Reemplazada por una versión más reciente

UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Z.100

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

(03/93)

LENGUAJES DE PROGRAMACIÓN

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN DEL CCITT

Recomendación UIT-T Z.100
Reemplazada por una versión más reciente

(Anteriormente «Recomendación del CCITT»)

Reemplazada por una versión más reciente

PREFACIO

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

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

La Recomendación UIT-T Z.100, revisada por la Comisión de Estudio X (1988-1993) del UIT-T, fue aprobada por la CMNT (Helsinki, 1-12 de marzo de 1993).

NOTAS

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

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

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

Reemplazada por una versión más reciente

ÍNDICE

Página

RESUMEN

Alcance – Objetivo

En esta Recomendación se define el Lenguaje de Especificación y Descripción del CCITT (SDL, *specification and description language*) concebido para especificar y describir sin ambigüedades los sistemas de telecomunicaciones. El alcance del SDL se indica en 1.1.1. La presente Recomendación es un manual de referencia para este lenguaje.

Campo de aplicación

El SDL tiene conceptos para describir comportamientos y datos y estructurar grandes sistemas. La descripción del comportamiento se basa en máquinas de estados finitos ampliados que comunican mediante mensajes. La descripción de datos se basa en tipos de datos algebraicos. La estructuración se basa en la descomposición jerárquica y jerarquías de tipos. Estos fundamentos del SDL se explican en las respectivas cláusulas principales de la Recomendación. Una característica distintiva del SDL es la representación gráfica.

Aplicaciones

El SDL puede aplicarse en entidades de normalización y en la industria. Los principales sectores de aplicación, para los cuales se ha diseñado el SDL, se indican en 1.1.2, pero este lenguaje suele ser adecuado para describir sistemas reactivos.

Estado/Estabilidad

Esta Recomendación es el manual de referencia completo de este lenguaje, acompañado por directrices de utilización en el Apéndice I. El Anexo F contiene una definición formal de semánticas SDL.

Trabajo asociado

El texto principal está complementado por los anexos siguientes:

- A Indice de palabras clave no terminales
- B Glosario
- C Modelo algebraico inicial
- D Datos predefinidos del SDL
- E Reservado para uso futuro
- F Definición formal

y por los apéndices:

- I Directrices sobre la metodología SDL
- II Bibliografía SDL

En la Recomendación Q.65 se describe un método de utilización del SDL de acuerdo con ciertas normas. En la Recomendación Z.110 se indica una estrategia para introducir en las normas una técnica de descripción formal como el SDL. En el Apéndice III figuran referencias a otros documentos sobre el SDL, e información sobre la utilización del lenguaje en el sector industrial.

Antecedentes

Desde 1976 el CCITT ha recomendado varias versiones del SDL. Esta versión es una revisión de la Recomendación Z.100 de 1988 de la UIT.

En comparación con el SDL definido en 1988, la versión contenida en este documento se ha ampliado en lo tocante a la estructuración de objetos para tener en cuenta el modelado de sistemas orientados a objetos. Se han ampliado algunos detalles, a la vez que se ha procurado no invalidar los documentos existentes sobre el SDL-88. En 1.6 se dan detalles de los cambios introducidos.

Palabras clave:

descomposición jerárquica, especificación funcional, máquina de estados, orientación a objetos, presentación gráfica, técnica de descripción formal, técnica de especificación, tipos de datos abstractos.

Recomendación Z.100

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN DEL CCITT

(Melbourne, 1988; revisada en Helsinki, 1993)

1 Introducción al SDL

El texto de la cláusula 1 no es normativo; trata más bien de definir los convenios utilizados para describir el SDL. El uso del SDL en esta cláusula sólo es ilustrativo. Los metalenguajes y convenios sólo se introducen con objeto de describir el SDL inequívocamente.

1.1 Introducción

La finalidad de recomendar el SDL es proporcionar un lenguaje que permita una especificación y descripción inequívocas del comportamiento de sistemas de telecomunicaciones. Se tiene el propósito de que las especificaciones y descripciones escritas en SDL sean formales en el sentido de que sea posible analizarlas e interpretarlas inequívocamente.

Los términos especificación y descripción se usan con el significado siguiente:

a) una especificación de un sistema es la descripción de su comportamiento nominal, y

b) una descripción de un sistema es la descripción de su comportamiento real.

En sentido general, una especificación de sistema es la especificación del comportamiento y del conjunto de parámetros generales del sistema. Sin embargo el SDL tiene por objetivo especificar los aspectos relativos al comportamiento de un sistema; los parámetros generales que describen propiedades como la capacidad y el peso deben describirse mediante técnicas diferentes.

NOTA – Como no se hace una distinción entre el uso del SDL para especificación y su uso para descripción, el término especificación se utiliza en el texto que sigue tanto para el comportamiento nominal como para el comportamiento real.

1.1.1 Objetivos

Los objetivos generales al definir el SDL han sido proporcionar un lenguaje que:

a) sea fácil de aprender, utilizar e interpretar;

b) proporcione una especificación inequívoca, apropiada para pedidos y ofertas;

c) pueda ampliarse de modo que sea aplicable a nuevos desarrollos;

d) admita diversas metodologías de especificación y diseño de sistemas, sin presuponer la utilización de una metodología particular.

1.1.2 Aplicaciones

Esta Recomendación es el manual de referencia para el SDL. El Apéndice I contiene directrices metodológicas con ejemplos de utilización del SDL.

El ámbito principal de aplicación del SDL es la especificación del comportamiento de aspectos de sistemas que funcionan en tiempo real. Entre estas aplicaciones están:

a) procesamiento de llamadas (por ejemplo, tratamiento de llamadas, señalización telefónica, determinación del importe de las comunicaciones) en sistemas de conmutación;

b) mantenimiento y tratamiento de los fallos (por ejemplo, alarmas, eliminación automática de fallos, pruebas de rutina) en sistemas generales de telecomunicaciones;

c) control de sistemas (por ejemplo, control de sobrecarga, procedimientos de modificación y ampliación);

d) funciones de operación y mantenimiento, gestión de la red;

e) protocolos de comunicación de datos;

f) servicios de telecomunicaciones.

El SDL puede utilizarse, desde luego, para la especificación funcional del comportamiento de cualquier objeto que pueda especificarse utilizando un modelo discreto, por lo que ha de entenderse que el objeto comunica con su entorno por mensajes discretos.

El SDL es un lenguaje rico y puede utilizarse para especificaciones informales de alto nivel (y/o formalmente incompletas), para especificaciones semiformales y para especificaciones detalladas. El usuario debe elegir las partes apropiadas del SDL para el nivel deseado de comunicación y el entorno en que se utiliza el lenguaje. Según el entorno en que se utiliza una especificación, muchos aspectos pueden dejarse para que sean definidos de común acuerdo entre el origen y el destino de la especificación.

Así, el SDL puede utilizarse para producir:

a) requisitos de facilidades;

b) especificaciones de sistemas;

c) Recomendaciones del CCITT;

d) especificaciones de diseño de sistemas;

e) especificaciones detalladas;

f) descripciones de diseños de sistemas (tanto de alto nivel como detallado);

g) descripciones de pruebas de sistemas,

y la organización usuaria puede elegir el nivel apropiado de aplicación del SDL.

1.1.3 Especificación de sistema

Una especificación SDL define un comportamiento de sistema en forma de estímulo/respuesta, suponiendo que tanto los estímulos como las respuestas son discretos y dan información. En particular, una especificación de sistema se percibe como la secuencia de respuestas a cualquier secuencia dada de estímulos.

El modelo de especificación de sistema se basa en el concepto de máquinas de estados finitos ampliados, que comunican.

El SDL proporciona también conceptos estructurales que facilitan la especificación de sistemas grandes y/o complejos. Estos constructivos permiten la partición de la especificación de sistema en unidades manejables que pueden ser tratadas y comprendidas independientemente. La partición puede efectuarse en cierto número de pasos como resultado de los cuales se obtiene una estructura jerárquica de unidades que definen el sistema a diferentes niveles.

1.2 Gramáticas SDL

El SDL permite elegir entre dos formas sintácticas diferentes para representar un sistema: una representación gráfica (GR, *graphic representation*) (SDL/GR) y una representación con frases textuales (PR, *phrase representation*) (SDL/PR). Ambas formas son equivalentes, pues son representaciones concretas del mismo SDL. En particular, son equivalentes a una gramática abstracta, para los conceptos correspondientes.

Existe un subconjunto común al SDL/PR y al SDL/GR. Este subconjunto se denomina gramática textual común.

La Figura 1.1 muestra las relaciones entre SDL/PR, SDL/GR, las gramáticas concretas y la gramática abstracta.

