosas.

D.6.4.1 *Operadores ocultos*

El conjunto de ecuaciones puede a veces simplificarse o hacerse de más fácil lectura si se introduce un operador adicional, pero este operador no debe utilizarse en el proceso. Esto significa que el operador es visible dentro de la definición de tipo, pero oculto fuera de ella.

Este resultado puede alcanzarse definiendo un «tipo oculto», es decir, un tipo que el usuario no debe utilizar. El usuario puede heredar de este tipo oculto todos los operadores a que se le permita tener acceso; es el tipo heredado el que deberá utilizarse. La inspección de todas las declaraciones de variable permite verificar el uso correcto (no deberá aparecer ninguna variable del género introducido por el tipo oculto).

La característica de los operadores ocultos supone que lo mismo puede conseguirse restringiendo la visibilidad de esos operadores solamente a las ecuaciones. Esto se hace colocando un signo de exclamación después del operador.

Ejemplo:

El modo correcto de hacer un conjunto de un elemento en el generador set(conjunto) es

```
Add(empty_set,x)
```

y esta es la manera en que cada usuario debería hacerlo. El especificador puede utilizar un operador especial en las ecuaciones, por ejemplo:

```
Mk_set!:Item->Set;
```

definido por la ecuación:

```
Mk_set!(itm) = = Add(empty_set,itm);
```

que puede utilizarse en definiciones parciales de tipo, pero no en el cuerpo de proceso LED.

D.6.4.2 *Ordenamiento*

Cuando se tiene que especificar ordenamiento de elementos de un género esto significa, en general, que deben definirse cuatro operadores (<, < =, >, > =), y sus propiedades matemáticas típicas (transitividad, etc.). Si los literales son muchos, deben darse también muchas ecuaciones. Por ejemplo, de esta manera se define el tipo de datos predefinidos carácter.

El LED suministra una característica que supera esas definiciones de tipo largas, difíciles de leer y aburridas: la notación taquigráfica ORDERING.

ORDERING se da en la lista de operadores, de preferencia al principio o al final de esta lista. Esto introduce los operadores de ordenamiento y las ecuaciones estándar. Cuando se especifica ORDERING los literales, de haberlos, tienen que darse en orden ascendente.

Ejemplo:

```
NEWTYPE Cifra_decimal_par
    LITERALS 0,2,4,6,8
    OPERATORS
        ORDERING
ENDNEWTYPE Cifra_decimal_par;
```

El ordenamiento 0 < 2 < 4 < 6 < 8 está ahora implícito.

En el § D.6.2.2 (Herencia) se advierte que hay que ser cuidadosos si se añaden literales a un género heredado. Este es un lugar excelente para mostrar por qué.

Supongamos que se quiere una ampliación del género Cifra_decimal_par como sigue:

```
NEWTYPE Cifra_decimal
    INHERITS Cifra_decimal_par
    OPERATORS ALL
    ADDING
        LITERALS 1,3,5,7,9
```

```
AXIOMS
  0 < 1; 1 < 2;
  2 < 3; 3 < 4;
  4 < 5; 5 < 6;
  6 < 7; 7 < 8;
  8 < 9
ENDNEWTYPE Cifra_decimal;
```

Los axiomas que aquí se dan no pueden omitirse. Sin estos axiomas hay solamente lo que se llama un ordenamiento parcial:

0 < 2 < 4 < 6 < 8

y

1 < 3 < 5 < 7 < 9.

Con los axiomas arriba indicados hay un ordenamiento completo:

0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9

pero con el axioma «9 < 0» en lugar del conjunto de axiomas indicados, el ordenamiento completo sería:

1 < 3 < 5 < 7 < 9 < 0 < 2 < 4 < 6 < 8.

D.6.4.3 *Género con campos*

Según se indica en el § 5.4.1.10 de la Recomendación, se puede definir un género estructurado sin construcciones especiales, pero como los géneros estructurados son tanto comunes como útiles, se justifica tener construcciones adicionales en el lenguaje.

Un género estructurado debe utilizarse cuando un valor objeto está formado por la asociación de valores de cierto número de géneros. Cada valor de esta asociación se caracteriza por un nombre, llamado el nombre de campo. El género de un campo es fijo.

Ejemplo:

```
NEWTYPE Subscriber
    STRUCT numbers Number_key;
           name Name_key;
           admin Administrative;
ENDNEWTYPE Subscriber;
NEWTYPE Name_key
    STRUCT name,
           street Charstring;
           number Integer;
           city Charstring;
ENDNEWTYPE Name_Key;
```

Algunos operadores están definidos implícitamente con el género estructurado:

a) el operador constructor "(." antes de los valores de campo y ".)" después de los mismos;

b) los operadores de selección del campo, la variable del género estructurado seguida por un ! y el nombre del campo, o seguida por el nombre del campo entre paréntesis. La variable seguida por un ! no debe confundirse con el operador oculto (§ D.6.4.1).

En la figura D-6.4.1 se da un ejemplo

```
PROCESS Some_process;
DCL  na, st, where  Charstring,
     nu             Integer,
     nk             Name_key;
  /* . . . Texto en que se asigna valores a las variables na y st */
  TASK nk := (. na, st, 5, 'London' .);
  /* . . . Texto en que no se asigna valores a nk */
  TASK where := nk!city;
  /* Ahora se cumple where = 'London' */
ENDPROCESS Some_process;
```

FIGURA D-6.4.1

Ejemplo de utilización de operadores implícitos en un género estructurado

D.6.4.4 *Géneros indizados*

Un género indizado es un género para el cual el tipo incluye extraer! (extract!) como nombre de operador. En los tipos de datos predefinidos, el generador matriz (array) es un tipo como ese. La matriz es uno de los ejemplos más comunes de un tipo indizado.

Para el operador oculto extraer! hay una sintaxis concreta especial que debe utilizarse fuera de las definiciones de tipo.

Podría pensarse que el tipo índice (index) en el generador predefinido matriz debería ser un tipo «sencillo», como entero, natural o carácter. Sin embargo, no hay razón para que una estructura como clave_nombre (name_key) no pueda ser utilizada como índice.

Ejemplo:

```
NEWTYPE Subsc_data_base
    Array (Name_key, Subscriber)
ENDNEWTYPE Subsc_data_base;
```

Los géneros name_key y subscriber son los definidos en la sección anterior. Supongamos que hay un procedimiento bill (facturar) con un parámetro de género subscriber, y que este procedimiento se define en un proceso que también tiene una variable sub_db de género subsc_data_base. En este proceso podría aparecer la llamada siguiente.

```
CALL Bill (Sub_db ((.'P.M. ', 'Downingstreet', 10, 'London'.)));
```

D.6.4.5 *Valores por defecto de variables*

Como ya se explicó en la sección sobre declaración de variables (§ D.3.10.1), es posible asignar valores directamente a una variable después de la declaración. Sin embargo, algunos tipos tienen un valor que (casi) siempre será el valor inicial de la variable. Hay una particularidad que evita tener que poner por escrito el valor inicial en cada declaración: la cláusula DEFAULT.

El conjunto se puede considerar como un ejemplo. Es muy posible que casi todas las variables, de cualquier conjunto en que se pueda pensar, se iniciarán con el conjunto_vacío (empty_set).

La notación:

```
DEFAULT empty_set
```

después de la lista de ecuaciones asegura que cada variable de cada instancia del generador se inicializará con el conjunto_vacío de esa instanciación, excepto si hay una inicialización explícita. (Véase § D.3.10.1.)

Si no es trivial que el valor inicial de todas las variables de un género deba ser el mismo, entonces no se usa la cláusula DEFAULT. De otro modo es difícil evitar sorpresas.

D.6.4.6 *Operadores activos*

Los usuarios familiarizados con el LED'84 Z.104 podrían preguntarse qué les sucedió a los llamados operadores activos. Bien, esta característica fue suprimida porque:

a) no se necesita ya que los operadores corrientes junto con los procedimientos y/o las macros suministran el mismo poder de expresión;

b) destruía la facilidad de lectura de las ecuaciones,

c) muchos usuarios tenían problemas para usarla correctamente,

d) no cuadra con el modelo de tipos de datos abstracto basado en álgebras iniciales que es el modelo escogido como base matemática de esta parte del LED.

D.7 *Directrices adicionales para dibujar y escribir*

Para cada concepto del LED la Recomendación especifica el modo principal de representar el concepto. Por ejemplo, el concepto tarea se representa en LED/GR por medio de un rectángulo y en LED/PR por medio de la palabra clave TASK.

Las directrices para el usuario suplementan la Recomendación al especificar reglas para dibujar y escribir, aplicables a todos los conceptos, con el objeto de recalcar lo que en dibujar/escribir puede considerarse apropiado, equivocado o de mal aspecto.

D.7.1 *Directrices para el LED/GR*

D.7.1.1 *Generalidades*

Las directrices generales para dibujar diagramas son:

– La secuencia de lectura deberá ser de arriba a abajo y de izquierda a derecha.

– Los diagramas no deben contener demasiada información al mismo nivel. A menudo es adecuado subdividir los diagramas grandes en subdiagramas que cubran diferentes partes o aspectos; utilizando, por ejemplo, el mecanismo de referenciación.

D.7.1.2 *Accesos de entrada y accesos de salida*

Los accesos de entrada y los accesos de salida hacia/desde cualquier símbolo deberán dibujarse verticalmente (correspondiendo a la lectura de arriba a abajo), y solamente donde no es práctico se utilizarán accesos de entrada y de salida horizontales.

Las excepciones a esta regla general son:

– Las decisiones con dos o tres accesos de salida se dibujan generalmente con dos accesos de salida horizontales (más uno vertical).

– Las llamadas de macro utilizan conexiones horizontales y verticales.

– Tanto los conectores de entrada como los de salida tienen generalmente accesos de entrada y de salida horizontales.

Sólo habrá líneas diagonales en casos excepcionales (por ejemplo, para canales y rutas de señales).

Los accesos de entrada y los accesos de salida verticales deberán dividir el símbolo en dos partes de igual longitud horizontal.

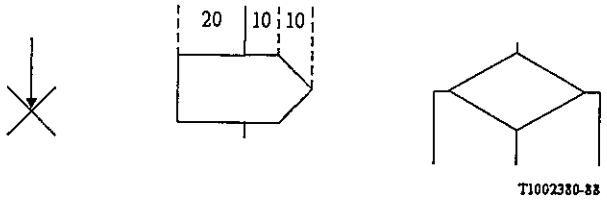


FIGURA D-7.1.1

Accesos de entrada y de salida dibujados correctamente

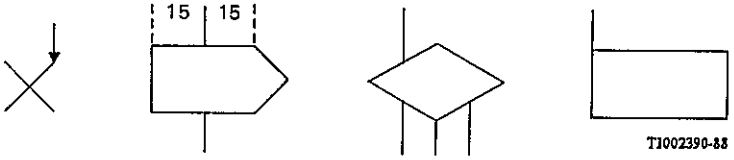


FIGURA D-7.1.2

Accesos de entrada y de salida dibujados incorrectamente

D.7.1.3 *Símbolos*

a) Los símbolos deberán dibujarse de modo que los ejes verticales y horizontales coincidan con los dos ejes del papel.

b) La imagen obtenida por simetría vertical de símbolos se permite solamente para símbolos de entrada, salida, ampliación de texto y comentario (véase figura D-7.1.3).

c) La relación general entre altura y longitud para todos los símbolos en los gráficos y también para los símbolos de referenciación es 1 : 2.

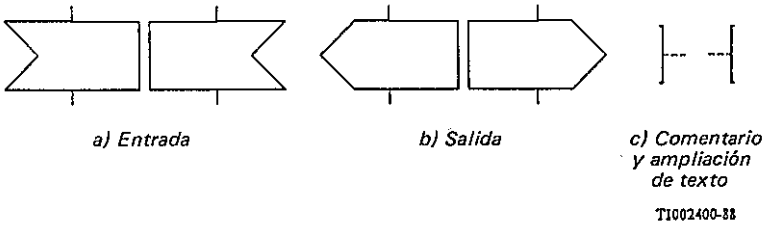


FIGURA D-7.1.3

Los cuatro símbolos permitidos de simetría vertical

D.7.1.4 *Plantilla*

En la parte interior de la tapa posterior de este fascículo hay una plantilla con los símbolos del LED/GR. Se representa esquemáticamente en la figura D-7.1.4.

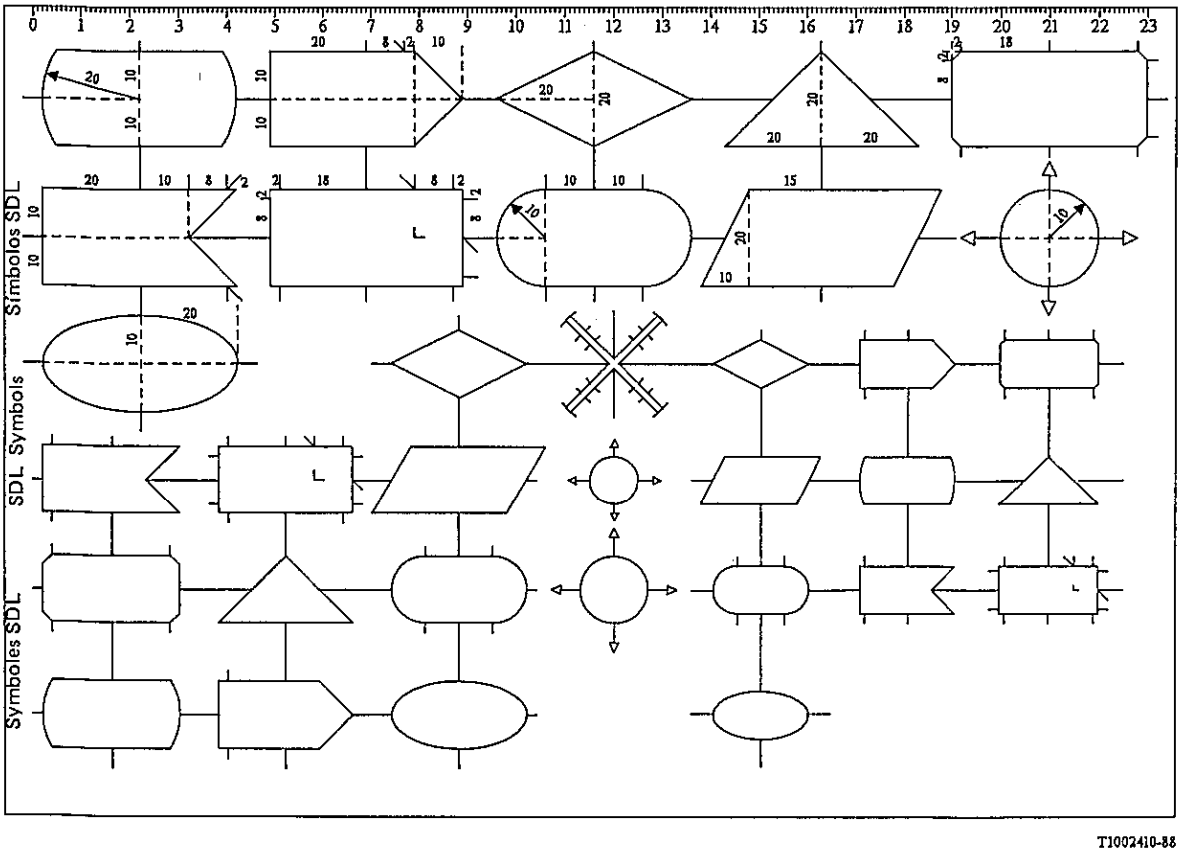


FIGURA D-7.1.4

Representación esquemática de la plantilla

En la figura, los símbolos de entrada, salida, decisión, alternativa, proceso, comienzo, tarea, estado, conservación, referencia de servicio, conector y parada se muestran directamente en 3 tamaños, esto es, 20 × 40 mm, $20/\sqrt{2} \times 40/\sqrt{2}$ mm y 10 × 20 mm. Se indican las relaciones internas para el tamaño más grande solamente.

Los símbolos de llamada de procedimiento, llamada de macro y crear pueden construirse a base del símbolo de tarea trazando las líneas horizontales o verticales adicionales indicadas en la figura.