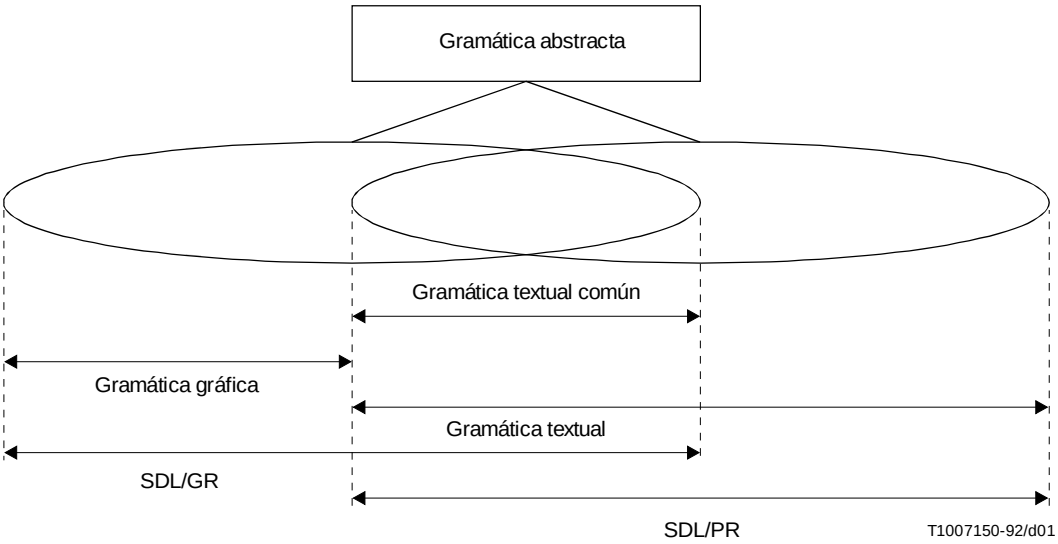


FIGURA 1.1/Z.100

Gramáticas SDL

Cada una de las gramáticas concretas tiene una definición de su propia sintaxis y de su relación con la gramática abstracta (es decir, una definición de cómo transformar a la sintaxis abstracta). Siguiendo este enfoque, sólo hay una definición de la semántica SDL; cada una de las gramáticas concretas heredará la semántica vía sus relaciones con la gramática abstracta. Este enfoque asegura también que SDL/PR y SDL/GR sean equivalentes.

Se proporciona también una definición formal del SDL que define cómo transformar una especificación de sistema a la sintaxis abstracta, y cómo interpretar una especificación dada en términos de la gramática abstracta. La definición formal figura en el Anexo F.

1.3 Definiciones básicas

En esta Recomendación se utilizan algunos conceptos generales y convenios cuyas definiciones se presentan a continuación.

1.3.1 Definición, tipo e instancia

En esta Recomendación, los conceptos de tipo y de instancia y sus relaciones son fundamentales. Se utilizan el esquema y la terminología definidos a continuación y representados en la Figura 1.2.

En esta subcláusula se presenta la semántica básica de definiciones de tipo, definiciones de instancia, definiciones de tipos parametrizados, parametrización, vinculación de parámetros de contexto, especialización e instanciación.

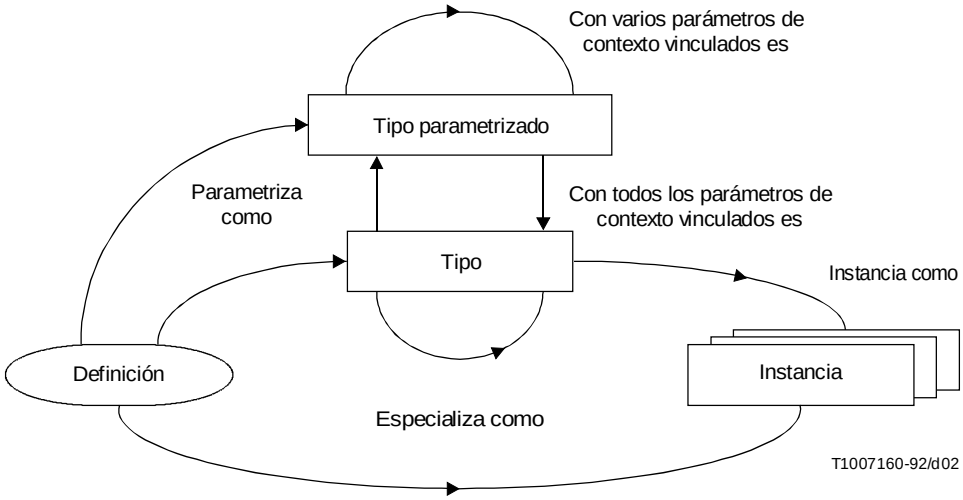


FIGURA 1.2/Z.100

Concepto de tipo

Las definiciones presentan entidades denominadas que son tipos o instancias. Una definición de un tipo define todas las propiedades asociadas con ese tipo. Un ejemplo de definición de instancia es una definición de variable. Un ejemplo de definición de tipo es una definición de señal.

Un tipo puede ser instanciado en varias instancias. Una instancia de un tipo particular tiene todas las propiedades definidas para ese tipo. Un ejemplo de tipo es un procedimiento, que puede ser instanciado por llamadas a procedimiento.

Un tipo parametrizado es un tipo en el cual varias entidades están representadas como parámetros de contexto formales. Un parámetro de contexto formal de una definición de tipo tiene una limitación. Las limitaciones permiten el análisis estático del tipo parametrizado. La vinculación de todos los parámetros de un tipo parametrizado da un tipo ordinario. Un ejemplo de tipo parametrizado es una definición de señal parametrizada en la que uno de los géneros transmitidos por la señal está especificado por un parámetro de contexto de género formal, lo que permite que el parámetro sea de géneros diferentes en contextos diferentes.

Una instancia se define directamente o mediante la instanciación de un tipo. Un ejemplo de instancia es una instancia de sistema que puede ser definida por una definición de sistema o ser una instanciación de un tipo de sistema.

La especialización permite basar un tipo, el subtipo, en otro tipo, su supertipo, añadiendo propiedades a las del supertipo o redefiniendo propiedades virtuales de ese supertipo. Una propiedad virtual puede estar limitada para permitir el análisis de tipos generales.

La vinculación de todos los parámetros de contexto de un tipo parametrizado da un tipo no parametrizado. No hay relación supertipo/subtipo entre un tipo parametrizado y el tipo no parametrizado que se deriva de él.

Tipo de datos es una clase especial de tipo (véanse 2.3 y 5).

NOTA – Para evitar textos recargados, el término *instancia* puede omitirse. Por ejemplo «un sistema se interpreta . . .» significa «una instancia de sistema se interpreta . . . ».

1.3.2 Entorno

Los sistemas especificados en SDL se comportan según los estímulos recibidos del mundo exterior. Este mundo exterior se denomina entorno del sistema que se especifica.

Se supone que hay una o más instancias de proceso en el entorno y, por lo tanto, las señales que pasan del entorno al sistema tienen asociadas identidades de estas instancias de proceso. Estos procesos tienen valores PId que pueden distinguirse de cualquier valor PId dentro del sistema (véase D.10).

Aunque el comportamiento del entorno es no determinista, se supone que obedece a las limitaciones impuestas por la especificación del sistema.

1.3.3 Errores

Una especificación de sistema es una especificación de sistema SDL válida, únicamente si satisface las reglas sintácticas y las condiciones estáticas de SDL.

Si al interpretar una especificación SDL válida se viola una condición dinámica, ocurre un error. Una interpretación de una especificación de sistema que conduce a un error significa que el comportamiento posterior del sistema no puede ser derivado de la especificación.

1.4 Estilo de presentación

1.4.1 División del texto

La Recomendación está organizada por temas descritos en una introducción opcional seguida por items de enumeración titulados sobre:

a) *Gramática abstracta* – Descrita por la sintaxis abstracta y las condiciones estáticas para la formación correcta.

b) *Gramática textual concreta* – Tanto la gramática textual común, utilizada para SDL/PR y SDL/GR, como la gramática utilizada solamente para SDL/PR. Esta gramática se describe por la sintaxis textual, las condiciones estáticas y las reglas de formación correcta para la sintaxis textual, y por la relación de la sintaxis textual con la sintaxis abstracta.

c) *Gramática gráfica concreta* – Descrita por la sintaxis gráfica, las condiciones estáticas y las reglas de formación correcta para la sintaxis gráfica, la relación de esta sintaxis con la sintaxis abstracta, y algunas reglas de dibujo adicionales (a las indicadas en 2.2.4).

d) *Semántica* – Da significado a un constructivo (elemento constructivo), establece sus propiedades, la forma en que se interpreta y las eventuales condiciones dinámicas que deben cumplirse para que el constructivo tenga un comportamiento correcto en el sentido SDL.

e) *Modelo* – Da la relación de correspondencia para notaciones taquigráficas expresadas en términos de constructivos de sintaxis concreta, estrictos, previamente definidos.

f) *Ejemplos.*

1.4.2 Items de enumeración titulados

Cuando un tema tiene una introducción seguida de un ítem de enumeración titulado, se considera que la introducción es una parte informal de la Recomendación, presentada solamente para facilitar la comprensión y no para completar la Recomendación.