El símbolo de comienzo de procedimiento puede construirse a partir del símbolo comienzo de proceso dibujando las líneas verticales adicionales indicadas.

El símbolo de retorno es una combinación de los símbolos de conector y de parada.

Los símbolos de comentario, ampliación de texto y lista de señales se dibujan utilizando el símbolo de tarea.

Los símbolos de entrada prioritaria y salida prioritaria pueden construirse a partir de los símbolos de entrada y salida dibujando la línea adicional indicada.

El símbolo de referencia de procedimiento puede construirse a partir del símbolo de referencia de proceso dibujando las dos líneas verticales adicionales, como se indica.

Los símbolos de condición habilitadora y de señal continua pueden dibujarse usando el símbolo de parada.

Los canales, rutas de señales y la línea que va al símbolo de ampliación de texto son líneas de trazo continuo.

La línea que va a un símbolo de comentario es una línea de rayas y blancos que guardan la relación 1 : 1.

El símbolo de texto se dibuja usando el símbolo de tarea y doblando la esquina superior derecha, como se indica.

Todos los símbolos recomendados se definen en la Recomendación. En el Anexo C1 – Resumen de la sintaxis LED/GR se puede encontrar una descripción general de los símbolos recomendados. Los tres tamaños que se muestran son los preferidos. Si se utilizan otros tamaños la relación deberá ser siempre la misma (es decir, 1 : 2). Los tamaños que se muestran, es decir, 40 mm, 28 mm y 20 mm de longitud permiten la reducción fotográfica de papel A3 a A4 con tamaños compatibles de símbolos (como 40 mm/$\sqrt{2}$ = 28 mm y 28 mm/$\sqrt{2}$ = 20 mm).

D.7.2 Directrices para el LED/PR

Las directrices generales para redactar LED textual son:

– La secuencia de lectura deberá ser de arriba a abajo y de izquierda a derecha.

– El texto deberá dividirse en partes que cubran aspectos diferentes.

– Los comentarios sobre las sentencias deberá comenzar en la misma columna.

– Las líneas deberán sangrarse. El sangrado debe seguir la jerarquía usual de los conceptos LED, como se muestra en el ejemplo de la figura D-7.1.5.

```
SYSTEM A;
   SIGNAL S1, S2;
   BLOCK B;
      PROCESS P;
         START;
         NEXSTATE ST1;
         STATE F;
            INPUT G: . . .
            INPUT F: . . .
            . . .
         ENDSTATE F;
      ENDPROCESS P;
   ENBLOCK B;
ENDSYSTEM A;
```

FIGURA D-7.1.5

Ejemplo de sangrado de texto en LED

D.8 *Documentación*

D.8.1 *Introducción*

La ISO define un documento como un «volumen limitado y coherente de información almacenado en un soporte de una forma recuperable». Por tanto, ha de considerarse como una unidad lógica que está estrictamente delimitada. Se utilizan documentos para facilitar toda la información relativa a un sistema que se especifica por medio del LED.

Cuando se utiliza papel como soporte físico para el almacenamiento de un documento, el término documento se aplica a menudo incorrectamente a las hojas de papel más bien que a su contenido lógico. Con la utilización creciente de medios de almacenamiento magnéticos, el término va volviendo a su significado original.

El presente capítulo se refiere a la organización lógica de los documentos más bien que a su organización física, la cual queda a criterio de los usuarios. La semejanza que presentan los requisitos de las organizaciones lógica y física de los documentos permite ofrecer al lector, en lo que sigue, algunas indicaciones útiles para facilitarle el establecimiento de una organización física de los documentos.

Subdividiendo la información en un número adecuado de documentos, el sistema puede resultar más legible y manejable.

El lenguaje no recomienda determinados documentos o estructuras de documentos. Sin embargo, se ofrecen algunas proposiciones para ayudar al usuario en el manejo de los documentos.

D.8.2 *Tipos de representación de sistemas*

Cuando se especifica un sistema utilizando LED el resultado es un conjunto de definiciones en LED/GR y/o LED/PR.

Estas definiciones pueden estar anidadas o ser secuenciales dependiendo de que el tipo de representación de sistema sea jerárquico o plano. En las figuras D-8.2.1 y D-8.2.2 se describe un sistema con las dos representaciones alternativas en LED/GR. Las dos figuras no son especificaciones completas del sistema ya que, por simplicidad, solamente se muestran entradas y salidas y faltan las posibles señales y definiciones de datos.

Se permite, naturalmente, utilizar una combinación de las representaciones cuando se especifica un sistema. Cuando las definiciones son secuenciales como en la figura D-8.2.1, ellas se «referencian», lo cual es un mecanismo posible para las definiciones tanto en LED/PR como en LED/GR.

Si se toma una especificación de sistema como un conjunto de definiciones, un documento puede considerarse como un contenedor para estas definiciones.

Si el sistema es pequeño y jerárquico como el de la figura D-8.2.2, un solo documento es suficiente. Si se utiliza una representación plana como en la figura D-8.2.1 pueden utilizarse más documentos, por ejemplo, un documento por definición.

Cuando se elige el tipo de representación es necesario considerar el tipo de documentos deseado. Para tener un documento por definición, se necesita una representación plana. Si se quiere un solo documento para toda la especificación del sistema, se necesita una representación jerárquica.

El caso normal es posiblemente una mezcla de estas representaciones. Al decidir esta mezcla se aplican las reglas siguientes:

1) Una definición no debe dividirse entre varios documentos.

2) Si se quiere una definición en un documento separado, debe referenciarse y no anidarse.

3) Cuando se utiliza el concepto de página lógica para dividir un diagrama en varias páginas de diagramas, las páginas de diagramas deberán coincidir con las páginas físicas del documento (véase figura D-8.2.4).

4) Si un diagrama ocupa más de una página debe referenciarse, no anidarse.

D.8.3 *Estructura de documento*

El conjunto de documentos que abarca la definición completa del sistema puede estructurarse. Una estructura de documento, en la cual los documentos se refieren a subdocumentos, puede asociarse a cualquier entidad del sistema, tal como sistema, bloques y procesos (véase figura D-8.3.1).

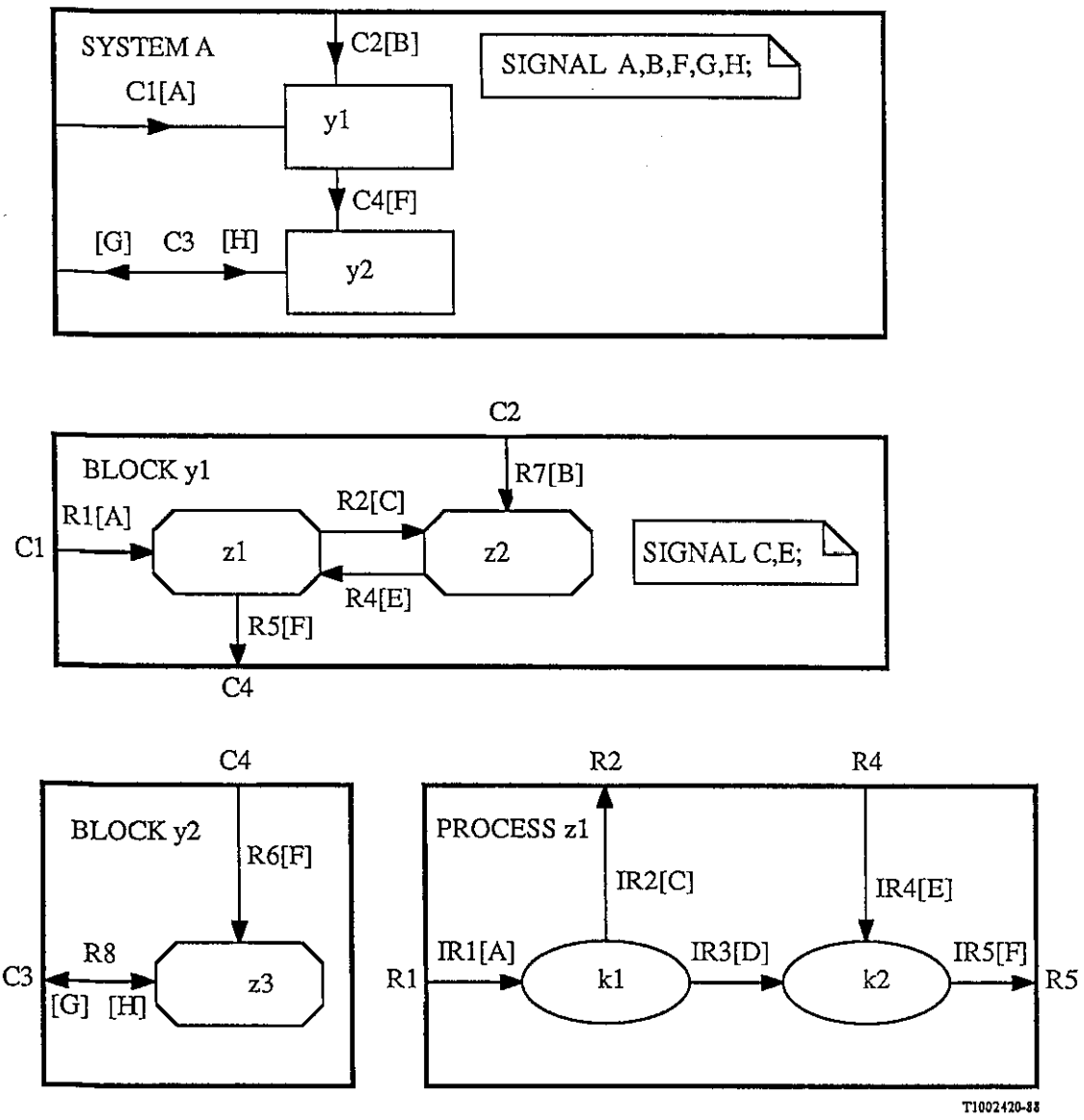


FIGURA D-8.2.1a)

a) Ejemplo de diagramas secuenciales (referenciados) utilizados para la representación plana de un sistema

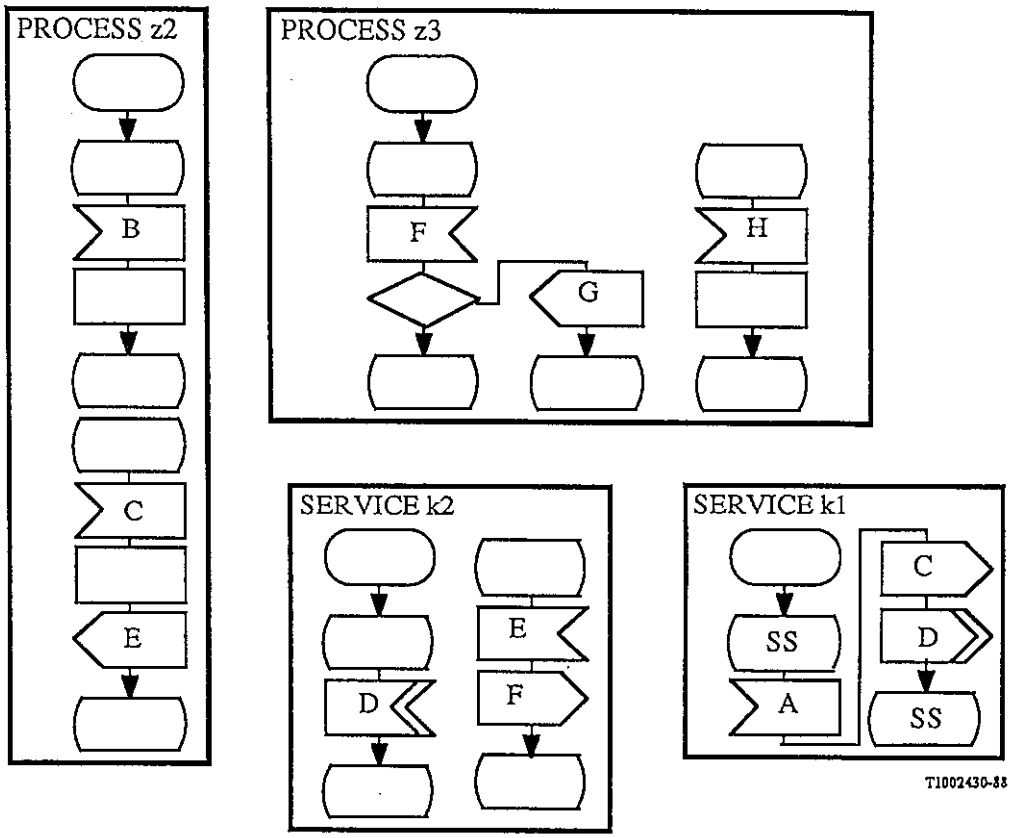


FIGURA D-8.2.1b)

b) Ejemplo de diagramas secuenciales (referenciados) utilizados para la representación plana de un sistema

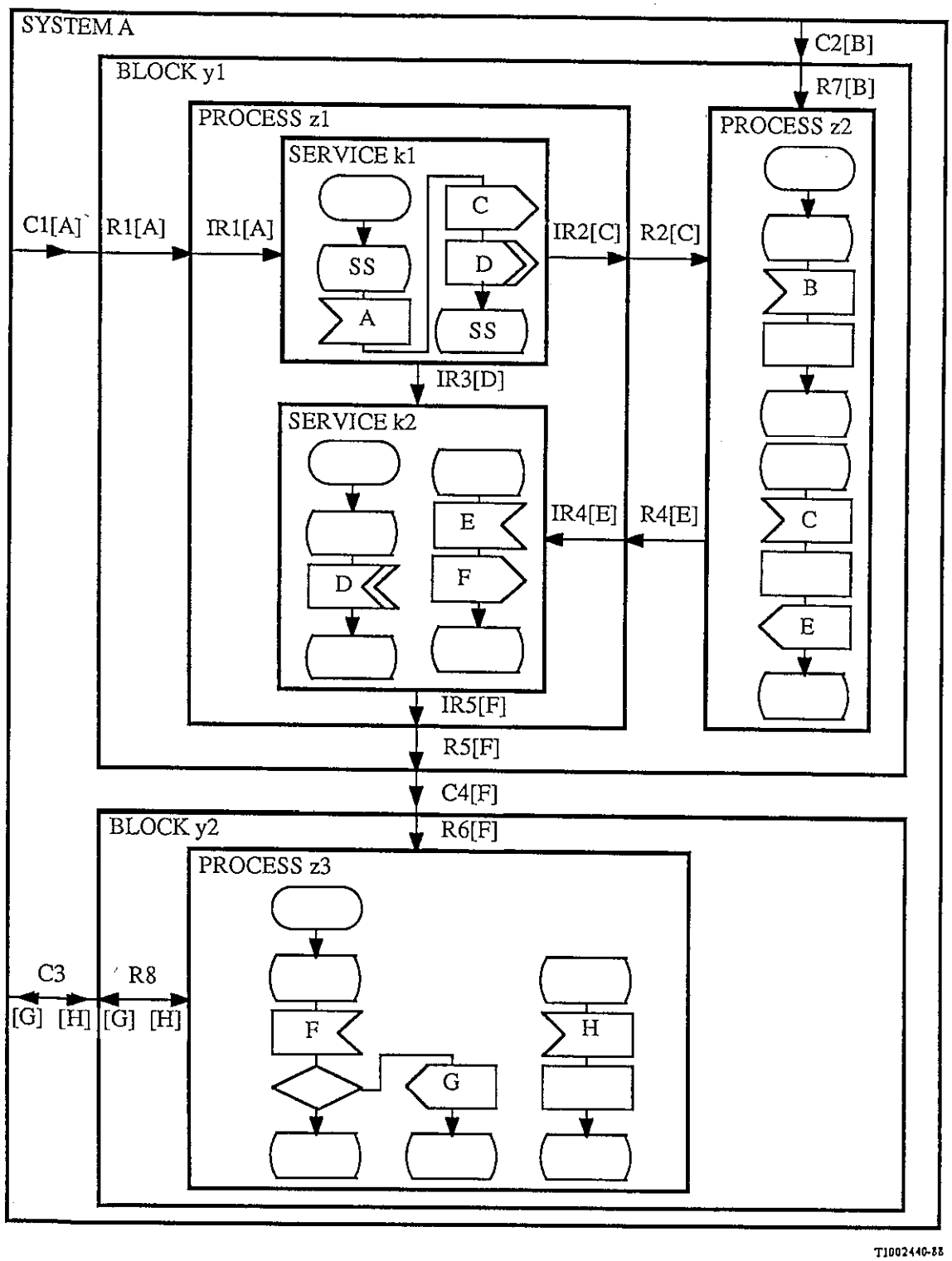


FIGURA D-8.2.2

Ejemplo de definiciones anidadas usadas para la representación jerárquica de un sistema

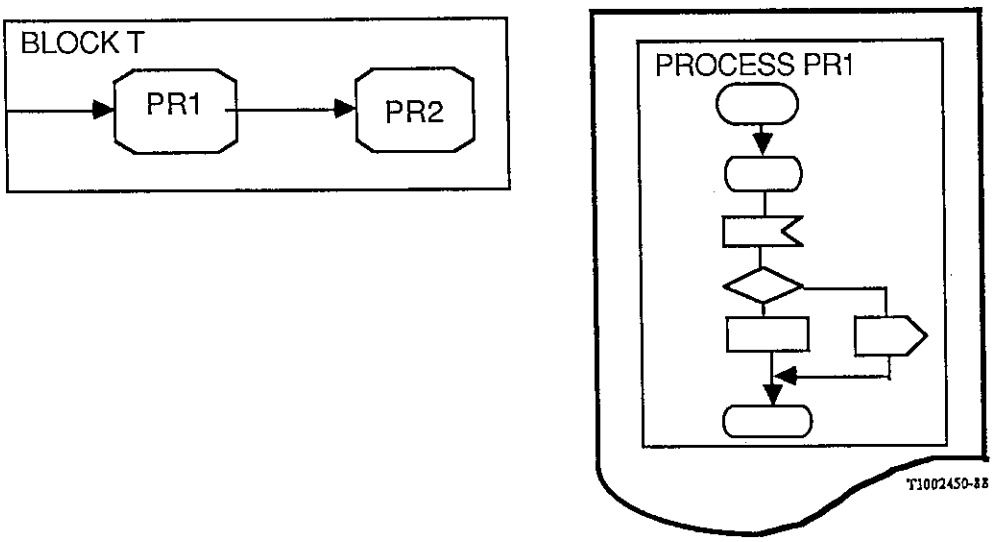


FIGURA D-8.2.3

Una definición referenciada puede colocarse en un documento separado

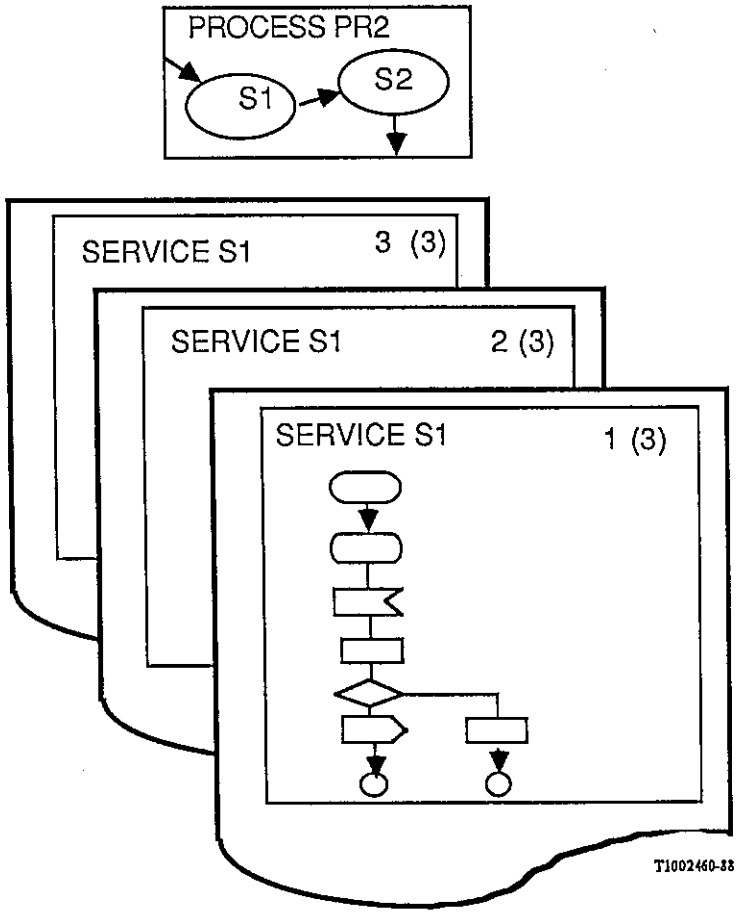


FIGURA D-8.2.4

Un diagrama de servicio referenciado repartido en 3 páginas

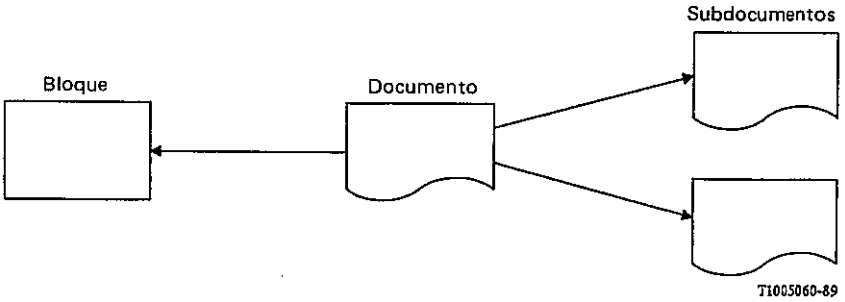


FIGURA D-8.3.1

Una estructura de documento

D.8.4 *Mecanismo de referenciación*

El mecanismo de referenciación del lenguaje, en el que se utilizan nombres para referencias entre conceptos, puede utilizarse también para referencias entre documentos. Cuando un documento coincide con una definición, este es un método natural.

D.8.5 *Clasificación de documentos*

Los documentos pueden clasificarse de acuerdo con los tipos de definiciones que contienen.

En esta clasificación deben colocarse en documentos separados por lo menos las definiciones de proceso o servicio que describen el comportamiento dinámico del sistema. Estos documentos pueden incluir también definiciones de variables.

En la figura D-8.5.1 se ofrece una posible estructura de documento para un sistema.

En este ejemplo las definiciones de canal y ruta de señales están incluidas en los documentos para las definiciones de sistema, bloque y proceso. Las definiciones de señal, las definiciones de datos y las definiciones de lista de señales se colocan en documentos separados y se ha supuesto que todas las definiciones de datos están a nivel de sistema.

Las definiciones de procedimiento, las definiciones de macro y las definiciones de servicio forman subdocumentos del documento de proceso.

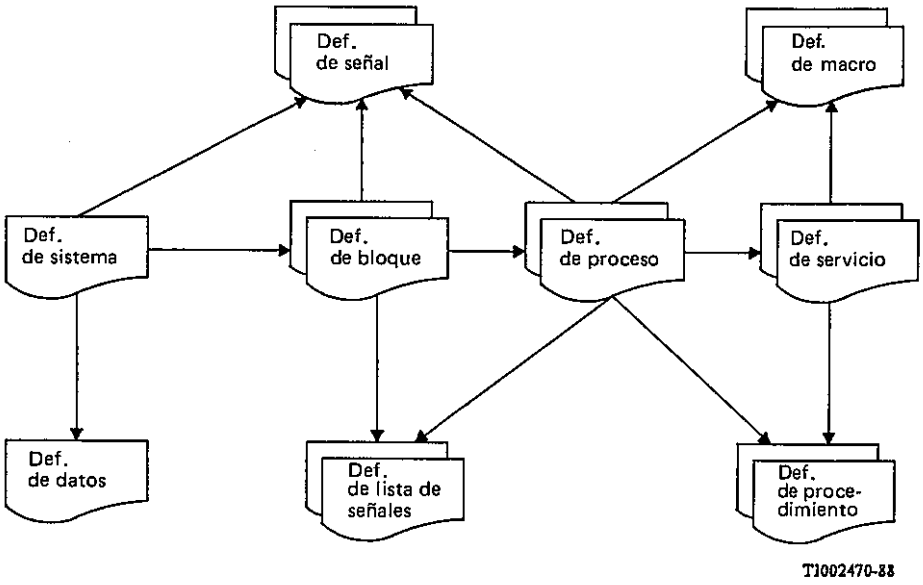


FIGURA D-8.5.1

Una posible estructura de documentos para un sistema

Si una función de sistema está formada por diferentes servicios, los servicios pueden describirse en un documento común.

Las diferentes definiciones de servicio pueden colocarse unas tras de otras en un documento de servicio pero también es posible tenerlas lado a lado en la misma página de documento. Esta última disposición de documento proporciona una buena comprensión de la interacción entre los servicios. La figura D-8.5.2 es un ejemplo de una página de documento de un documento de servicio.

Para especificaciones de grandes sistemas deben proporcionarse «índices de materias» del sistema a fin de indicar dónde encontrar los estados, las entradas, las salidas, etc. Además, los índices de materias deben contener también todos los conceptos, es decir, dónde encontrar las definiciones y dónde se las utiliza. Por ejemplo, sistemas, bloques canales, señales, procesos, servicios, macros, procedimientos.

Esos índices de materias del sistema pueden crearse como documentos separados.

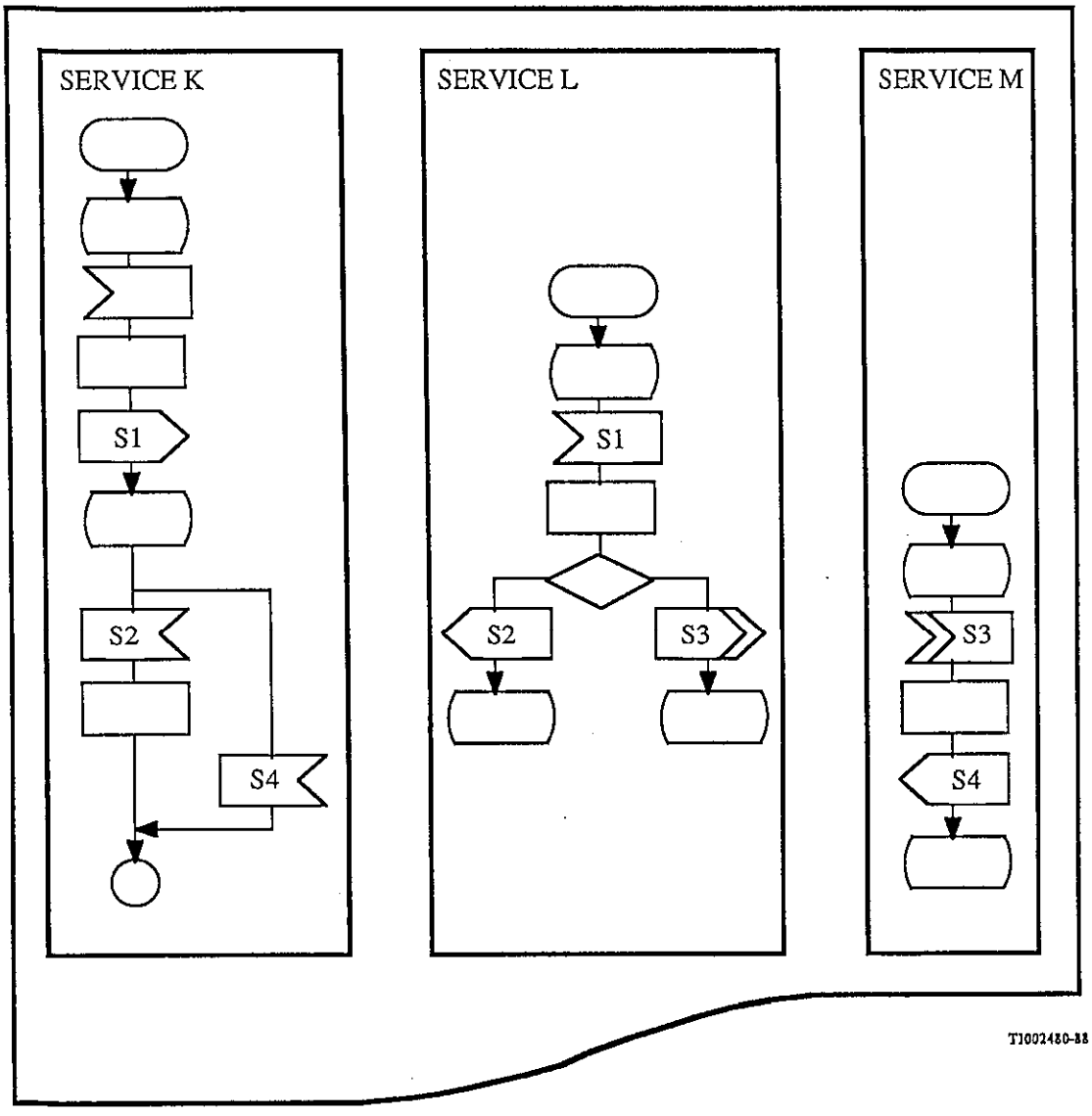


FIGURA D-8.5.2

Una página de un documento de servicio

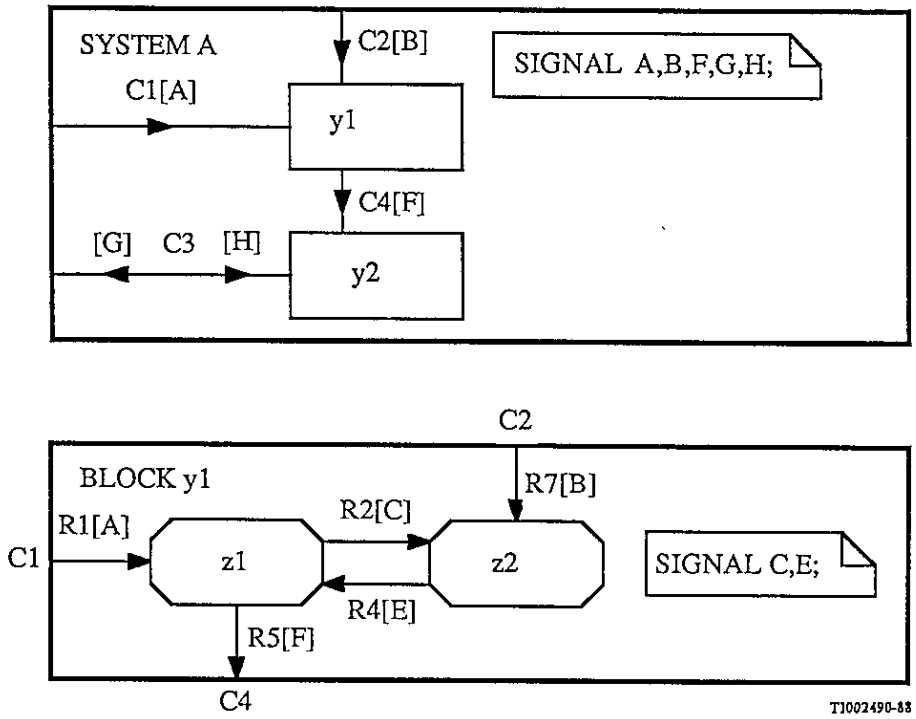
D.8.6 *Combinación de LED/GR y LED/PR*

LED/GR y LED/PR pueden utilizarse juntos para especificar un sistema.

Los conceptos de sistema, bloque, proceso, servicio, procedimiento, macro, canal, etc. pueden especificarse tanto en LED/GR como en LED/PR.

El cambio de LED/PR a LED/GR o de LED/GR a LED/PR se efectúa utilizando el mecanismo de referencia del lenguaje. Un concepto que se referencia en LED/PR puede especificarse en LED/GR y un concepto referenciado en LED/GR puede especificarse en LED/PR.