Si un ítem de numeración titulado no tiene texto, se omite por completo.

La parte restante de esta subcláusula describe los otros formalismos especiales utilizados en cada ítem de enumeración titulado y los títulos utilizados. Puede considerarse también como un ejemplo de la disposición tipográfica de los items de enumeración titulados del primer nivel, definidos más arriba, donde este texto formaría parte de una sección de introducción.

Gramática abstracta

La notación de sintaxis abstracta se define en 1.5.1.

Si se omite el ítem de enumeración titulado *gramática abstracta*, no habrá entonces sintaxis abstracta adicional para el tema que se está presentando, y la sintaxis concreta corresponderá con la sintaxis abstracta definida por otra sección numerada de texto.

Se podrá hacer referencia a las reglas de la sintaxis abstracta a partir de cualesquiera items de enumeración titulados, utilizando el nombre de regla escrito en cursiva.

Las reglas en la notación formal pueden ir seguidas de subcláusulas que definen condiciones que deben ser satisfechas por una definición SDL formada correctamente y que pueden ser verificadas sin interpretación de una instancia. Las condiciones estáticas en este punto se refieren solamente a la sintaxis abstracta. Las condiciones estáticas que sólo son importantes para la sintaxis concreta se definen después de la sintaxis concreta. Las condiciones estáticas de la sintaxis abstracta, junto con la sintaxis abstracta, definen la gramática abstracta del lenguaje.

Gramática textual concreta

La sintaxis textual concreta se especifica en la forma Backus-Naur (BNF, backus-naur form) ampliada de la descripción de sintaxis definida en 2.1/Z.200 (véase también 1.5.2 de la présente Recomendación).

La sintaxis textual va seguida de subcláusulas que definen las condiciones estáticas que deben ser satisfechas en un texto formado correctamente y que deben ser verificadas sin interpretación de una instancia. Son también aplicables las condiciones estáticas (si existen) de la gramática abstracta.

En muchos casos hay una relación simple entre la sintaxis concreta y la abstracta, pues una regla de sintaxis concreta se representa simplemente por una sola regla en la sintaxis abstracta. Cuando el mismo nombre se utiliza en la sintaxis abstracta y en la concreta para representar el mismo concepto, el texto «<x> en la sintaxis concreta representa *X* en la sintaxis abstracta» está implícito en la descripción del lenguaje y suele omitirse. En este contexto no se distingue entre mayúsculas y minúsculas pero las subcategorías semánticas subrayadas son significativas.

La sintaxis textual concreta que no es una forma taquigráfica (sintaxis derivada, modelada por otros constructivos SDL) es sintaxis textual concreta estricta. La relación de una sintaxis textual concreta con una sintaxis abstracta está definida solamente para la sintaxis textual concreta estricta.

La relación entre sintaxis textual concreta y sintaxis abstracta se omite si el tema que se está definiendo es una forma taquigráfica que está modelada por otros constructivos SDL (véase *Modelo*, más adelante).

Gramática gráfica concreta

La sintaxis gráfica concreta se especifica en la forma Backus-Naur ampliada de descripción de sintaxis definida en 1.5.3.

La sintaxis gráfica va seguida de subcláusulas que definen las condiciones estáticas que deben cumplirse en SDL/GR formado correctamente y que pueden verificarse sin interpretación de una instancia. Son también aplicables las condiciones estáticas (si existen) de la gramática abstracta y las condiciones estáticas pertinentes de la gramática textual concreta.

La relación entre sintaxis gráfica concreta y sintaxis abstracta se omite si el tema que se está definiendo es una forma taquigráfica que está modelada por otros constructivos SDL (véase *Modelo*, más adelante).

En muchos casos hay una relación simple entre diagramas de gramática gráfica concreta y definiciones de sintaxis abstracta. Cuando el nombre de un símbolo no terminal comienza en la gramática concreta por la palabra «diagrama» y hay un nombre en la gramática abstracta que sólo difiere en que comienza por la palabra *definition*, las dos reglas representan el mismo concepto. Por ejemplo, <system diagram> en la gramática concreta y *System-definition* en la gramática abstracta son correspondientes.

La expansión en la sintaxis concreta, proveniente de facilidades tales como definiciones referenciadas (2.4.1.3), macros (4.2) y correspondencias de literales (5.3.1.15), etc., debe considerarse antes que la correspondencia entre la sintaxis concreta y la abstracta. Estas expansiones se describen detalladamente en 7.

Semántica

Las propiedades son relaciones entre conceptos diferentes en SDL. Se utilizan propiedades en las reglas de formación correcta.

Un ejemplo de propiedad es el conjunto de identificadores de señal de entrada válida de un proceso. Esta propiedad se utiliza en la condición estática «Para cada *State-node*, todos los *Signal-identifiers* de entrada (en el conjunto de señales de entrada válidas) aparecen, bien en un *Save-signalset*, bien en un *Input-node*».

Todas las instancias tienen una propiedad de identidad pero, a menos que esté formada de alguna manera poco usual, esta propiedad se determina como se define en la cláusula general sobre identidades en 2. Esto generalmente no se menciona como una propiedad de identidad. Además, no ha sido necesario mencionar subcomponentes de definiciones contenidos en la definición, pues la pertenencia de tales subcomponentes es evidente en la sintaxis abstracta. Por ejemplo, es evidente que una definición de bloque «tiene» encerradas definiciones de proceso y/o una definición de subestructura de bloque.

Una propiedad es estática si puede determinarse sin interpretación de una especificación de sistema SDL, y es dinámica si se requiere una interpretación de la misma para determinarla.

La interpretación se describe de una manera operacional. Toda lista en la sintaxis abstracta deberá interpretarse en el orden dado. Esto es, la Recomendación describe cómo se crean las instancias a partir de la definición de sistema y cómo estas instancias se interpretan en una «máquina SDL abstracta».

Son condiciones dinámicas aquéllas que deben cumplirse durante la interpretación y no pueden verificarse sin interpretación. Las condiciones dinámicas pueden conducir a errores (véase 1.3.3).

Modelo

Se considera que algunos constructivos son «sintaxis concretas derivadas» (o notaciones taquigráficas) de otros constructivos de sintaxis concreta equivalentes. Por ejemplo, omitir una entrada de una señal es sintaxis concreta derivada de una entrada de esa señal seguida de una transición nula que retorna al mismo estado.

Algunas veces esta «sintaxis concreta derivada», si fuese expandida, daría lugar a una representación extremadamente grande (quizá infinita). Sin embargo, la semántica de tal especificación puede ser determinada.

Ejemplos

El ítem de enumeración titulado *Ejemplo(s)* contiene ejemplo(s).

1.5 Metalenguajes

Para la definición de propiedades y sintaxis de SDL se han utilizado diferentes metalenguajes según las necesidades particulares.

A continuación se presenta una introducción a los metalenguajes utilizados; cuando procede, se hace referencia solamente a libros de texto o publicaciones específicas de la UIT.

1.5.1 Meta IV

Se utiliza el siguiente subconjunto de Meta IV para describir la sintaxis abstracta de SDL.

Una definición en la sintaxis abstracta puede considerarse como un objeto compuesto (un árbol) denominado que define un conjunto de subcomponentes.

Por ejemplo, la sintaxis abstracta para definición de visión es

View-definition :: *Variable-identifier*
Sort-reference-identifier

que define el dominio del objeto compuesto (árbol) denominado *View-definition*. Este objeto consiste en dos subcomponentes, que a su vez podrían ser árboles.

Process-identifier = *Identifier*

La definición Meta IV

expresa que un *Process-identifier* es un *Identifier*, por lo que no puede distinguirse sintácticamente de otros identificadores.

Un objeto podría también pertenecer a algunos dominios elementales (no compuestos). En el contexto de SDL, éstos son:

a) Objetos enteros

Ejemplo

Number-of-instances :: *Intg* [*Intg*]

Number-of-instances denota un dominio compuesto que contiene un valor entero (*Intg*) obligatorio y un entero opcional ([*Intg*]) que denotan respectivamente el número inicial y el número máximo opcional de instancias.

b) Objetos citación

Se representan por una secuencia cualquiera, en negrilla, de letras mayúsculas y dígitos.

Ejemplo

Destination = *Process-identifier* | *Service-identifier* | **ENVIRONMENT**

El *Destination* es un *Process-identifier*, un *Service-identifier* o el entorno que está denotado por la citación **ENVIRONMENT**.

c) Objetos testigo

Token denota el dominio de testigos. Puede considerarse que este dominio consiste en un conjunto potencialmente infinito de objetos atómicos distintos que no requieren una representación.

Ejemplo

Name :: *Token*

Un nombre consiste en un objeto atómico tal que cualquier *Name* puede distinguirse de cualquier otro nombre.

d) Objetos no especificados

Un objeto no especificado denota dominios que podrían tener alguna representación, pero esa representación no concierne a esta Recomendación.

Ejemplo

Informal-text :: ...

Informal-text contiene un texto que no es interpretado.

Los siguientes operadores (constructores) en BNF (véase 1.5.2) se utilizan también en la sintaxis abstracta: «*» para una lista que puede estar vacía, «+» para una lista no vacía, «|» para alternativa y «[» «]» para opcional.

Se utilizan paréntesis para la agrupación de dominios que están lógicamente relacionados.

Por último, la sintaxis abstracta utiliza otro operador postfijo «**-set**» que da un conjunto (colección no ordenada de objetos distintos).

Ejemplo:

Process-graph :: *Process-start-node State-node*-**set**

Un *Process-graph* consiste en un *Process-start-node* y un conjunto de *State-node*s.

1.5.2 BNF

En la forma Backus-Naur un símbolo terminal, bien se indica sin encerrarlo entre paréntesis angulares (es decir, el signo menor que y el signo mayor que, <y>), o bien es una de las dos representaciones <name> y <character string>. Obsérvese que los dos terminales especiales <name> y <character string> pueden tener también semánticas resaltadas como se define más adelante.

Los paréntesis angulares y la palabra o palabras encerradas son, bien un símbolo no terminal, bien uno de los dos símbolos terminales <character string> o <name>. Categorías sintácticas son los no terminales indicados por una o más palabras encerradas entre paréntesis angulares. Para cada símbolo no terminal se da una regla de producción sea en gramática textual concreta, sea en gramática gráfica. Por ejemplo,

<textual block reference> ::=
block <block name> **referenced** <end>

Una regla de producción para un símbolo no terminal consiste en el símbolo no terminal a la izquierda del símbolo ::=, y uno o más constructivos, que consisten en uno o más símbolos no terminales y/o terminales en el lado derecho. Por ejemplo <textual block reference>, <block name> y <end> en el ejemplo anterior son no terminales; **block** y **referenced** son símbolos terminales.

Algunas veces el símbolo incluye una parte subrayada. Esta parte subrayada resalta un aspecto semántico de ese símbolo. Por ejemplo <block name> es sintácticamente idéntico a <name>, pero semánticamente requiere que el identificador sea un nombre de bloque.

En el lado derecho del símbolo ::= puede haber varias producciones alternativas para el no terminal, separadas por barras verticales (|). Por ejemplo,

<block area> ::=
<block diagram>
| <existing typebased block definition>
| <graphical block reference>
| <graphical typebased block definition>

expresa que un <block area> es una <graphical block reference>, un <block diagram>, una <graphical typebased block definition> o una <existing typebased block definition>.

Los elementos sintácticos pueden agruparse utilizando llaves ({ y }), similares a los paréntesis en Meta IV (véase 1.5.1). Un grupo encerrado entre llaves puede contener una o más barras verticales, que indican elementos sintácticos alternativos. Por ejemplo,

<block interaction area> ::=
{<block area> | <channel definition area>}+

La repetición de grupos encerrados en llaves se indica por un asterisco (*) o signo más (+). Un asterisco indica que el grupo es opcional y puede repetirse cualquier número de veces; un signo más indica que el grupo tiene que estar presente y puede repetirse cualquier número de veces. El ejemplo anterior expresa que un <block interaction area> contiene por lo menos un <block area> o <channel definition area> y puede contener más <block area>s y <channel definition area>s.

Si se agrupan elementos sintácticos por medio de corchetes ([y]), el grupo es opcional. Por ejemplo,

<valid input signal set> ::=
signalset [<signal list>] <end>

expresa que un <valid input signal set> puede, pero no tiene necesariamente que, contener <signal list>.

1.5.3 Metalenguaje para gramática gráfica

Para la gramática gráfica el metalenguaje descrito en 1.5.2 se amplía con los siguientes metasímbolos:

a) ***contains***

b) ***is associated with***

c) ***is followed by***

d) ***is connected to***

e) ***set***

El metasímbolo ***set*** es un operador postfijo que actúa sobre los elementos sintácticos que le preceden inmediatamente, dentro de llaves, e indica un conjunto (no ordenado) de items. Cada ítem puede ser un grupo cualquiera de elementos sintácticos, en cuyo caso debe ser ampliado antes de aplicar el metasímbolo ***set***.

Ejemplo:

{{<system text area>}* {<macro diagram>}* <block interaction area>
{<type in system area>}* }***set***

es un conjunto de cero o varias <system text area>s, cero o varios <macro diagram>s, un <block interaction area> y cero o varias <type in system area>s.

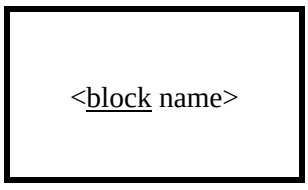
Todos los demás metasímbolos son operadores infijos, que tienen un símbolo gráfico no terminal como argumento de la izquierda. El argumento de la derecha es, bien un grupo de elementos sintácticos encerrados por llaves, bien un elemento sintáctico único. Si el lado derecho de una regla de producción tiene un símbolo gráfico no terminal como primer elemento y contiene uno o más de estos operadores infijos, el símbolo gráfico no terminal es el argumento de la izquierda de cada uno de estos operadores infijos. Un símbolo gráfico no terminal es un no terminal que termina con la palabra «symbol».

El metasímbolo ***contains*** indica que su argumento de la derecha debe situarse dentro de su argumento de la izquierda y del <text extension symbol> asociado, si existe. Por ejemplo,

<graphical block reference> ::=
<block symbol> ***contains*** <block name>

<block symbol> ::=

significa lo siguiente



El metasímbolo ***is associated with*** indica que su argumento de la derecha está lógicamente asociado a su argumento de la izquierda (como si estuviese «contenido» en ese argumento; la asociación inequívoca se asegura por reglas de dibujo adecuadas).

El metasímbolo ***is followed by*** significa que su argumento de la derecha sigue (tanto lógicamente como en el dibujo) a su argumento de la izquierda.

El metasímbolo ***is connected to*** significa que su argumento de la derecha está conectado (tanto lógicamente como en el dibujo) a su argumento de la izquierda.

1.6 Diferencias con respecto al SDL-88

El lenguaje definido en la presente Recomendación es una ampliación de la Recomendación Z.100 publicada en el *Libro Azul* de 1988. En esta subcláusula, el lenguaje definido en el *Libro Azul* se llama SDL-88 y el definido en la presente Recomendación SDL-92. Se ha hecho todo lo posible para que el SDL-92 sea una ampliación pura del SDL-88, sin invalidar la sintaxis ni cambiar la semántica de ningún uso actual del SDL-88. Además, las modificaciones sólo se han aceptado cuando las han considerado necesarias varias entidades miembros del CCITT.

Las principales ampliaciones se refieren esencialmente a la orientación a objetos. Si bien el SDL-88 está basado en objetos en su modelo subyacente, se han añadido algunos constructivos de lenguaje para que el SDL-92 sustente más completa y uniformemente el paradigma de objeto: (véanse 2.4.1.2 y 6):

a) paquetes (2.4.1.2);

b) tipos de sistemas, bloques, procesos, y servicios (6.1.1);

c) (conjunto de) instancias de sistemas, bloques, procesos y servicios basadas en tipos (6.1.2);

d) parametrización de tipos mediante parámetros de contexto (6.2);

e) especialización de tipos, y redefinición de tipos y transiciones virtuales (6.3).

Las otras ampliaciones son las siguientes: transición espontánea (2.6.6), elección no determinista (2.7.5), símbolo de entrada y salida internas en SDL/GR para compatibilidad con diagramas existentes (2.9), operador imperativo no determinista **any** (5.4.4.6), canal sin retardo (2.5.1), llamada a procedimiento remoto (4.14) y procedimiento de retorno de valor (5.4.5), entrada de campo de variable (2.6.4), definición de operador (5.3.2), combinación con descripciones de datos externos (5.4.6), capacidades de direccionamiento ampliadas en salida (2.7.4), acción libre en transición (2.6.7), transiciones continuas en el mismo estado con la misma prioridad (4.11), conexiones m:n de canales y rutas de señal en fronteras de estructura (2.5.3). Además, se ha introducido cierto grado de flexibilidad en la sintaxis.