La figura D-8.6.1 es una especificación de sistema «completa» que utiliza una combinación de LED/PR y LED/GR. Es el mismo sistema de las figuras D-8.2.1 y D-8.2.2. Cada definición del ejemplo puede situarse en un documento separado.




```
BLOCK y2;
   SIGNALROUTE R8FROM ENV TO z3 WITH H;
                FROM z3 TO ENV WITH G;
   SIGNALROUTE R6FROM ENV TO z3 WITH F;
   CONNECT C3 AND R8;
   CONNECT C4 AND R6;
   PROCESS z3 REFERENCED;
ENDBLOCK y2;
```

```
PROCESS z1;
   SIGNALROUTE IR1 FROM ENV TO k1 WITH A;
   SIGNALROUTE IR2 FROM k1 TO ENV WITH C;
   SIGNALROUTE IR3 FROM k1 TO k2 WITH D;
   SIGNALROUTE IR4 FROM ENV TO k2 WITH E;
   SIGNALROUTE IR5 FROM k2 TO ENV WITH F;
   CONNECT R1 AND IR1;
   CONNECT R2 AND IR2;
   CONNECT R4 AND IR4;
   CONNECT R5 AND IR5;
   SERVICE k1;
      START;
      STATE SS;
         INPUT A;
         OUTPUT C;
         PRIORITY OUTPUT D;
         NEXSTATE SS;
   ENDSERVICE k1;
   SERVICE k2 REFERENCED;
ENDPROCESS z1;
```

FIGURA D-8.6.1a)

a) Especificación de sistema con una combinación posible de LED/GR y LED/PR

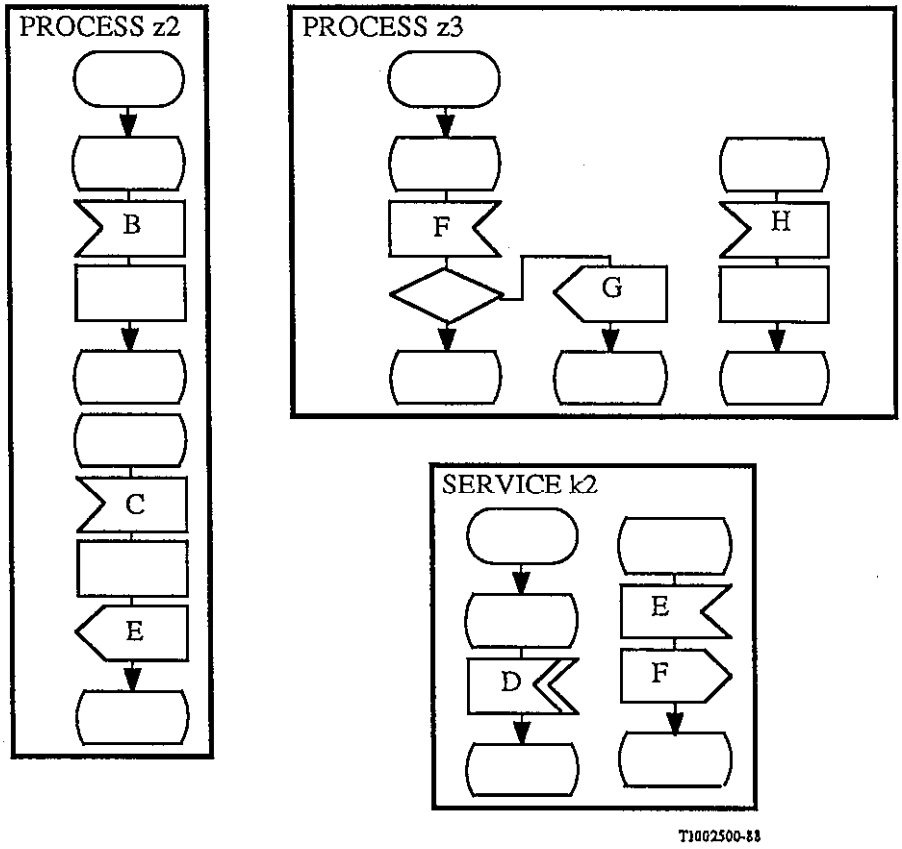


FIGURA D-8.6.1 b)

b) Especificación de sistema con una combinación posible de LED/GR y LED/PR

D.9 *Relaciones de correspondencia*

En este punto se describen algunos aspectos de la correspondencia entre LED y CHILL (§ D.9.1), así como la correspondencia entre el LED/GR y LED/PR (§ D.9.2).

D.9.1 *Correspondencia entre LED y CHILL*

A continuación se ilustran algunas formas posibles de correspondencia entre LED y CHILL. Se hace brevemente y no se pretende ser exhaustivo ni tampoco se sugiere que alguna de estas formas tenga que utilizarse en la práctica.

El análisis de la correspondencia deberá considerar no solamente el compilador CHILL y la máquina objetivo, sino ser más general. El establecimiento de una correspondencia es una actividad intelectual muy compleja y solamente a través de la experiencia los diseñadores/programadores pueden decidir sobre una estructura particular de un programa CHILL que se utilice para implementar una representación LED particular. Esto también se aplica a la representación de las funciones implementadas por un programa CHILL. Una correspondencia exacta (si se consigue) no es necesariamente la mejor manera de usar LED para representar las funciones implementadas en CHILL.

Al utilizar este método, la estructura global de un programa CHILL derivado de un diagrama LED aparecería como se muestra en la figura D-9.1.1.

```
Declaring: MODULE
        /*    módulo CHILL que contiene las señales y variables asociadas, utilizadas en el diagrama LED   */
        GRANT
        /*    adjudicación de señales y variables                                                          */
        SIGNALS
        /*    definición de señales                                                                        */
        SYNMODE (OR NEWMODE)
        /*    definición de tipo                                                                           */
END Declaring;

Functional_Block: MODULE
        /*    módulo que contiene la parte procedimiento del diagrama LED                                  */
        SEIZE
        /*    toma de todas las señales y variables que pueden recibirse y transmitirse desde (hacia) este bloque
              funcional                                                                                    */
        /*    definición y declaración de datos; estos datos son globales a todos los procesos pertenecientes a
              este módulo                                                                                  */
        Process name: PROCESS ( );
            /*    definición y declaración local de datos                                                  */
            nextstate:=...;
            join:=none;
            DO FOR EVER;
              state_loop: CASE nextstate OF
              /*    bucle en la variable nextstate que indica el estado LED                              */
              (state_label1): RECEIVE CASE
                                (signal name1):
                                .
                                . . .  .
                                .
                                (signal namen):
              ESAC state_loop;
              DO WHILE join/=none;
                CASE join OF
                    (join_lab1): join: = none;
                                 .
                                 . . .  .
                                 .
                    (join_labm): join: = none;
                                 .
                                 .
                ESAC;
            OD;
        OD
    END process_name;
END Functional_Block;
```

FIGURA D-9.1.1

Estructura global de un programa CHILL derivado de un diagrama LED

Las figuras D-9.1.2 a D-9.1.5 dan algunos ejemplos de establecimiento de correspondencia entre construcciones de los dos lenguajes. Se refieren a las siguientes construcciones LED:

– estado y recepción/conservación de señales; selección de un estado-siguiente;

– salida;

– unión;

– decisión.

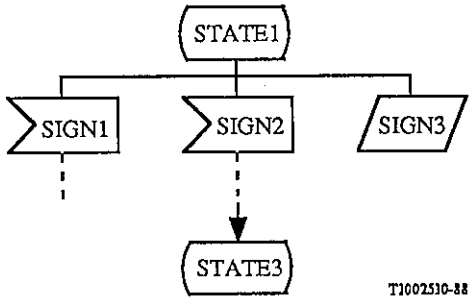
El módulo declarante contiene la definición y la declaración de todas las señales utilizadas en el diagrama LED transformado y todas las variables asociadas con estas señales. Todas estas variables son adjudicadas al módulo que representa el bloque funcional del diagrama LED.

El módulo de bloque funcional representa el comportamiento (parte procedimiento) de los procesos LED.

En este esquema de traducción, cada proceso LED se representa como un bucle infinito, una variable llamada "nextstate" (estado-siguiente) indica el estado que hay que examinar y una variable llamada "join" (unión) indica posibles puntos de unión que determinan conjuntos comunes de sentencias.

Se efectúa una selección del valor de estado-siguiente, por medio de la construcción "case" (caso) de CHILL; cada anotación de caso identifica un estado LED. En cada anotación se efectúa una selección entre las posibles señales de entrada. Cada señal de entrada determina el conjunto de acciones que hay que realizar (el «trayecto de transición»).

Cada trayecto de transición termina con una asignación relativa a la variable «estado-siguiente», que determina directamente el estado siguiente que hay que examinar, o relativa a la variable «unión». Un bucle de selección subsiguiente en el valor corriente de la variable «unión» permite terminar cada transición, en un sentido LED, y finalmente asignar un valor a la variable «estado-siguiente».



a) LED

```
STATE1:
   RECEIVE CASE
      (SIGN1): ...
               ...
      (SIGN2): ....
               * NEXTSTATE := STATE3;
      ELSE GETOUT (LIST1)
   ESAC STATE1;
```

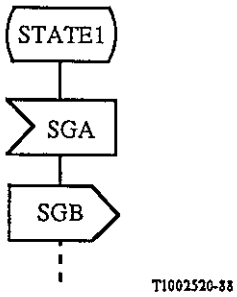
b) CHILL

FIGURA D-9.1.2

Ejemplo de correspondencia de STATE/INPUT/SAVE/NEXTSTATE

Uno de los principales problemas para relacionar el LED con el CHILL es la diferente semántica en la recepción de señales; de hecho, en tanto que el CHILL no consume (y, por consiguiente, no destruye) sino las señales esperadas (señales persistentes), al proceso. LED consume todas las señales recibidas y destruye las señales no esperadas para dicho estado. La discrepancia semántica se ha resuelto introduciendo la rutina incorporada GETOUT, como alternativa (trayecto ELSE) en la construcción CHILL RECEIVE CASE, como se muestra en la figura D-9.1.2. La rutina GETOUT incorporada en el CHILL, que conoce (por parámetros) la lista de señales de entrada y de conservación, destruye las demás señales puestas a disposición del proceso cuando es llamada.

Después de ejecutar la rutina GETOUT el selector de estado se inicializa a fin de repetir el bucle correspondiente a dicho estado hasta que se selecciona una señal de entrada válida (o llega, si no está ya presente).



a) LED

```
STATE1:
  RECEIVE CASE
    (SGA): pi := get_instance_value();
           send SGB to pi;
           nextstate := ... ;
    ELSE nextstate := state1;
  ESAC STATE1;
```

b) CHILL

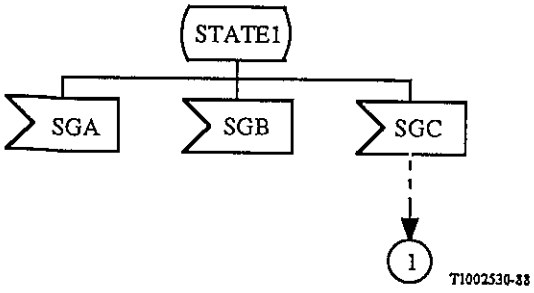
FIGURA D-9.1.3

Ejemplo de correspondencia de OUTPUT

Por ejemplo, una vez que se ha reconocido la señal de entrada SGA en la figura D-9.1.3, se selecciona la instancia de proceso de destino adecuada para la señal SGB y se envía la señal SGB.

Antes de enviar la señal SGB puede ser necesario rellenar algunos campos de información que la señal debe transportar. Esto puede efectuarse inmediatamente antes del envío de la señal o por anticipado.

Cuando se encuentra un punto de unión en el diagrama (véase la figura D-9.1.4), se asigna el valor adecuado a la variable "JOIN" (UNIÓN). Como se explica en la figura D-9.1.1, se realiza un bucle en el valor de la variable «unión» para determinar el estado siguiente que hay que examinar. Un punto de unión puede ser visto, desde el punto de vista del lenguaje de programación, como una construcción "goto" que reúne todos los puntos de unión de forma que puedan examinarse, lo que permite escribir la estructura completa del programa sin utilizar ningún "goto", lo cual facilita la lectura.



a) LED

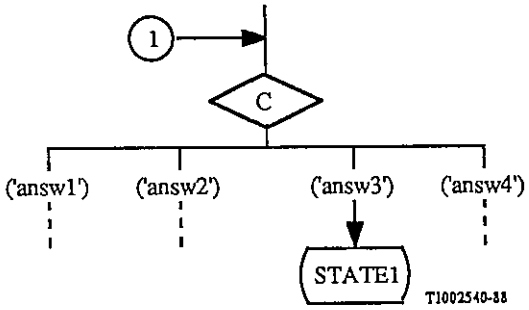
```
STATE1:
  RECEIVE CASE
    (S1 in m): case m.id of
                 (SGA): ... ;
                 (SGB): ... ;
                 (SGC): ... ; JOIN:=1;
                 ELSE nextstate := state1;
               esac;
  ESAC STATE1;
```

b) CHILL

FIGURA D-9.1.4

Ejemplo de correspondencia de JOIN

Una decisión LED tiene traducción directa a la construcción de caso del CHILL, como se muestra en la figura D-9.1.5.



a) LED

```
...
1: CASE C OF
    ('answ1'): ... ;
    ('answ2'): ... ;
    ('answ3'): nextstate := state1;
    ('answ4'): ... ;
   ESAC 1;
```

b) CHILL

FIGURA D-9.1.5

Ejemplo de correspondencia de DECISIÓN

D.9.2 *Correspondencia entre GR y PR*

Tomando en consideración las restricciones mencionadas acerca de las macros (§ D.5.1), el GR siempre puede hacerse corresponder con el PR, y viceversa.

Ejemplos de especificaciones equivalentes en ambas formas pueden encontrarse a lo largo de todo este texto.

D.10 *Ejemplos de aplicación*

D.10.1 *Introducción*

El § D.10 contiene algunos ejemplos de la utilización del LED. Los ejemplos están tomados del campo de aplicación de las telecomunicaciones; se intentó tomar ejemplos realistas y cubrir tantos conceptos LED como fuese posible.

Con los ejemplos se pretende mostrar el uso del LED, y no son especificaciones internacionales.

D.10.2 *El concepto de servicio*

El sistema siguiente es una ilustración del concepto de servicio. En el sistema son de especial interés para este ejemplo tres «funciones de sistema». Ellas son: una función de tratamiento de tráfico, una función de mantenimiento y una función de alarma. La figura siguiente, D-10.2.1, muestra cómo las funciones de sistema están compuestas de servicios subordinados en cinco procesos diferentes en cinco bloques.

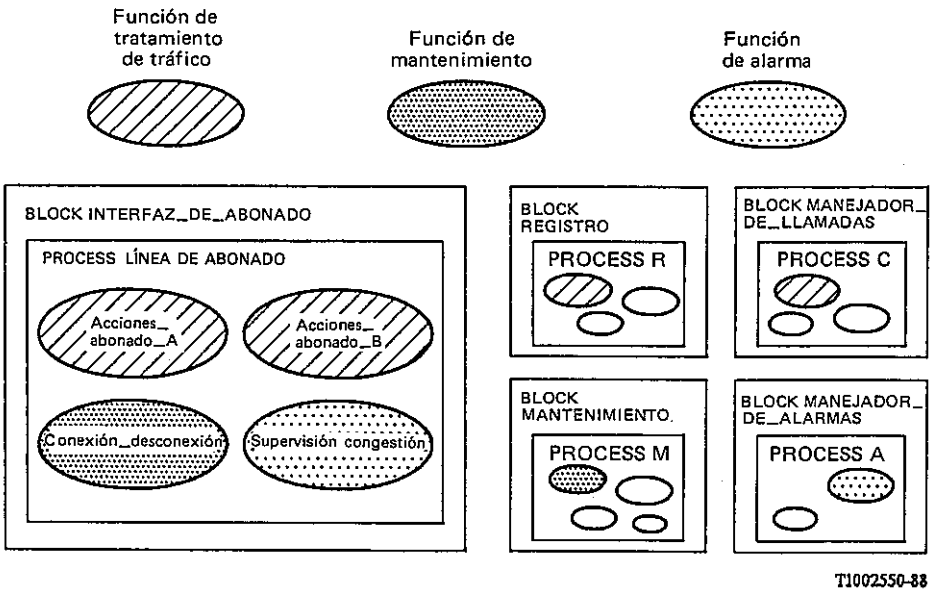


FIGURA D-10.2.1

Composición de funciones

La función de tráfico incluye el establecimiento y la terminación de una llamada telefónica.