En algunos casos ha sido necesario modificar el SDL-88. Los cambios sólo se han introducido cuando la definición del SDL-88 no era coherente. Los cambios y restricciones introducidos pueden salvarse mediante un procedimiento de traducción automática. Este procedimiento también es necesario si un documento SDL-88 contiene nombres consistentes en palabras que son palabras clave del SDL-92.

Para el constructivo **output** se ha simplificado la semántica, y ello puede invalidar alguna utilización especial de **output** (cuando no se da la cláusula **to** y existen varios trayectos posibles para la señal) en especificaciones SDL-88. Asimismo, se han cambiado algunas propiedades de la propiedad de igualdad de géneros.

Para el constructivo **export/import** se ha introducido una definición de variable remota optativa para armonizar la exportación de variables con la exportación de procedimientos (procedimiento remoto) introducida. Para ello deben modificarse los documentos SDL-88 que contienen calificadores en expresiones de importación o introducen varios nombres importados en el mismo campo con géneros diferentes. En los (pocos) casos en que es necesario calificar variables de importación para efectuar una resolución por contexto, la corrección consiste en introducir <remote variable definition>s y calificarlas con el identificador del nombre de variable remota introducido.

Para el constructivo **view**, la definición de visión corresponde localmente al proceso de visión o servicio. Esto exige modificar los documentos SDL-88 que contienen calificadores en definiciones de visión o en expresiones de visión. La corrección consiste en suprimir estos calificadores, pero no modificará la semántica de las expresiones de visión, ya que éstas son decididas por sus expresiones PId (que no cambian).

El constructivo **service** se ha definido como un concepto primitivo, en lugar de un concepto taquigráfico, sin ampliar sus propiedades. La utilización del servicio no es afectada por este cambio, pues se ha utilizado de todos modos como si fuera un concepto primitivo. El motivo del cambio es simplificar la definición del lenguaje y armonizarla con la utilización real, así como reducir el número de restricciones del servicio causadas por las reglas de transformación en el SDL-88. Como consecuencia de este cambio se ha suprimido el constructivo ruta de señales de servicio, y en su lugar pueden utilizarse rutas de señales. Este es sólo un pequeño cambio conceptual, y no tiene repercusiones sobre la utilización concreta (las sintaxis de ruta de señales de servicio SDL-88 y de ruta de señales SDL-92 son las mismas).

Reemplazada por una versión más reciente

El constructivo **priority output** se ha suprimido del lenguaje. Este constructivo puede ser sustituido por **output to self** con un procedimiento de traducción automática.

Puede parecer que algunas de las definiciones del SDL básico se han ampliado considerablemente, como por ejemplo, la definición de **signal**, pero cabe observar que las ampliaciones son optativas y sólo se deben utilizar para aprovechar la potencia aportada por las extensiones orientadas a objetos, por ejemplo, utilizar la parametrización y especialización para señales.

Las palabras clave del SDL-92 que no son palabras clave del SDL-88 son:

any, as, atleast, connection, endconnection, endoperator, endpackage, finalized, gate, interface, nodelay, noequality, none, package, redefined, remote, returns, this, use, virtual.

2 SDL básico

2.1 Introducción

Un sistema SDL consta de un conjunto de bloques. Los bloques están conectados entre sí y con el entorno por canales. Dentro de cada bloque hay uno o más procesos. Estos procesos comunican entre sí por señales y se supone que actúan concurrentemente.

La cláusula 2 se ha dividido en nueve temas principales:

a) *Reglas generales*

Conceptos básicos tales como reglas léxicas e identificadores, reglas de visibilidad, texto informal, partición de diagramas, reglas de dibujo, comentarios, ampliaciones de texto, símbolos de texto.

b) *Conceptos básicos de datos*

Conceptos de datos básicos tales como valores, variables, expresiones.

c) *Estructura de sistema*

Contiene conceptos referentes a los conceptos estructurales generales del lenguaje. Estos conceptos son sistema, bloque, proceso, procedimiento.

d) *Comunicación*

Contiene mecanismos de comunicación tales como canal, ruta de señales, señal.

e) *Comportamiento*

Los constructivos relevantes para el comportamiento de un proceso: reglas generales de conectividad de un gráfico de proceso o de procedimiento, definición de variable, arranque, estado, entrada, conservación, transición espontánea, etiqueta, transición.

f) *Acción*

Constructivos activos tales como tarea, crear proceso, llamada a procedimiento, salida, decisión.

g) *Temporizadores*

Definición de temporizador y primitivas de temporizador.

h) *Entrada y salida internas*

Notaciones taquigráficas para la compatibilidad con versiones anteriores del SDL.

i) *Ejemplos*

Ejemplos a los cuales se hace referencia en los demás temas.

2.2 Reglas generales

2.2.1 Reglas léxicas

Las reglas léxicas definen unidades léxicas. Las unidades léxicas son los símbolos terminales de la *sintaxis textual concreta*.

<lexical unit> ::=
<word>
| <character string>
| <special>
| <composite special>
| <note>
| <keyword>

<word> ::=
{<alphanumeric> | <full stop>}*
<alphanumeric>
{<alphanumeric> | <full stop>}*

<alphanumeric> ::=
<letter>
| <decimal digit>
| <national>

<letter> ::=
A | B | C | D | E | F | G | H | I | J | K | L | M
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z
| a | b | c | d | e | f | g | h | i | j | k | l | m
| n | o | p | q | r | s | t | u | v | w | x | y | z

<decimal digit> ::=
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<national> ::=
| ‘ | ¤ | @ | \
| <left square bracket>
| <right square bracket>
| <left curly bracket>
| <vertical line>
| <right curly bracket>
| <overline>
| <upward arrow head>

<left square bracket> ::= [

<right square bracket> ::=]

<left curly bracket> ::= {

<vertical line> ::= |

<right curly bracket> ::= }

<overline> ::= ~

<upward arrow head> ::= ^

<full stop> ::= .

<underline> ::= _

<character string> ::=
<apostrophe> {<alphanumeric>
| <other character> | <special>
| <full stop> | <underline>
| <space>
| <apostrophe><apostrophe>}* <apostrophe>

<apostrophe> <apostrophe> representa un <apostrophe> dentro de una <character string>.

<text> ::=
{ <alphanumeric>
| <other character>
| <special>
| <full stop>
| <underline>
| <space>
| <apostrophe> }*

<apostrophe> ::= ’

<other character> ::=
? | & | %

<special> ::=
+ | - | ! | / | > | * | (|) | " | , | ;
| < | = | :

<composite special> ::=
<< | >> | == | ==> | /= | <= | >= | // | := | =>
| -> | (. | .)

<note> ::=
/* <text> */

<keyword> ::=
active | **adding** | **all**
| **alternative** | **and** | **any**
| **as** | **atleast** | **axioms**
| **block** | **call** | **channel**
| **comment** | **connect** | **connection**
| **constant** | **constants** | **create**
| **dcl** | **decision** | **default**
| **else** | **endalternative** | **endblock**
| **endchannel** | **endconnection** | **enddecision**
| **endgenerator** | **endmacro** | **endnewtype**
| **endoperator** | **endpackage** | **endprocedure**
| **endprocess** | **endrefinement** | **endselect**
| **endservice** | **endstate** | **endsubstructure**
| **endsyntype** | **endsystem** | **env**
| **error** | **export** | **exported**
| **external** | **fi** | **finalized**
| **for** | **fpar** | **from**
| **gate** | **generator** | **if**
| **import** | **imported** | **in**
| **inherits** | **input** | **interface**
| **join** | **literal** | **literals**
| **macro** | **macrodefinition** | **macroid**
| **map** | **mod** | **nameclass**
| **newtype** | **nextstate** | **nodelay**
| **noequality** | **none** | **not**
| **now** | **offspring** | **operator**

operators	**or**	**ordering**
out	**output**	**package**
parent	**priority**	**procedure**
process	**provided**	**redefined**
referenced	**refinement**	**rem**
remote	**reset**	**return**
returns	**revealed**	**reverse**
save	**select**	**self**
sender	**service**	**set**
signal	**signallist**	**signalroute**
signalset	**spelling**	**start**
state	**stop**	**struct**
substructure	**synonym**	**syntype**
system	**task**	**then**
this	**timer**	**to**
type	**use**	**via**
view	**viewed**	**virtual**
with	**xor**	

Los caracteres <national> se han representado según figuran en la versión internacional de referencia del Alfabeto N.° 5 del CCITT (Recomendación T.50). La responsabilidad de la definición de las representaciones nacionales de estos caracteres incumbe a los organismos nacionales de normalización.

Los caracteres de control se definen en la Recomendación T.50. Una secuencia de caracteres de control puede aparecer cuando aparece un <space>, y tiene el mismo significado que <space>. El <space> representa el carácter del Alfabeto N.° 5 del CCITT para espacio.