Este ejemplo no es una documentación completa de las funciones a nivel de sistema, es sólo una descripción de los cuatro servicios del proceso «LINEA_DE_ABONADO».

Las tres figuras siguientes son el diagrama de sistema (figura D-10.2.2), el diagrama del bloque «INTERFAZ_DE_ABONADO» (figura D-10.2.3) y el diagrama del proceso «LINEA_DE_ABONADO» (figura D-10.2.4). En estos diagramas pueden verse los canales, rutas de señales y señales necesarias.

```
SYSTEM Concepto_de_servicio

/*Este sistema es un ejemplo que ilustra el concepto de servicio en LED. Se ha seleccionado para la
ilustración el bloque INTERFAZ_DE_ABONADO. El diagrama de sistema muestra los bloques
interconectados y los canales que se necesitan para los servicios en INTERFAZ_DE_ABONADO*/


SIGNAL A_DESCOLGADO,A_COLGADO,CIFRA_MARCADA,B_COLGADO B_DESCOLGADO,
  TONO_DE_MARCAR,TONO_CONG,TONO_DE_MARCAR_AUSENTE,TONO_CONG_
  _AUSENTE,TONO_LLAMADA_A_A,
  TONO_LLAMADA_A_A_AUSENTE,TONO_LLAMADA_A_B,TONO_LLAMADA_A _B_
  _AUSENTE,CONECTAR_REG_CIFRAS,
  CIFRA,DESCON_REG_CIFRAS,CONEXION,CONGESTION,TRAER_CIFRA_SIGUIENTE,
  CONECTAR_MANEJADOR_DE_LLAMADAS,A_ACTIVADO,A_DESACTIVADO,LINEA_
  _CONECTADA,LINEA_DESCONECTADA
  ENVIAR_TONO_LLAMADA,B_CONTESTA,DESCON_A,SEÑAL_LLAMADA,DESCON_B,
  ENVIAR_ALARMA,
  CESAR_ALARMA,AC_PET_CON1,AC_PET_CON2,AC_PET_DESC1,AC_PET_DESC2,
  PET_CON,PET_DESC,B_ACTIVADO,B_DESACTIVADO;


SIGNALLIST L1A = A_DESCOLGADO,A_COLGADO,CIFRA_MARCADA;
SIGNALLIST L1B = B_COLGADO,B_DESCOLGADO;
SIGNALLIST L2A =   TONO_DE_MARCAR,TONO_CONG,TONO_DE_MARCAR_AUSENTE,
                   TONO_CONG_AUSENTE;
                   TONO_LLAMADA_A_A,TONO_LLAMADA_A_A_AUSENTE;
SIGNALLIST L2B = SEÑ_LLAMADA_A_B,SEÑ_LLAMADA_A_B_AUSENTE;
SIGNALLIST LAL = ENVIAR_ALARMA,CESAR_ALARMA;
SIGNALLIST L1MAIN = PET_CON,PET_DESC;
SIGNALLIST L2MAIN =    AC_PET_CON1,AC_PET_CON_2,AC_PET_DESC1,AC_PET_DESC2;
SIGNALLIST L1REG = CONECTAR_REG_CIFRAS,DESCON_REG_CIFRAS;
SIGNALLIST L2REG = CONEXION,CONGESTION,TRAER_CIFRA_SIGUIENTE;
SIGNALLIST L1CALL = ENVIAR_TONO_LLAMADA,B_CONTESTA,DESC_A;
SIGNALLIST L2CALL = A_DESACTIVADO,A_ACTIVADO;
SIGNALLIST L3CALL = SEÑAL_LLAMADA,DESC_B;
SIGNALLIST L4CALL =    LINEA_CONECTADA,LINEA_DESCONECTADA,B_ACTIVADO,
                       B_DESACTIVADO;
```

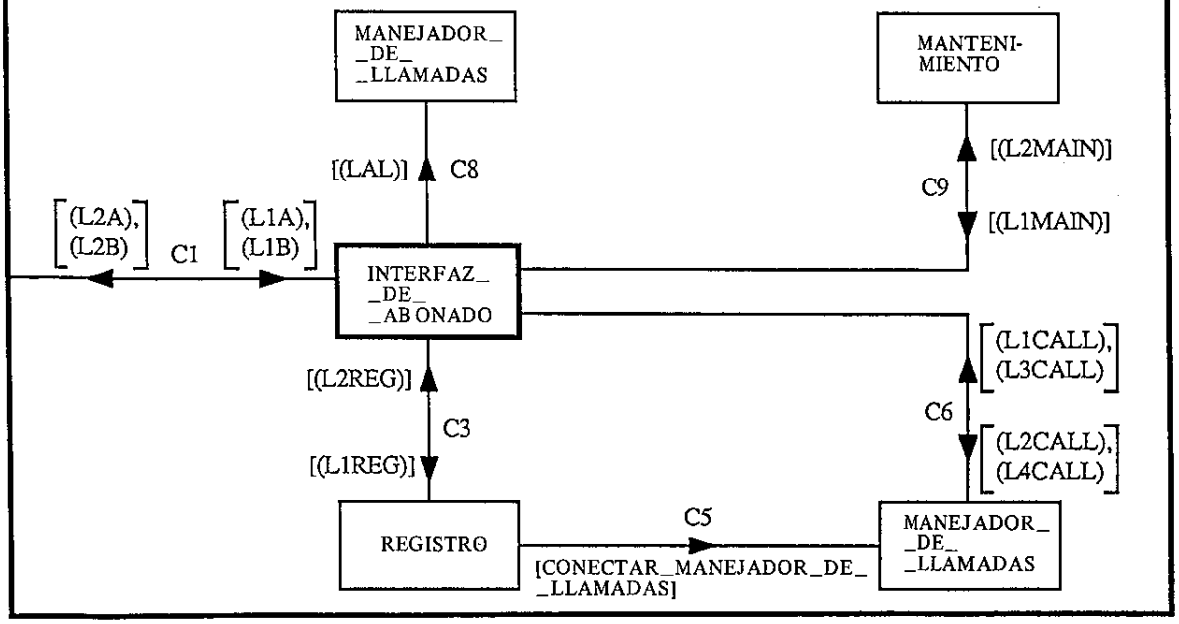


FIGURA D-10.2.2

Diagrama de sistema

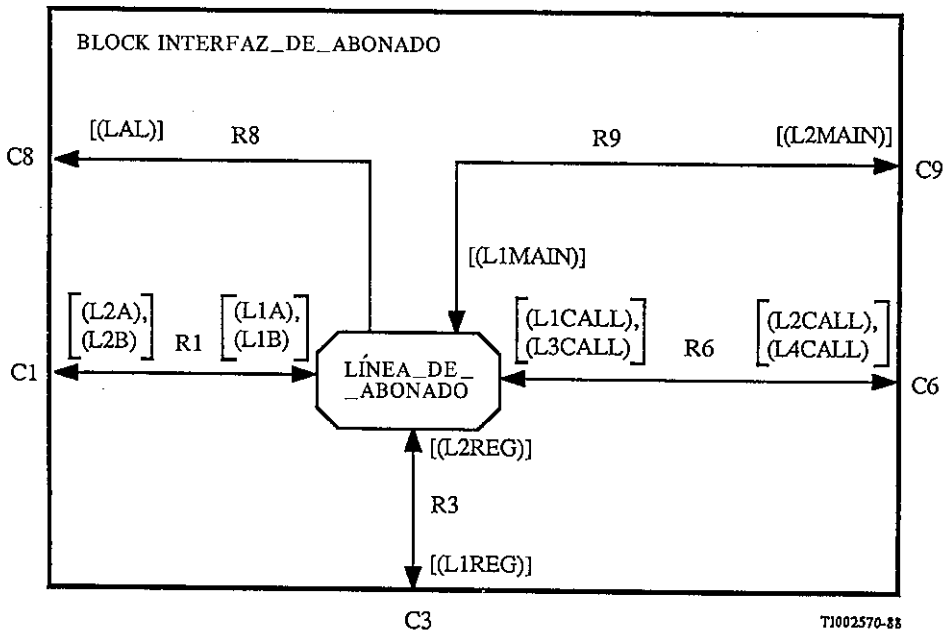


FIGURA D-10.2.3

Diagrama de bloque

Como puede verse en el diagrama de proceso (figura D-10.2.4) los servicios interfuncionan con señales prioritarias vía las rutas de señales IR01, IR02, IR03 e IR04. Pero los servicios también pueden interactuar y afectarse mutuamente mediante la variable “global” «conectado» declarada en el proceso.

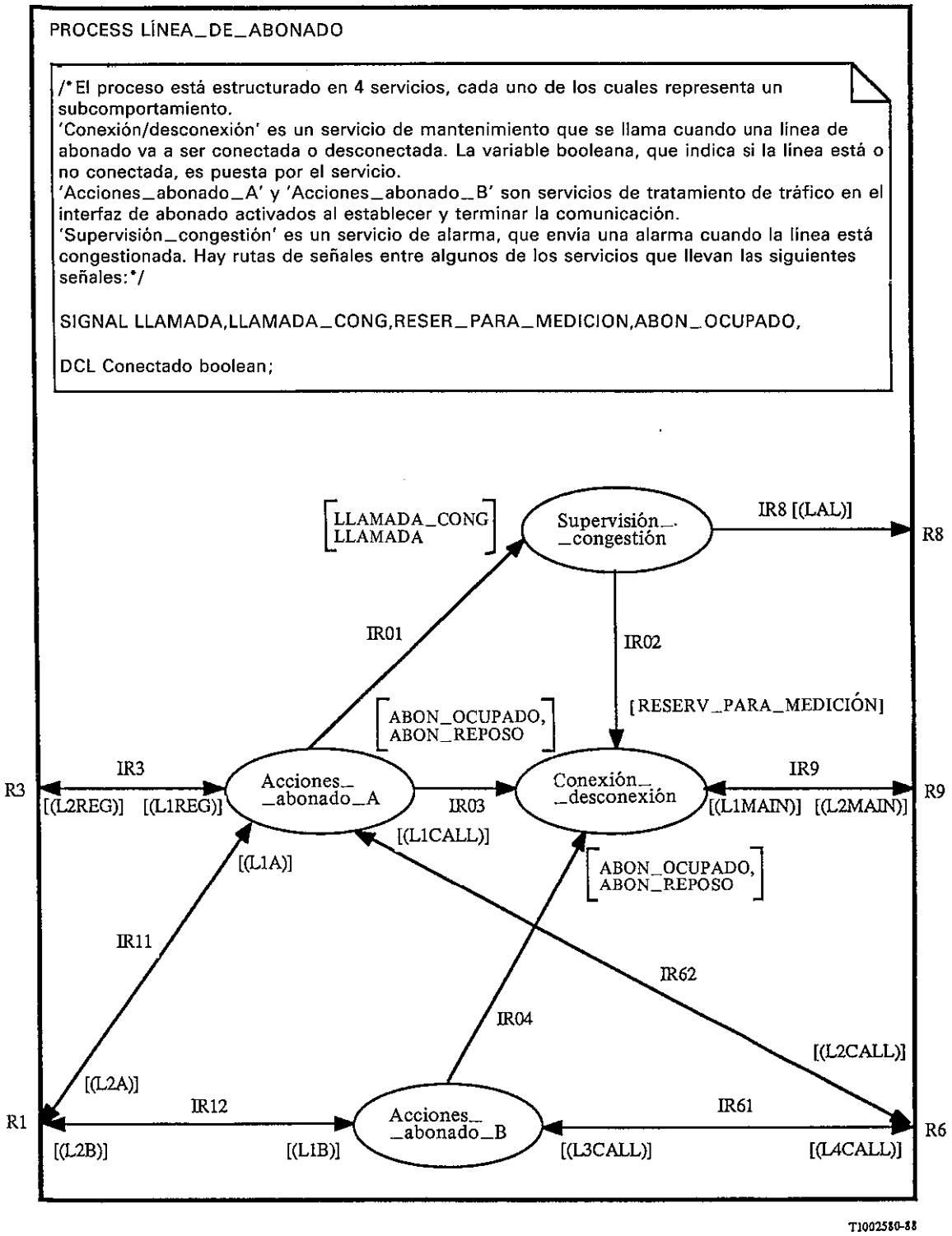


FIGURA D-10.2.4

Diagrama de proceso

En el ejemplo se incluyen algunos esquemas secuenciales para comprender mejor los servicios y interacción entre los bloques.

Los dos primeros esquemas secuenciales muestran el caso normal de interacción entre los bloques durante una llamada. En el tercer esquema se ilustra la interacción en caso de congestión del registro. Para simplificar los esquemas no se ha supuesto demora entre la emisión y la recepción de una señal (véanse las figuras D-10.2.5 y D-10.2.6).

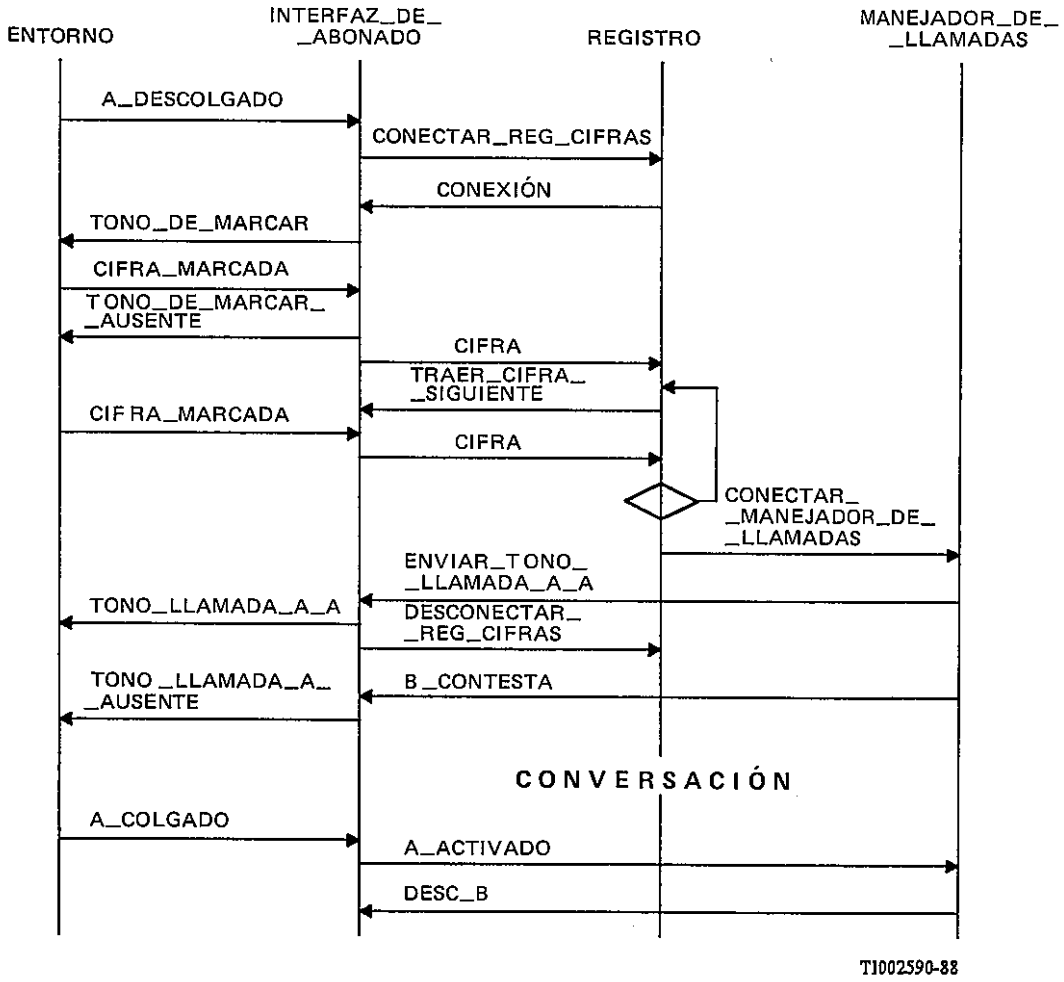
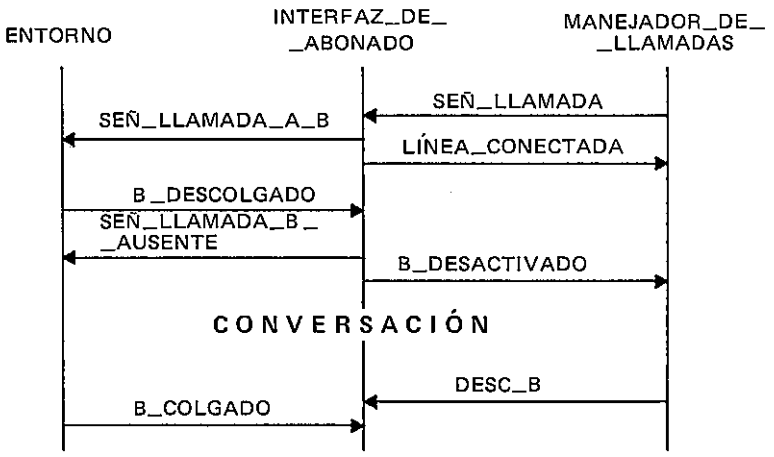


FIGURA D-10.2.5

Esquema secuencial. Interacción de bloques en el caso normal. Señales necesarias para el abonado A



Nota – Esquema secuencial. Interacción de bloques en el caso normal. Señales necesarias para el abonado B.