Una ocurrencia de un carácter de control no es significativa en <informal text> ni en <note>. Un carácter de control no puede aparecer en literales de cadena de caracteres si su presencia es significativa. En esos casos debe emplearse el operador // y los literales para caracteres de control.

En toda <lexical unit>s salvo <character string>, <letter>s se trata siempre como si fuesen mayúsculas. (El tratamiento de <national>s puede ser definido por los organismos nacionales de normalización.)

Una <lexical unit> es terminada por el primer carácter que no puede formar parte de <lexical unit> de acuerdo con la sintaxis especificada anteriormente. Cuando un carácter <underline> va seguido de uno o más <space>s, todos esos caracteres (incluido <underline>) se pasan por alto; por ejemplo, A_B denota el mismo <name> que AB. Este uso de <underline> permite dividir <lexical unit>s en más de una línea.

Cuando el carácter / va seguido inmediatamente del carácter * fuera de una <note>, comienza una <note>. El carácter * inmediatamente seguido del carácter / en una <note> siempre termina la <note>. Una <note> puede ser insertada antes o después de cualquier <lexical unit>.

Dentro de <macro body> se aplican reglas léxicas especiales.

2.2.2 Reglas de visibilidad, nombres e identificadores

Gramática abstracta

Identifier	::	*Qualifier Name*
Qualifier	=	*Path-item +*
Path-item	=	*System-qualifier* |
		Block-qualifier |
		Block-substructure-qualifier |
		Signal-qualifier |
		Process-qualifier |
		Service-qualifier |
		Procedure-qualifier |
		Sort-qualifier

System-qualifier :: *System-name*
Block-qualifier :: *Block-name*
Block-substructure-qualifier :: *Block-substructure-name*
Process-qualifier :: *Process-name*
Service-qualifier :: *Service-name*
Procedure-qualifier :: *Procedure-name*
Signal-qualifier :: *Signal-name*
Sort-qualifier :: *Sort-name*
Name :: *Token*

Gramática textual concreta

<name>::=
<word> {<underline> <word> }*

<identifier> ::=
[<qualifier>] <name>

<qualifier> ::=
<path item> {/<path item>}*
| << <path item> {/<path item>}* >>

<path item> ::=
<scope unit kind> {<name> | <quoted operator> }

<scope unit kind> ::=
package
| **system type**
| **system**
| **block**
| **block type**
| **substructure**
| **process**
| **process type**
| **service**
| **service type**
| **procedure**
| **signal**
| **operator**
| **type**

Cuando un carácter <underline> va seguido por una <word> en un <name>, puede especificarse uno o más caracteres de control o espacios en lugar del carácter <underline>, siempre y cuando una de las <word>s que incluyen al carácter <underline> no constituyan una <keyword>; por ejemplo A B denota el mismo <name> que A_B. Esta regla no se aplica a la utilización de <underline> y <space> en <character string>.

Sin embargo, en algunos casos la ausencia de <underline> en <name>s es ambigua, por lo que se aplican las reglas siguientes:

1) Los <underline>s en un <name> en un <path item> deben especificarse explícitamente.
2) Cuando uno o más <name>s o <identifier>s pueden ir seguidos directamente por un <sort> (por ejemplo <variable definition>s, <view definition>s), deben especificarse explícitamente los <underline>s en esos <name>s o <identifier>s.
3) Cuando una <data definition> contiene <generator transformations>, los <underline>s en el <sort name> que siguen a la palabra clave **newtype** deben especificarse explícitamente.
4) Debe especificarse explícitamente <underline>s en <process name> de un <process context parameter> con <process constraint> <process identifier> solamente.

5) Debe especificarse explícitamente <underline>s en el <sort> en <procedure result>.

<quoted operator> sólo es aplicable cuando <scope unit kind> es operador; véase 5.3.2.

No hay sintaxis abstracta correspondiente para la <scope unit kind> denotada por paquete, tipo de sistema, tipo de bloque, tipo de proceso, tipo de servicio u operador.

El <qualifier> se refiere a un supertipo o refleja la estructura jerárquica desde el nivel de sistema o de paquete hasta el contexto definidor, y de tal modo que el nivel de sistema o de paquete es la parte textual más a la izquierda.

Se permite omitir algunos de los <path item>s más a la izquierda, o todo el <qualifier>. Cuando el <name> denota una entidad de la clase de entidad que contiene variables, sinónimos, literales y operadores (véase *Semántica* más adelante), la vinculación del <name> a una definición debe poder ser resuelta por el contexto efectivo. En otros casos el <identifier> está vinculado a una entidad que tiene su contexto de definición en la unidad de ámbito circundante más cercana en que el <qualifier> del <identifier> es el mismo que la parte más a la derecha del <qualifier> completo que denota esa unidad de ámbito. Si el <identifier> no contiene un <qualifier>, no es aplicable el requisito de concordancia de los <qualifier>s.

La calificación por supertipo hace visibles nombres de tipos virtuales en un supertipo que si no son ocultados por una redefinición en el subtipo (véase 6.3.2). Son los nombres en el supertipo que pueden ser calificados por supertipo.

Si puede interpretarse que un <qualifier> califica por un ámbito circundante y por un supertipo, denota un ámbito circundante.

Una subseñal tiene que estar calificada por su señal progenitora si existen otras señales visibles (sin parámetros de contexto) en ese lugar con el mismo <name>.

La resolución por el contexto es posible en los casos siguientes:

a) La unidad de ámbito en que se utiliza el <name> no es una <partial type definition> y contiene una definición que tiene ese <name>. El <name> estará vinculado a esa definición.

b) La unidad de ámbito en la cual se utiliza el <name> no contiene ninguna definición que tenga ese <name> o la unidad de ámbito es una <partial type definition>, y existe exactamente una definición visible de una entidad que tiene el mismo <name> y a la cual se puede vincular el <name> sin violar ninguna de las propiedades estáticas (compatibilidad de géneros, etc.) del constructivo en el que ocurre el <name>. El <name> estará vinculado a esa definición.

Sólo podrán utilizarse identificadores visibles, excepto el <identifier> utilizado en lugar de un <name> en una definición referenciada (es decir, una definición tomada de la <system definition>).

Semántica

Las unidades de ámbito se definen por el siguiente esquema:

Gramática textual concreta	*Gramática gráfica concreta*
<package definition>	<package diagram>
<textual system definition>	<system diagram>
<system type definition>	<system type diagram>
<block definition>	<block diagram>
<block type definition>	<block type diagram>
<process definition>	<process diagram>
<process type definition>	<process type diagram>
<service definition>	<service diagram>
<service type definition>	<service type diagram>
<procedure definition>	<procedure diagram>
<block substructure definition>	<block substructure diagram>
<channel substructure definition>	<channel substructure diagram>
<partial type definition>	
<operator definition>	<operator diagram>
<sort context parameter>	
<signal definition>	
<signal context parameter>	

Una unidad de ámbito tiene asociada una lista de definiciones, cada una de las cuales define una entidad que pertenece a cierta clase de entidad y tiene un nombre asociado. Están incluidas en la lista <gate definition>s, <formal context parameter>s, <formal parameters>s, <formal variable parameters> y definiciones de subestructura contenidas en la unidad de ámbito. Para una <partial type definition>, la lista asociada de definiciones consiste en las <operator signature>s, las <literal signature>s y cualesquiera <operator signature> y <literal signature>s heredadas de un género progenitor, de una instancia de generador, o indicadas por el uso de notaciones taquigráficas como <specialization> u <ordering>.

Aunque los <quoted operator>s, <operator name>s con una <exclamation> y <character string>s tienen su propia notación sintáctica, de hecho son <name>s y se representan en *sintaxis abstracta* con un *Name*. En lo sucesivo se tratarán como si también fuesen <name>s sintácticamente. Sin embargo, <state name>s, <connector name>s, <gate name>s que aparecen en rutas de señales y definiciones de canal, <generator formal name>s, <value identifier>s en ecuaciones, <macro formal name>s y <macro name>s tienen reglas de visibilidad especiales y por lo tanto no pueden ser calificados. <state name>s y <connector name>s no son visibles fuera de un solo <body>. En las subcláusulas apropiadas se explican otras reglas de visibilidad.

Se dice que cada entidad tiene su contexto definidor en la unidad de ámbito que la define. Las entidades son referenciadas por medio de <identifier>s.

El <qualifier> dentro de un <identifier> especifica unívocamente el contexto definidor del <name>.

Existen las siguientes especies de entidades:

a) paquetes;
b) sistema;
c) tipos de sistemas;
d) tipos de bloques;
e) bloques;
f) canales, rutas de señales, puertas;
g) subestructuras de bloque, subestructuras de canal;
h) señales, temporizadores;
i) tipos de procesos;
j) procesos;
k) tipos de servicios;
l) servicios;
m) procedimientos, procedimientos remotos;
n) variables (incluidos parámetros formales), sinónimos, literales, operadores;
o) variables remotas;
p) géneros;
q) generadores;
r) listas de señales;
s) visión.

Se dice que un <identifier> es visible en una unidad de ámbito

a) si la parte nombre del <identifier> tiene su contexto definidor en esa unidad de ámbito y el <identifier>
   1) no aparece en una <gate definition>, un <actual context parameter>, un <parameters of sort>, un <formal variable parameters> o una <specialization>, o
   2) denota un <formal context parameter>; o
b) si es visible en la unidad de ámbito que define esa unidad de ámbito; o
c) si la unidad de ámbito contiene una <partial type definition> en la que se define el <identifier>; o
d) si la unidad de ámbito contiene una <signal definition> en la que se define el <identifier>; o
e) si la unidad de ámbito tiene una <package reference clause> a través de la cual el <identifier> se hace visible (como se indica en 2.4.1.2).

Dos definiciones en la misma unidad de ámbito y pertenecientes a la misma clase de entidad no pueden tener el mismo <name>. La única excepción la constituyen operadores y definiciones en la misma <partial type definition> que pueden tener el mismo <name> con diferentes <argument sort>s o diferente género <result>.

En la gramática textual concreta, el nombre o identificador opcional en una definición después de la palabra clave de terminación (**endsystem, endblock**, etc.) debe ser sintácticamente el mismo que el nombre o identificador que sigue a la correspondiente palabra clave de comienzo (**system, block**, etc., respectivamente).

2.2.3 Texto informal

Gramática abstracta

Informal-text :: ...

Gramática textual concreta

<informal text> ::=
<character string>

Semántica

Si se utiliza texto informal en una especificación de sistema SDL, debe tenerse presente que este texto no es SDL formal, es decir, SDL no le da ninguna semántica. La semántica del texto informal puede definirse por otros medios.

2.2.4 Reglas de dibujo

El usuario podrá elegir el tamaño de los símbolos gráficos.

Los símbolos no deben superponerse ni cruzarse unos con otros. Una excepción a esta regla la constituyen los símbolos formados por líneas, por ejemplo <channel symbol>, <signal route symbol>, <create line symbol>, <flow line symbol>, <solid association symbol> y <dashed association symbol>, que pueden cruzarse unos con otros. El cruce de un símbolo con otro no conlleva una asociación lógica.

El metasímbolo ***is followed by*** implica un <flow line symbol>.

Los símbolos formados por líneas pueden consistir en uno o más segmentos de recta.

Un <flow line symbol> debe tener una flecha cuando entra en otro <flow line symbol>, en un <out-connector symbol> o en un <nextstate area>. En otros casos, las flechas son opcionales en los <flow line symbol>s. Los <flow line symbol>s son horizontales o verticales.

Se permiten imágenes especulares verticales de <input symbol>, <output symbol>, <internal input symbol>, <internal output symbol>, <priority input symbol>, <comment symbol> y <text extension symbol>.

El argumento de la derecha del metasímbolo ***is associated with*** debe estar más cerca del argumento de la izquierda que de cualquier otro símbolo gráfico. Los elementos sintácticos del argumento de la derecha deben distinguirse unos de otros.

El texto dentro de un símbolo gráfico debe leerse de izquierda a derecha, comenzando por la esquina superior izquierda. El borde derecho del símbolo se interpreta como un carácter de nueva línea, indicativo de que la lectura debe continuar en el punto situado más a la izquierda de la línea siguiente (si existe).

2.2.5 Partición de diagramas

La siguiente definición de partición de diagrama no forma parte de la *gramática gráfica concreta*, pero se utiliza el mismo metalenguaje.

<page> ::=
<frame symbol> ***contains***
{<heading area> <page number area>
{<syntactical unit>}*}

<heading area> ::=
<implicit text symbol> ***contains*** <heading>

<heading> ::=
<kernel heading> [<additional heading>]

<kernel heading> ::=
[<virtuality>] [**exported**]
<diagram kind> {<diagram name> | <diagram identifier>}

<page number area> ::=
<implicit text symbol> ***contains*** [<page number>
[**(**<number of pages>**)**]]

<page number> ::=
<literal name>

<number of pages> ::=
<Natural literal name>

La <page> es un no terminal de comienzo, por lo cual no se hace referencia al mismo en ninguna regla de producción. Un diagrama puede dividirse (particionarse) en varias <page>s, en cuyo caso el <frame symbol> que delimita el diagrama y el <heading> del diagrama se reemplazan por un <frame symbol> y un <heading> para cada <page>.

El usuario puede decidir que la frontera del medio en el que se reproducen los diagramas implique los <frame symbol>s.

El <implicit text symbol> no se muestra, pero está implícito para obtener una clara separación entre <heading area> y <page number area>. El <heading area> se sitúa en la esquina superior izquierda del <frame symbol>. El <page number area> se coloca en la esquina superior derecha del <frame symbol>. El <heading> y la <syntactical unit> dependen del tipo de diagrama.

El <additional heading> debe figurar en la primera página de un diagrama, pero es optativo en las páginas siguientes. <heading> y <diagram kind> se elaboran para los diagramas particulares en cada una de las subcláusulas de la Recomendación. <additional heading> no se define más detalladamente en la presente Recomendación.

<virtuality> denota la virtualidad del tipo definido por el diagrama (véase 6.3.2) y **exported** indica si un procedimiento es exportado como procedimiento remoto (véase 4.14).

2.2.6 Comentario

Un comentario es una notación para representar comentarios asociados con símbolos o texto.

En la *gramática textual concreta* se utilizan dos formas de comentarios. La primera es la <note>.

En las Figuras 2.10.1 a 2.10.4 se muestran ejemplos.

La sintaxis concreta de la segunda forma es:

<end> ::=
[<comment>] ;

<comment> ::=
comment <character string>

En la Figura 2.10.2 se muestra un ejemplo.

En la *gramática gráfica concreta* se utiliza la sintaxis siguiente:

<comment area> ::=
<comment symbol> ***contains*** <text>
is connected to <dashed association symbol>

<comment symbol> ::=

<dashed association symbol> ::=

- - - - - - - - - - -

Un extremo del <dashed association symbol> debe estar conectado al centro del segmento vertical del <comment symbol>.

Un <comment symbol> puede conectarse a cualquier símbolo gráfico por medio de un <dashed association symbol>. El <comment symbol> se considera un símbolo cerrado completando (en la imaginación) el rectángulo para encerrar el texto. Contiene texto de comentario relacionado con el símbolo gráfico.

<text> en *gramática gráfica concreta* corresponde a <character string> en *gramática textual concreta* sin tener <apostrophe>s.

En la Figura 2.10.4 se muestra un ejemplo.

2.2.7 Ampliación de texto

<text extension area> ::=
<text extension symbol> ***contains*** <text>
is connected to <solid association symbol>

<text extension symbol> ::=
<comment symbol>

<solid association symbol> ::=

_____

Un extremo del <solid association symbol> tiene que estar conectado con el centro del segmento vertical del <text extension symbol>.

Un <text extension symbol> puede conectarse con cualquier símbolo gráfico por medio de un <solid association symbol>. El <text extension symbol> se considera un símbolo cerrado completando (en la imaginación) el rectángulo.

El texto contenido en el <text extension symbol> es una continuación del texto dentro del símbolo gráfico y se considera contenido en ese símbolo.

2.2.8 Símbolo de texto

<text symbol> se utiliza en cualquier <diagram>. El contenido depende del diagrama.

<text symbol> ::=

2.3 Conceptos básicos de datos

El concepto de datos en SDL se define en 5; comprende la terminología de datos, los conceptos para definir nuevos géneros y los datos predefinidos.

2.3.1 Definiciones de tipos de datos

Los datos en SDL están principalmente relacionados con géneros. Un género define conjuntos de valores, un conjunto de operadores que puede aplicarse a estos valores y un conjunto de reglas algebraicas (ecuaciones) que definen el comportamiento de estos operadores cuando son aplicados a los valores. Los valores, los operadores y las reglas algebraicas definen colectivamente las propiedades del tipo de datos. Estas propiedades son definidas por definiciones de género.

SDL permite la definición de cualquier tipo de género que se necesite, incluidos mecanismos de composición (tipos compuestos), con la sola condición de que tal definición pueda especificarse formalmente. En cambio, a los efectos de lenguajes de programación hay consideraciones de implementación que requieren que el conjunto de géneros disponible y, en particular, el mecanismo de composición proporcionado (matriz, estructura, etc.), estén limitados.

2.3.2 Variable

Variables son objetos a los cuales puede asociarse un valor por asignación. Cuando se accede a la variable, se obtiene el valor asociado.