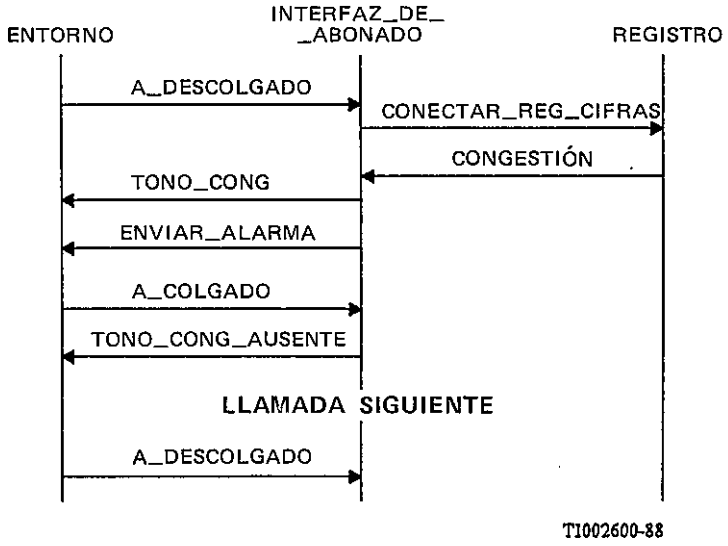
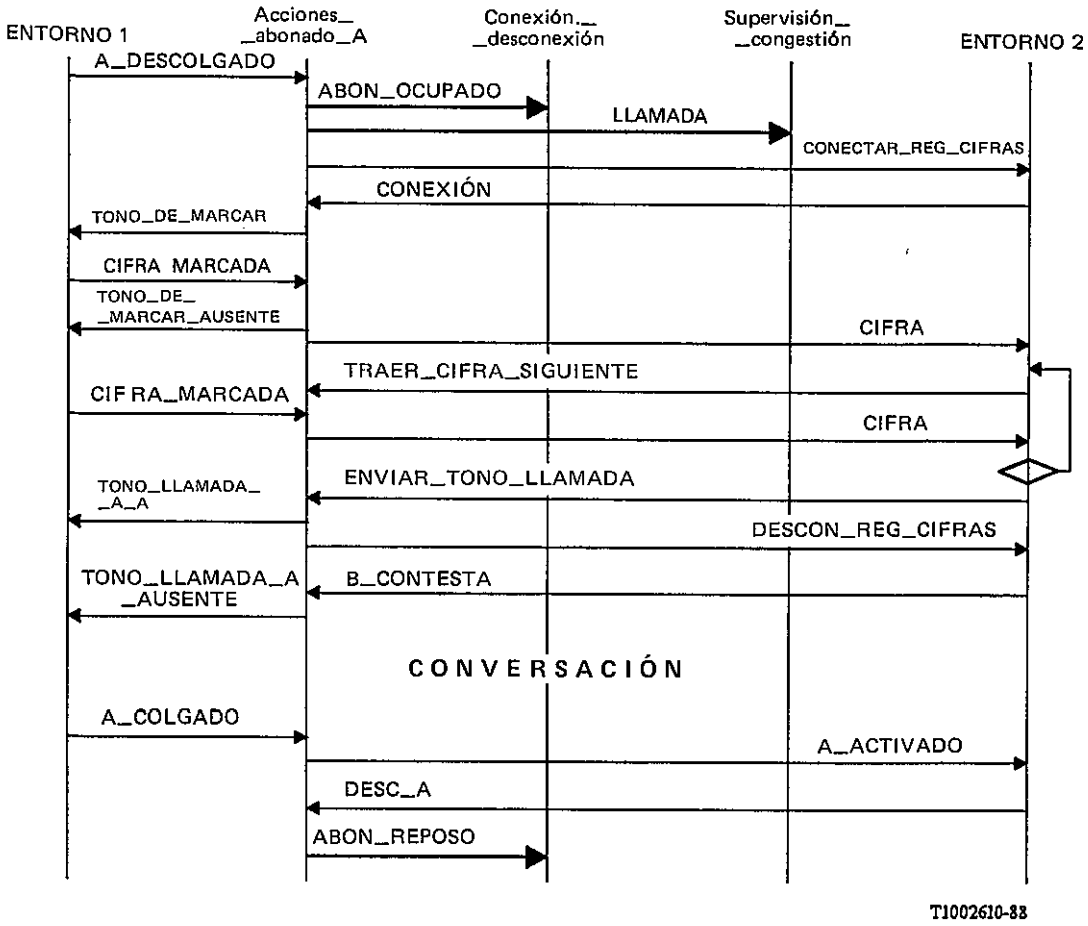


FIGURA D-10.2.6

Esquema secuencial. Interacción de bloques en caso de congestión de registro

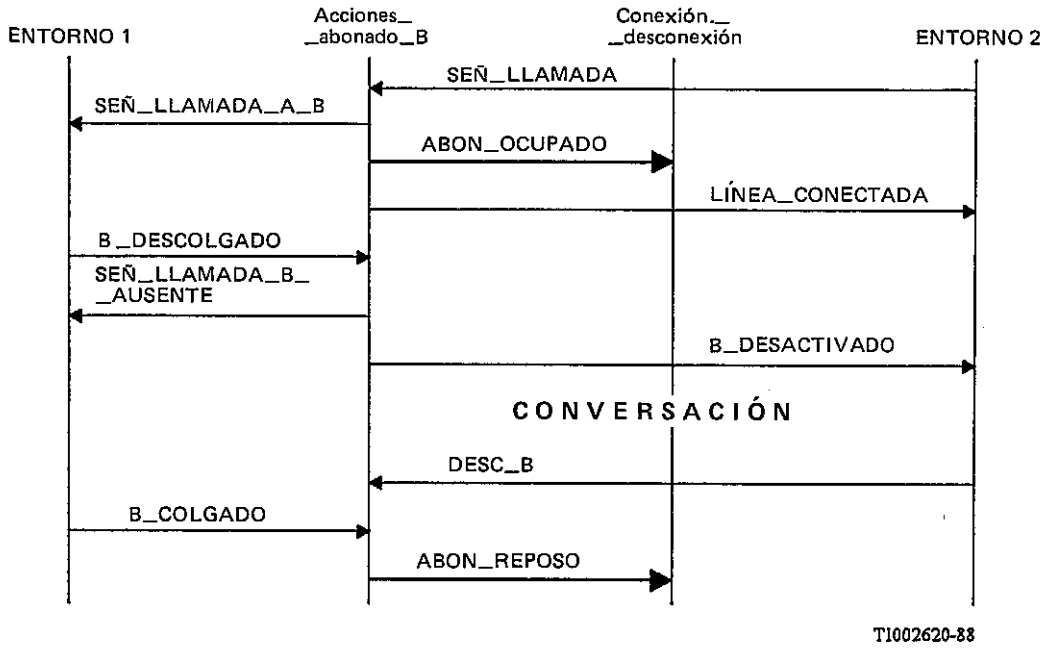
En los esquemas secuenciales que siguen se describe la interacción entre los servicios en el proceso «LINEA_DE_ABONADO» (“SUBSCRIBER_LINE”) (véanse las figuras D-10.2.7 y D-10.2.8).



Nota – Las señales prioritarias se indican con líneas más gruesas.

FIGURA D-10.2.7

Esquema secuencial. Interacción de servicios en el caso normal. Señales necesarias para el abonado A



Nota – Las señales prioritarias se indican con líneas más gruesas.

FIGURA D-10.2.8

Esquema secuencial. Interacción de servicios en el caso normal. Señales necesarias para el abonado B

En los cuatro diagramas de servicio que siguen (figuras D-10.2.9 a D-10.2.15)) se describe el comportamiento de cada servicio del proceso "LÍNEA_DE_ABONADO".

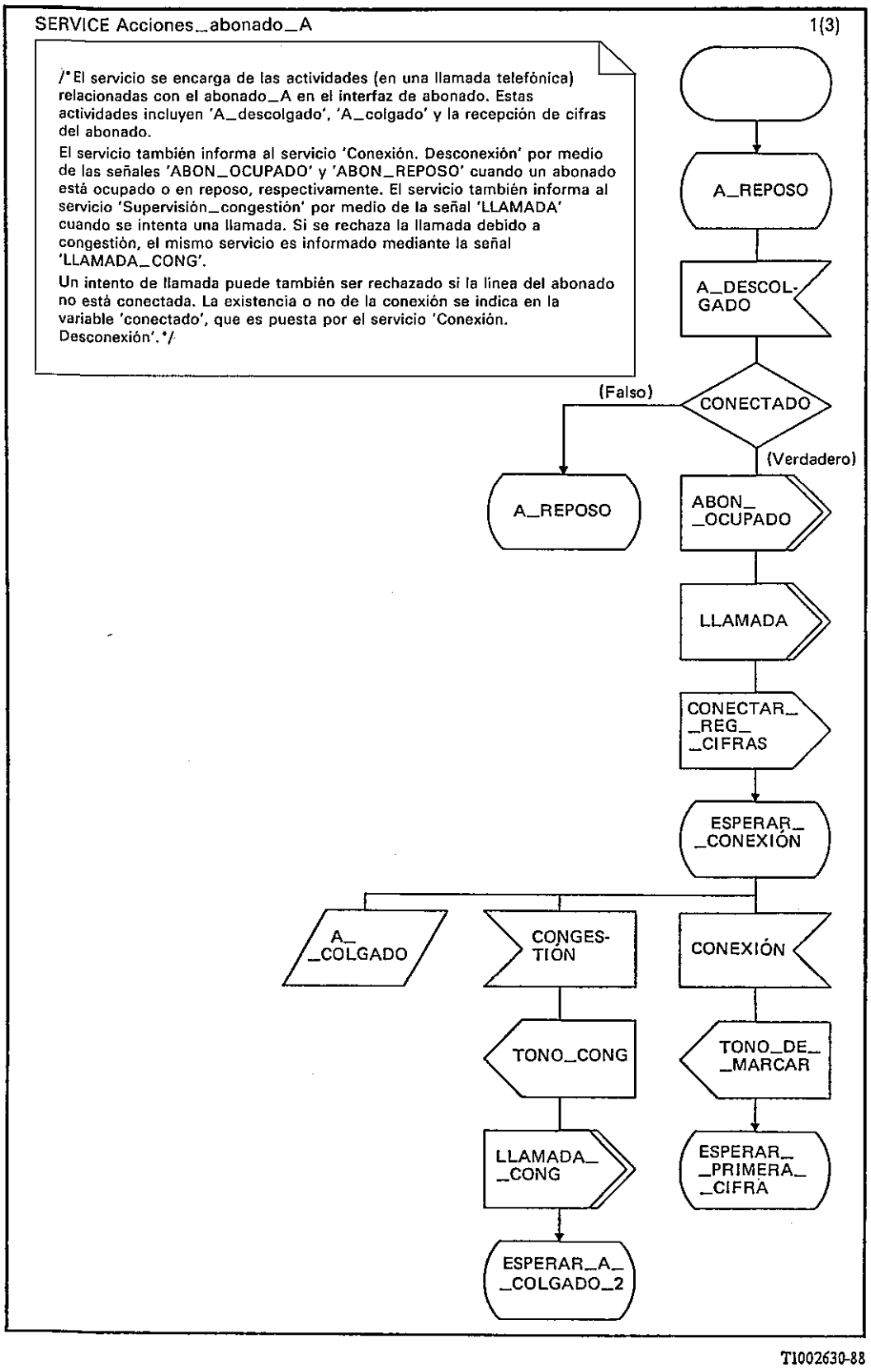


FIGURA D-10.2.9

Diagrama de servicio

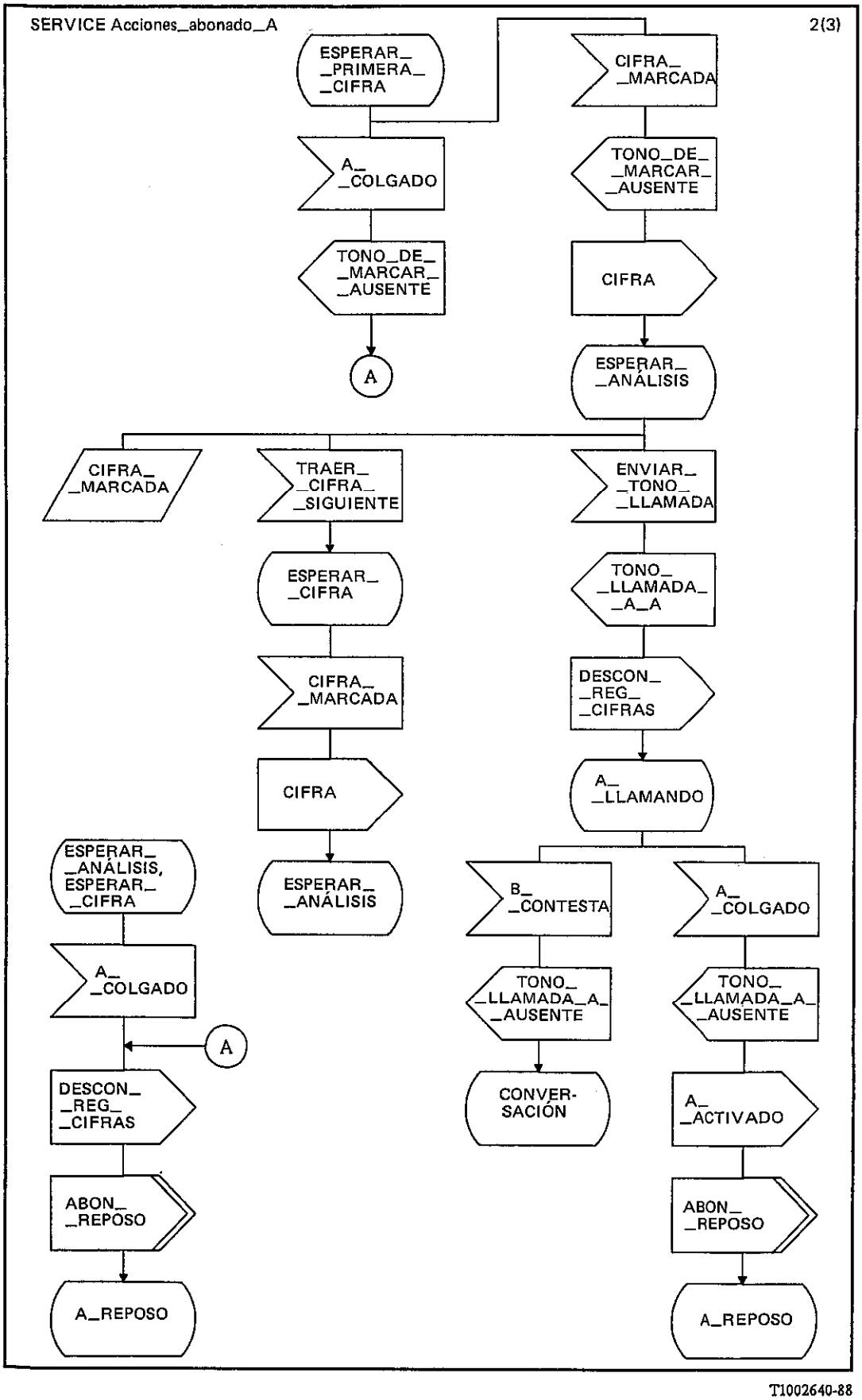


FIGURA D-10.2.10

Diagrama de servicio

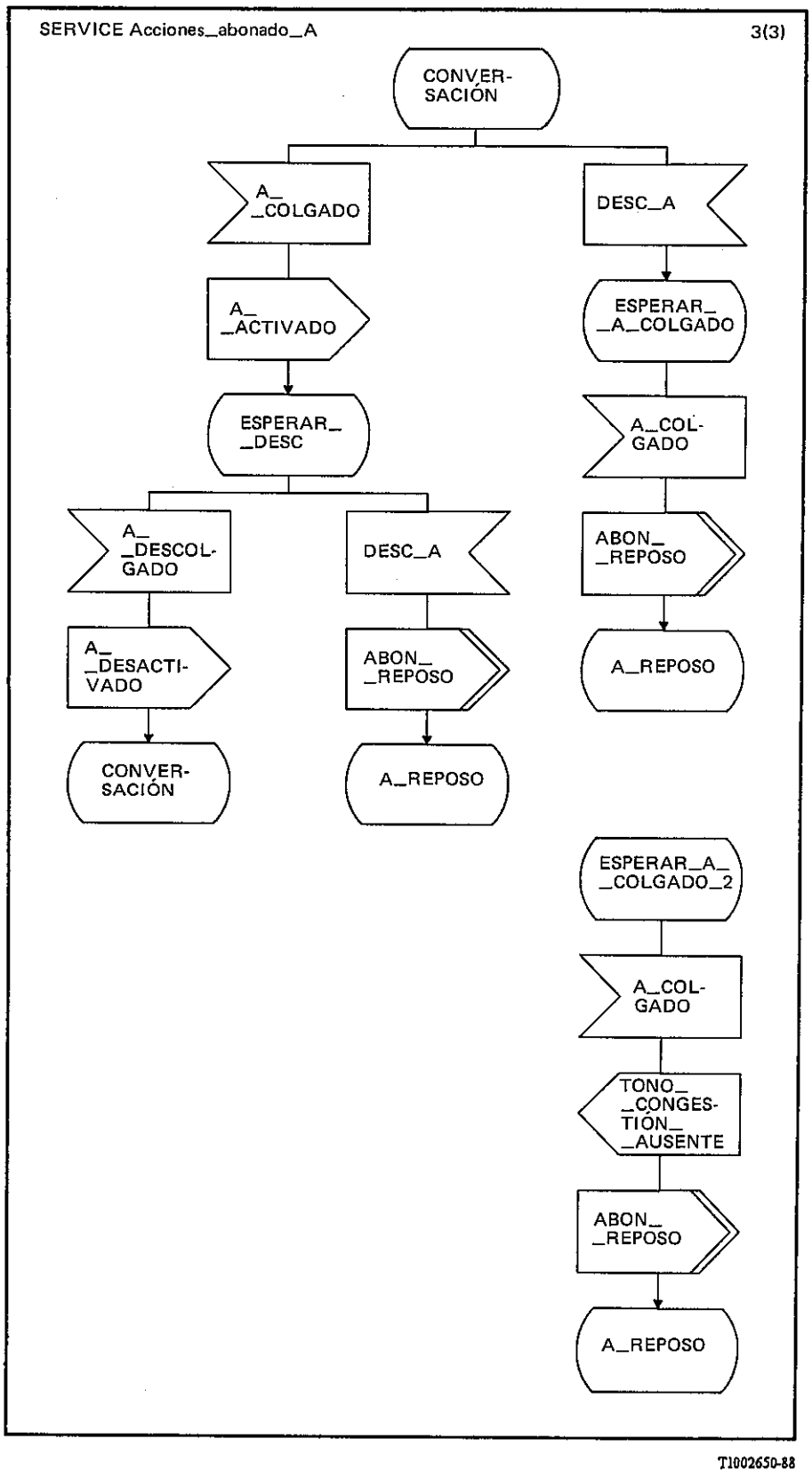


FIGURA D-10.2.11

Diagrama de servicio

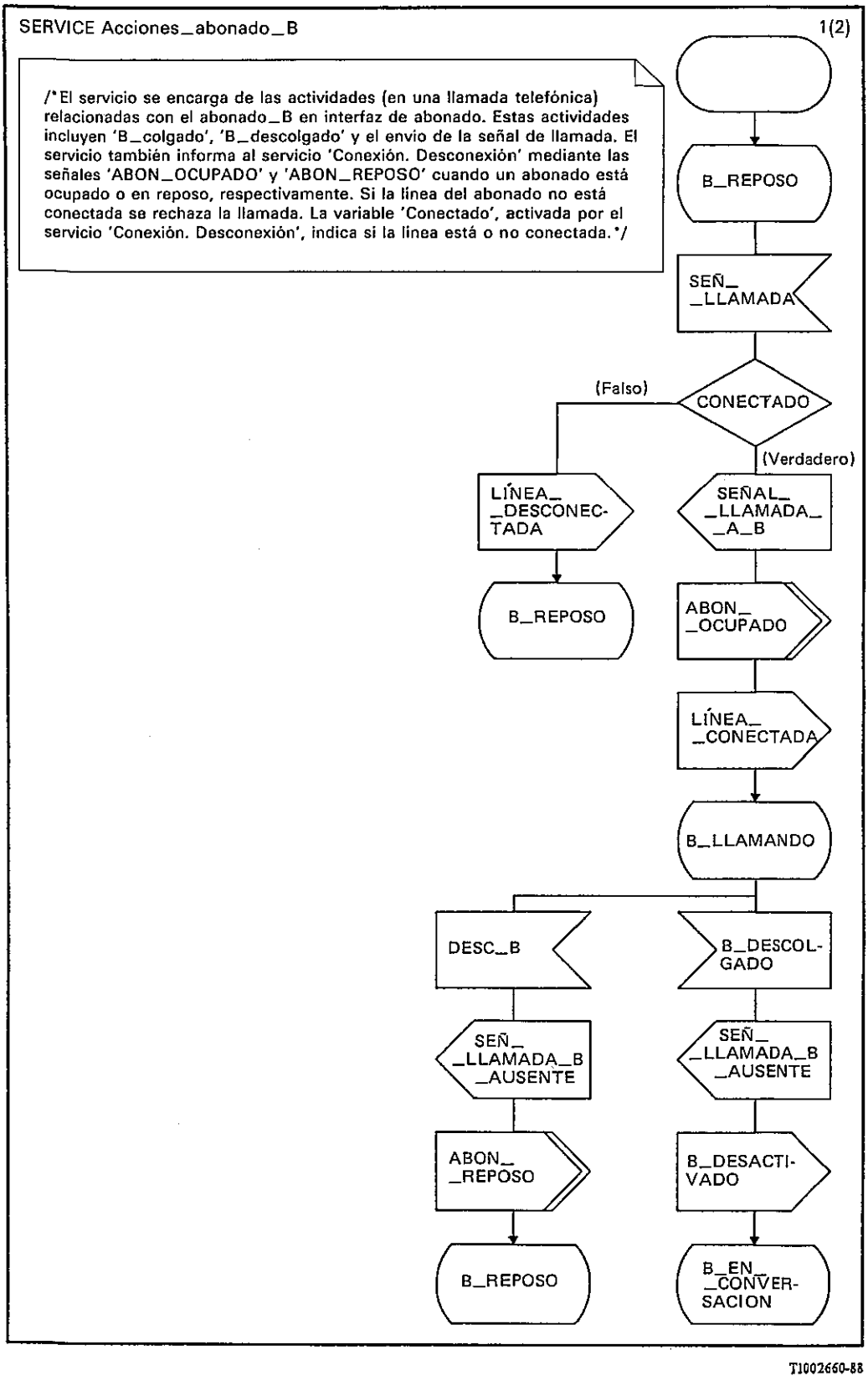


FIGURA D-10.2.12

Diagrama de servicio

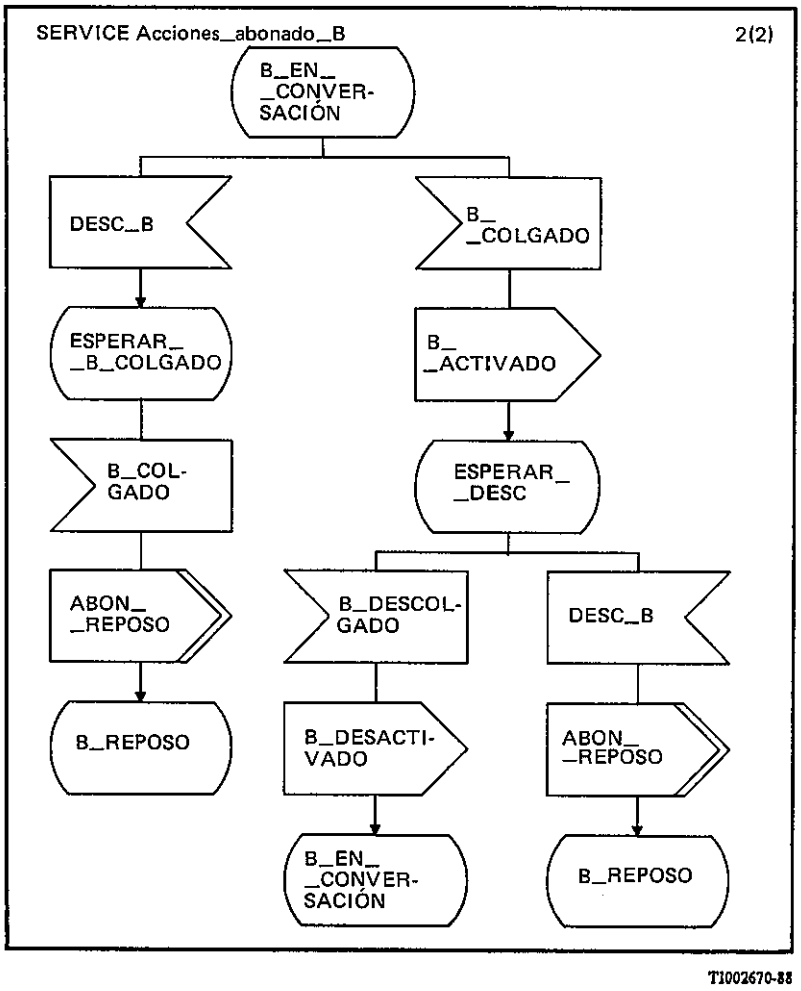


FIGURA D-10.2.13

Diagrama de servicio

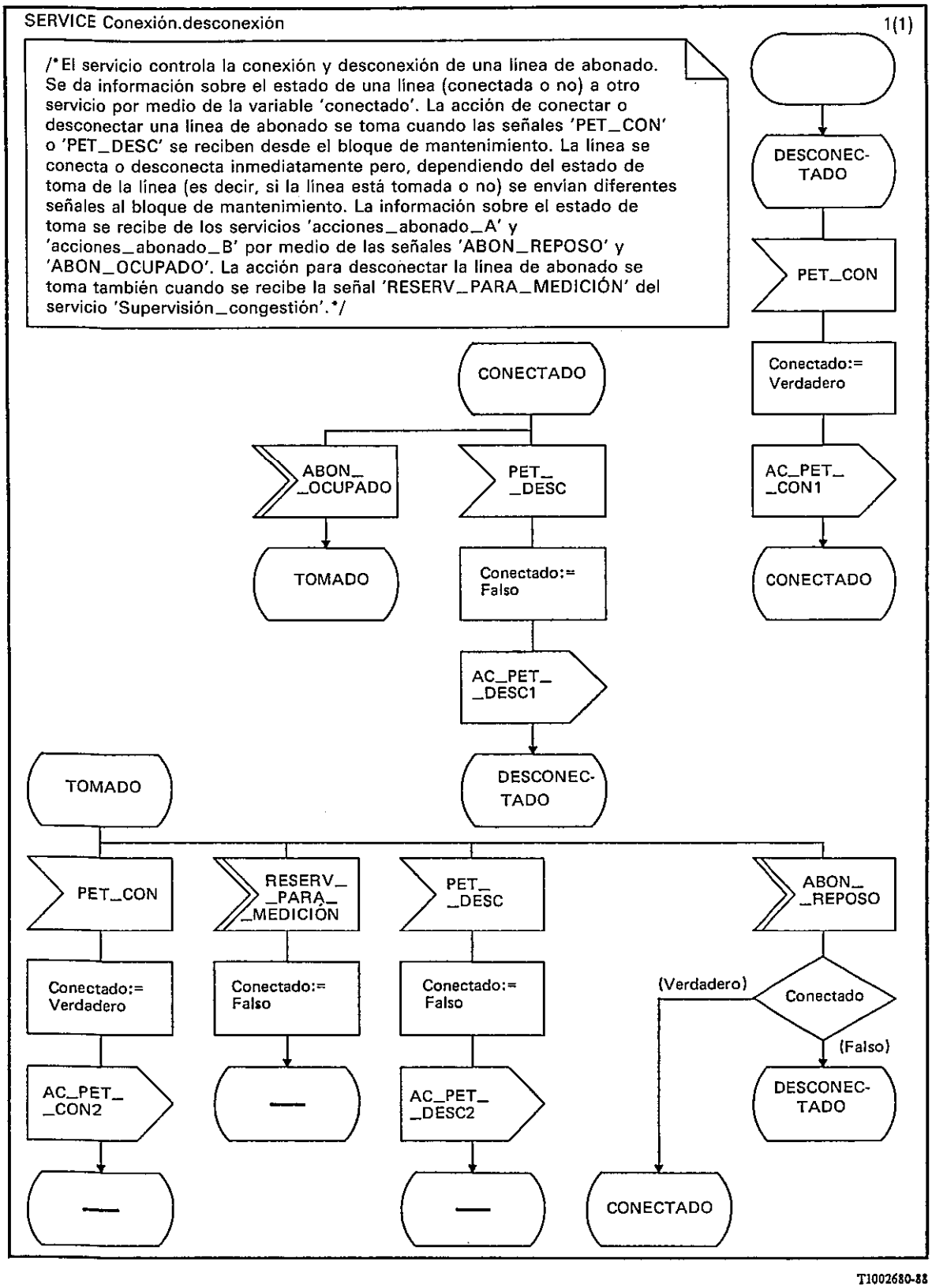


FIGURA D-10.2.14

Diagrama de servicio

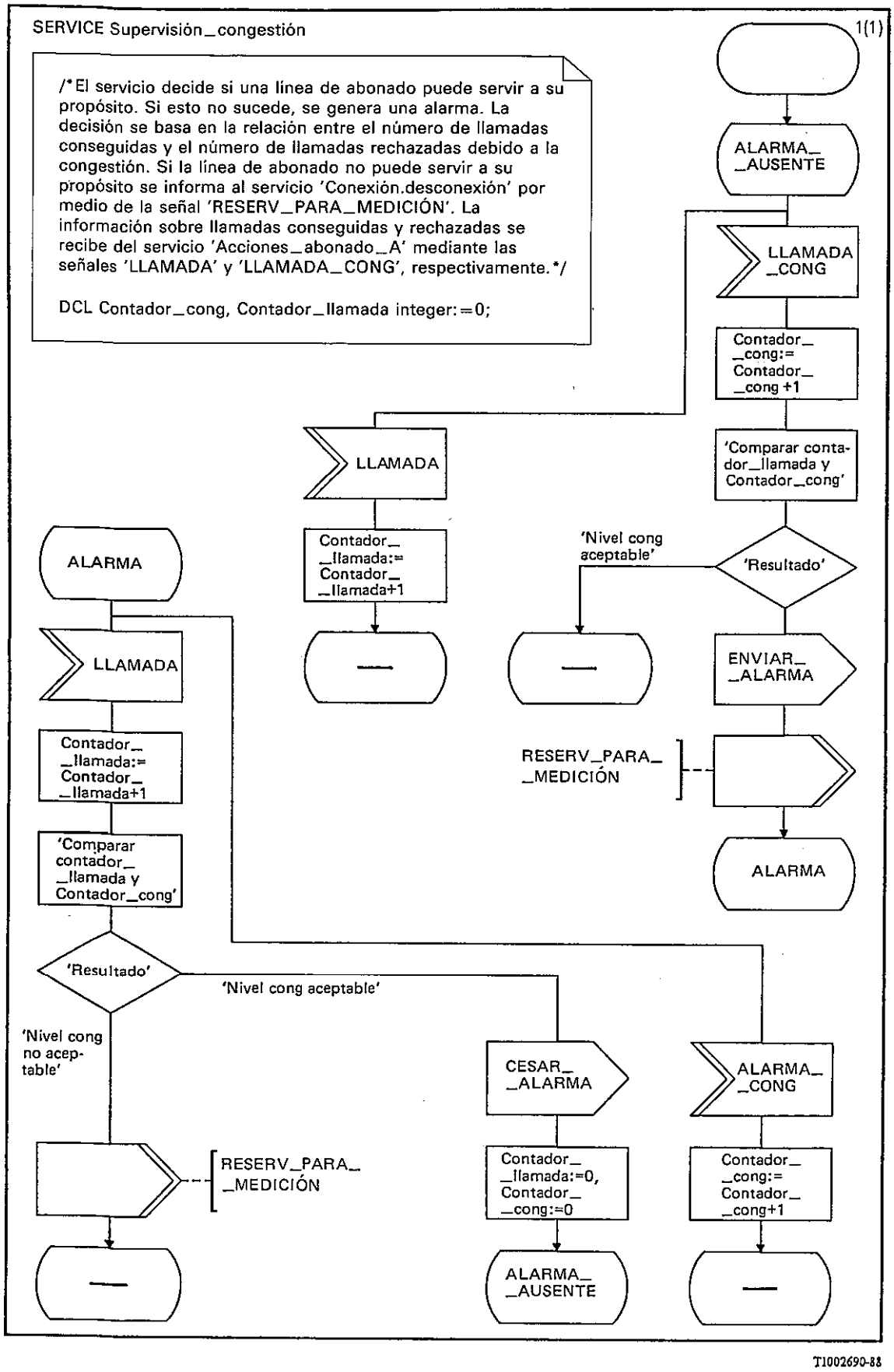


FIGURA D-10.2.15

Diagrama de servicio

D.11 *Instrumentos para el LED*

D.11.1 *Introducción*

En este punto se describe un conjunto de instrumentos que pueden facilitar la utilización del LED. Estos instrumentos se pueden aplicar a la generación de documentos, diagramas LED (LED/GR) o sentencias LED (LED/PR), y/o la validación de especificaciones LED.

Estas directrices no contienen la lista completa de los posibles instrumentos. Los instrumentos requeridos son función de la metodología escogida por el usuario.

En principio, el LED se puede utilizar sin ningún tipo de instrumento. Sin embargo, la complejidad propia de los sistemas modernos hace que las representaciones LED sean a menudo complicadas. Por consiguiente, se requieren instrumentos automáticos que faciliten la gestión de la especificación, diseño y documentación de muchos sistemas. Por ejemplo, la complejidad y el costo del trazado manual y la eventual actualización de la documentación gráfica relativa a una central de conmutación se podrían reducir considerablemente utilizando ayudas adecuadas.

Teniendo en cuenta estas consideraciones, se ha concebido el LED de forma que se puedan utilizar eficazmente instrumentos que faciliten su empleo.

D.11.2 *Clases de instrumentos*

Los instrumentos del LED se pueden clasificar según las actividades efectuadas durante la producción de documentación LED, por ejemplo:

- Instrumentos para entrada: según las formas sintácticas, existen ayudas a la entrada de gráficos, frases textuales y elementos pictográficos del LED.
- Instrumentos para verificación sintáctica: incluyen analizadores de sintaxis para cada una de las dos sintaxis.
- Instrumentos para generación de documentos: una vez que los documentos LED están almacenados en una máquina, los instrumentos pueden tener acceso a la misma y reproducirlos, utilizando posiblemente diversos periféricos. Éstos pueden emplear una forma sintáctica diferente de la utilizada para la entrada del documento. Además, los instrumentos pueden estar en condiciones de generar nuevos tipos de documentos derivados de los que se entraron originalmente.
- Instrumentos para modelado y análisis de sistemas: los documentos LED que representan un sistema se pueden utilizar para producir un modelo abstracto del sistema. Se pueden efectuar pruebas con dicho modelo, para la búsqueda de conflictos, la comparación entre diversos modelos de un mismo sistema (por ejemplo, entre una especificación y una descripción), para la ejecución de una simulación del comportamiento del sistema, etc.
- Instrumentos que pueden utilizarse para la generación de códigos: se pueden utilizar especificaciones LED muy detalladas para facilitar la programación del soporte lógico. Se pueden preparar instrumentos que permitan realizar secuencias de código orientadas semiautomáticas.

  La clase indicada a continuación constituye un tipo de instrumento específico pero útil:

- Instrumentos para la capacitación en LED: Estos instrumentos se pueden presentar por separado o integrados con otros. La integración permite facilitar ayuda, cuando es necesario, a los utilizadores de otras funciones.

Teniendo en cuenta que el LED se utiliza en muchas de las diversas fases del ciclo de vida útil de los sistemas, puede concebirse fácilmente una utilización para todos los tipos de instrumentos en el entorno de un proyecto integrado.

D.11.3 *Entrada de documentos*

Para la entrada del LED en forma de frases textuales no se necesitan requisitos especiales, ya que el LED/PR es equivalente, desde el punto de vista de la entrada, a la entrada de cualquier cadena de caracteres. Por consiguiente, se pueden utilizar los mismos instrumentos (editores de cadenas de caracteres). Sin embargo, la otra sintaxis supone una capacidad de tratamiento gráfico.

Es evidente que, mientras que la entrada LED/PR puede ser beneficiosa, en el caso de la entrada LED/GR es esencial si queremos utilizar dicha sintaxis como medio de entrada.

Se requiere siempre un editor gráfico para funciones como la conexión de dos símbolos, el desplazamiento de un conjunto de símbolos a otra parte de la página o a otras páginas y la supresión concatenada (la supresión de un símbolo implica la supresión de la conexión a ese símbolo). Como para los instrumentos LED/PR, los instrumentos de entrada LED/GR deben modelarse utilizando la semántica/sintaxis LED. Por lo tanto, deben hacer resaltar las conexiones inválidas e incitar al usuario a rellenar todas las partes incompletas, etc.

Los instrumentos están confrontados con diversos problemas que derivan de las limitaciones físicas de los dispositivos gráficos, tales como la «resolución». Es casi imposible disponer de un número suficiente de caracteres que sean legibles y además permitan la presentación de un número razonable de símbolos en la pantalla.

Si bien deben tenerse en cuenta las soluciones del tipo «ventana de ampliación» (zooming window) o de "desfile" (scrolling), las mismas no son completamente satisfactorias. La alta resolución no se tiene que considerar obligatoria si los diagramas son copiados por el usuario, pero es muy conveniente si los diagramas son producidos directamente por el usuario. Por el mismo motivo (requisito de una visión global y de cierta cantidad de detalles) la alta resolución es conveniente en la presentación de diagramas.

Los instrumentos para facilitar la entrada LED/PR pueden ser útiles: pueden sugerir al usuario las palabras clave LED/PR esperadas.

Pueden estructurar inmediatamente el LED/PR según las palabras clave recibidas, insertar automáticamente delimitadores, presentar al usuario teclas de función orientadas al LED/PR, proporcionar una disposición clara, etc.

La aplicación de esos instrumentos se puede basar en editores de cadenas de caracteres existentes, que pueden ampliarse incluyendo las características arriba mencionadas.

D.11.4 *Verificación de documentos*

Una vez que los documentos están almacenados en una máquina, la siguiente etapa es su verificación. En primer lugar se tienen que verificar individualmente y seguidamente combinar diagramas conexos y verificarlos hasta que se haya logrado la comprobación del sistema completo.

Si la entrada se ha efectuado mediante un instrumento basado en el LED, se puede ya haber efectuado una gran parte de la verificación de cada documento individual.

Todos los errores derivados de operaciones «no posibles» (por ejemplo, entradas o conservaciones que sigan a cualquier elemento que no sea un estado) deben detectarse y corregirse durante la fase de entrada. No obstante, la detección de ciertos errores sólo es posible una vez que se ha completado la fase de entrada, tanto en un solo documento como, desde luego, en el caso de incoherencias entre documentos.

Hay varias reglas LED que se pueden verificar automáticamente. Por ejemplo, el requisito de que todas las salidas tengan su correspondiente entrada.

En el caso de una especificación de múltiples niveles se puede verificar en cierta medida la coherencia entre niveles.

El modelo LED formal se puede utilizar para derivar un conjunto de procedimientos de verificación.

D.11.5 *Reproducción de documentos*

Los documentos LED almacenados en una máquina se tienen que poder recuperar, visualizar y reproducir. Se requieren instrumentos para todas estas actividades. Puede ser conveniente poder recuperar sólo una parte, o subconjuntos, de documentos. La recuperación puede estar basada en el LED, por ejemplo, «buscar todos los procesos que emiten» una señal determinada o bien «en qué estados» se efectúa una acción determinada, etc. Los instrumentos para la visualización de información son particularmente interesantes cuando la información se tiene que visualizar utilizando la sintaxis gráfica. Son pertinentes las mismas observaciones formuladas para la entrada de documentos en sintaxis LED/GR. La reproducción de documentos es función del tipo de documento que hay que reproducir, de la forma en que el documento se ha almacenado y de las características del periférico de salida. Puede depender asimismo de la forma en que se efectuó la entrada. Los usuarios pueden desear la salida en una sintaxis diferente de la utilizada para entrar el documento.

Las limitaciones de salida de los periféricos tienen repercusiones en la reproducción de documentos. Por ejemplo, un diagrama puede ser demasiado grande para caber en una superficie de papel determinada y, por consiguiente, tiene que dividirse en varias partes. Se tienen que agregar conectores y referencias. Puede ser conveniente establecer una diferencia entre una «adición» efectuada por el instrumento y las características de entrada originales. Es conveniente disponer de instrumentos con cierta flexibilidad en cuanto al formato de salida: estas características incluyen tamaños de símbolos diferentes, formatos de salida diferentes, representación vertical u horizontal, etc.

Siempre tiene que ser posible reproducir un documento exactamente en la misma forma en que se efectuó su entrada.

D.11.6 *Generación de documentos*

Partiendo de los documentos LED entrados por los usuarios y almacenados en la máquina, se pueden generar automáticamente otros varios documentos, entre los que figuran:

– las listas de señales, organizadas por proceso, por bloque o por sistema;

– los diagramas globales de estado, que muestran el gráfico de proceso como un conjunto de estados conectados por arcos que representan las transiciones;

– el cuadro de referencias, organizado por proceso, por bloque o por sistema;

– el diagrama de árbol de bloques, que muestra la estructura de los bloques y niveles;

– el comportamiento del sistema, como respuesta a secuencias de acciones del entorno;

– índices: los documentos generados se tendrán que reproducir, y por ello las mismas consideraciones antes expuestas son válidas.

Los documentos LED entrados en LED/GR se pueden traducir automáticamente a LED/PR y viceversa.

Es preciso tener en cuenta las consideraciones siguientes:

– el LED/GR contiene información visual que no puede traducirse a LED/PR (no existe en LED/PR). Por ejemplo, las coordenadas de los símbolos no tienen ningún significado en LED/PR;

– se pueden eliminar los conectores que enlazan líneas de flujo de hojas diferentes.

No obstante, la traducción inversa, de LED/PR a LED/GR, es mucho más compleja y es probable que no satisfaga completamente a todos los lectores potenciales.

Debido a la representación bidireccional del LED/GR, se pueden suprimir algunas etiquetas insertadas a causa de la estructura secuencial del PR, ya que basta una línea de conexión.

Por regla general, la traducción genera un modelo de los diagramas LED/GR. Este modelo contiene toda la información necesaria para que un instrumento pueda formar y reproducir el diagrama en un dispositivo gráfico.

Hay que tener en cuenta que dos instrumentos diferentes que efectúen la traducción de LED/PR a LED/GR pueden arrojar dos especificaciones LED/GR disputadas de manera diferente. Las especificaciones LED/GR obtenidas de esta forma son en ambos casos correctas, a condición de que mantengan la semántica expresada en la especificación original.

D.11.7 *Modelado y análisis de sistemas*

Los documentos LED, tanto si especifican como si describen un sistema, son esencialmente un modelo de dicho sistema.

Este modelo, cuya función principal es transferir información de una persona a otra, también puede interpretarse mediante instrumentos y verificarse en lo relativo a la coherencia y compleción (este aspecto puede no satisfacerse en casos de especificación que tengan por objeto especificar sólo algunas partes del sistema) así como en lo que atañe a la corrección y respeto de las reglas LED (en la forma descrita en el punto relativo a la verificación de documentos).

Además, pueden desarrollarse instrumentos que permitan utilizar el modelo para simular el comportamiento funcional de los sistemas. El simulador puede interactuar con el entorno y pueden formularse conclusiones sobre la respuesta del modelo a las expectativas de los usuarios.

Si se agrega información adicional para indicar el tiempo consumido para ejecutar cada acción y para dimensionar los recursos disponibles (colas de espera, instancias, etc.), la simulación permite también estudiar la capacidad del sistema.

Se pueden preparar instrumentos para crear un modelo del entorno, partiendo del modelo del sistema, para crear secuencias significativas que permitan verificar el sistema real. Los análisis de los trayectos pueden detectar conflictos del modelo.

El modelo del sistema también puede utilizarse como documentación en línea. Si existen enlaces apropiados entre el sistema real y la documentación almacenada, se puede desarrollar un instrumento que permita seguir en el modelo los eventos del sistema en tiempo real.

Para conseguirlo, debe existir una correlación entre los eventos físicos, vistos por el sistema, y los eventos lógicos que trata la documentación LED. Si la documentación está organizada en varios niveles de abstracción, el usuario puede escoger el nivel que desea seguir (rastrear). Esto puede ser muy útil pues permite que usuarios con conocimientos y capacitación diferentes inspeccionen las actividades del sistema.

Los instrumentos que interpretan el modelo LED pueden asimismo utilizarse para destacar diferencias de comportamiento de diferentes modelos del mismo sistema. Pueden además utilizarse para comparar descripciones de sistemas diferentes (sistemas producidos por empresas diferentes) o para comparar la especificación del sistema con su descripción. De esta manera es posible ver si una descripción de sistema se ajusta a la especificación original.

D.11.8 *Generación de códigos*

La existencia de una sintaxis definida formalmente y de una definición matemática formal del LED permite realizar instrumentos que puedan convertir la semántica de especificaciones LED en semántica de lenguajes de programación. Es posible que tales instrumentos no estén en condiciones de facilitar programas completos que puedan utilizarse, pero pueden ser muy útiles para proporcionar como mínimo el marco para un programa real.

El § D.9.1 de estas Directrices para el Usuario contiene un ejemplo de cómo puede realizarse la conversión entre LED y CHILL.

D.11.9 *Capacitación*

Se ha elaborado un curso de capacitación completo sobre el LED; abarca todos los aspectos del lenguaje y proporciona también ejemplos, así como algunas sugerencias sobre el uso del LED.

El curso sobre el LED puede solicitarse a la Sección de Ventas de la Secretaría General de la Unión Internacional de Telecomunicaciones – Place des Nations, CH-1211 Genève 20 (Suiza).

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Transmisiones de señales radiofónicas, de televisión y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos y comunicación entre sistemas abiertos
Serie Y	Infraestructura mundial de la información y aspectos del protocolo Internet
Serie Z	**Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación**

Impreso en Suiza
Ginebra, 2008