2.3.3 Valores y literales

Un conjunto de valores con ciertas características se denomina un género. Los operadores se definen para que actúen desde y hacia valores de géneros. Por ejemplo, la aplicación del operador de suma («+») desde y hacia valores del género entero es válida, en tanto que la suma de valores del género booleano no lo es.

Todo valor pertenece a un género, y solamente a uno; es decir, los géneros no pueden tener valores en común.

La mayor parte de los géneros tienen formas literales para denotar valores del género (por ejemplo, para enteros se utiliza «2» más bien que «1 + 1». Puede haber más de un literal para denotar el mismo valor (por ejemplo 12 y 012 pueden utilizarse para denotar el mismo valor entero). La misma denotación literal puede utilizarse para más de un género; por ejemplo 'A' es un carácter, pero es también una cadena de caracteres de longitud 1. Algunos géneros pueden no tener literales; por ejemplo, un valor compuesto a menudo no tiene literales propios pero sí valores definidos por operaciones de composición sobre valores de sus componentes.

2.3.4 Expresiones

Una expresión denota un valor. Si una expresión no contiene una variable o un operador imperativo, por ejemplo si es un literal de un género dado, toda ocurrencia de la expresión denotará siempre el mismo valor. Una expresión que contiene variables u operadores imperativos puede interpretarse como valores diferentes durante la interpretación de un sistema SDL según el valor asociado a las variables.

2.4 Estructura de sistema

2.4.1 Organización de especificaciones SDL

2.4.1.1 Marco

Una <sdl specification> puede describirse como una <system definition> monolítica o como una colección de <package>s y <referenced definition>s. Un <package> permite utilizar definiciones en diferentes contextos «empleando» el paquete en esos contextos, es decir, en sistemas o paquetes que pueden ser independientes. Una <referenced definition> es una definición que se ha suprimido de su contexto definidor para obtener una visión general dentro de una descripción de sistema. Es similar a una definición de macro (véase 4.2), pero está «insertada» exactamente en un lugar (el contexto definidor) empleando una referencia.

Gramática concreta

<sdl specification> ::=

<package list> [<system definition> {<referenced definition>}*]

2.4.1.2 Paquete

Para utilizar una definición de tipo en sistemas diferentes, tiene que estar definida como parte de un *package*.

Las definiciones como parte de un paquete definen tipos, generadores de datos, listas de señales, especificaciones remotas y sinónimos.

Las definiciones dentro de un paquete son visibles para un sistema u otros paquetes mediante una *package reference clause*.

Gramática textual concreta

<package list> ::=
{ <package> {<referenced definition>}* }*

<package> ::=
<package definition> | <package diagram>

<package definition> ::=
{<package reference clause>}*
package <package name>
[<interface>] <end>
{<entity in package>}*
endpackage [<package name>] <end>

<entity in package> ::=
<system type definition>
| <textual system type reference>
| <block type definition>
| <textual block type reference>
| <process type definition>
| <textual process type reference>
| <procedure definition>
| <remote procedure definition>
| <textual procedure reference>
| <signal definition>
| <signal list definition>
| <service type definition>
| <textual service type reference>
| <select definition>
| <remote variable definition>
| <data definition>
| <macro definition>

<package reference clause> ::=
use <package name> [/ <definition selection list>] <end>

<definition selection list> ::=
<definition selection> {, <definition selection>}*

<definition selection> ::=
[<entity kind>] <name>

<entity kind> ::=
system type
| **block type**
| **process type**
| **service type**
| **signal**
| **procedure**
| **newtype**
| **signallist**

| **generator**
| **synonym**
| **remote**

<interface> ::=
interface <definition selection list>

newtype se utiliza también para la selección de un nombre de sintipo en un paquete. La <entity kind> **remote** se utiliza para la selección de una definición de variable remota.

Gramática gráfica concreta

<package diagram> ::=
<package reference area>
is associated with
<frame symbol> ***contains***
{ <package heading>
{{<package text area>}*
{<diagram in package> }*} ***set*** }

<package reference area> ::=
<text symbol> ***contains*** {<package reference clause>}*

<package heading> ::=
package <package name> [<interface>]

<package text area> ::=
<text symbol> ***contains***
{ <entity in package>}*

<diagram in package> ::=
<system type diagram>
| <system type reference>
| <type in system area>
| <macro diagram>
| <option area>

La <package reference area> debe estar situada encima de <frame symbol>.

Semántica

Para cada <package name> mencionado en una <package reference clause>, debe existir un <package> correspondiente. Si este <package> no forma parte de una <package list>, debe existir un mecanismo para acceder al <package> referenciado, como si fuera una parte textual de <package list>. Este mecanismo no se define en la presente Recomendación.

Asimismo, si se omite <system definition> en una <sdl specification>, debe existir un mecanismo para utilizar los <package>s en la <package list> en otras <sdl specification>s. Antes de utilizar la <package list> en otras <sdl specification>s, se aplica el modelo para macros y definiciones referenciadas. En esta Recomendación no se define el mecanismo.

El nombre de una entidad definida dentro de un <package> es visible en otro <package> o la <system definition> solamente si:

1) el <package> o la <system definition> tiene una <package reference clause> que menciona el <package> y <package reference clause> tiene la <definition selection list> omitida o el nombre se menciona en una <definition selection>; y
2) el <package> que define el nombre tiene el <interface> omitido o el nombre se menciona en el <interface>.

La resolución por contexto en expresiones tendrá en cuenta también los géneros en paquetes que no son visibles en una <package reference clause>. Las señales que no son visibles en una cláusula **use** pueden formar parte de una lista de señales a través de un <signal list identifier> visible en una cláusula **use,** y estas señales se pueden afectar por lo tanto al conjunto completo de señales de entrada válidas de un proceso o servicio.

Si un nombre en una <definition selection> denota un <sort>, la <definition selection> denota también implícitamente todos los literales y operadores definidos para el <sort>, o <parent sort identifier> en caso de un sintipo.

Si un nombre en una <definition selection> denota una señal, la <definition selection> también denota implícitamente todas las subseñales de esa señal.

Los nombres que tienen su ocurrencia de definición en un <package> están referenciados mediante <identifier>s que tienen **package** <package name> como <path item> a la izquierda. No obstante, el <path item> puede omitirse si el <package name> denota el paquete que lo contiene, si (nombre, tipo de entidad) sólo está visible desde un paquete o si se aplica la resolución por contexto.

La <entity kind> en <definition selection> denota el tipo de entidad de <name>. Cualquier par de (<entity kind>, <name>) debe ser distinto dentro de una <definition selection list>. Para una <definition selection> en un <interface>, <entity kind> puede omitirse únicamente si no hay otro nombre que tiene su ocurrencia de definición directamente en el <package>. Para una <definition selection> en un <package reference clause>, puede omitirse <entity kind> sólo si exactamente se menciona una entidad de ese nombre en cualquier <definition selection list> para el paquete, o el paquete no tiene <definition selection list> y contiene directamente una definición única de ese nombre.

Una <system definition> y cada <package definition> tienen una <package reference clause> implícita:

use `Predefined`;

donde Predefined denota un paquete que contiene los datos predefinidos indicados en el Anexo D.

Modelo

Si un paquete se menciona en varias <package reference clause>s de una <package definition>, esto corresponde a una <package reference clause> que selecciona la unión de las definiciones seleccionada en las <package reference clause>s.

Una <sdl specification> con <system definition> y una <package list> no vacía corresponden a una <system definition>, donde

1) todas las ocurrencias del mismo nombre (incluida la ocurrencia de definición) de cualquier entidad definida dentro de un <package> se vuelven a denominar con el mismo nombre anónimo y único;

2) todas las definiciones dentro de los <package>s se han incluido en el nivel del sistema;

3) todas las ocurrencias del **package** <package name> en calificadores han sido sustituidas por **system** <system name>, donde <system name> es el nombre para <system definition>.

La transformación se detalla en 7. La relación entre <system definition> y *gramática abstracta* se define en 2.4.2.

2.4.1.3 Definición referenciada

Gramática concreta

<referenced definition> ::=

<definition> | <diagram>

<system definition> ::=

{<textual system definition> | <system diagram>}

<definition> ::=

<system type definition>
| <block definition>
| <block type definition>
| <process definition>
| <process type definition>
| <service definition>
| <service type definition>
| <procedure definition>
| <block substructure definition>
| <channel substructure definition>
| <macro definition>
| <operator definition>

<diagram> ::=

<system type diagram>
| <block diagram>
| <block type diagram>
| <process diagram>
| <process type diagram>
| <service diagram>
| <service type diagram>
| <procedure diagram>
| <block substructure diagram>
| <channel substructure diagram>
| <macro diagram>
| <operator diagram>

Para cada <referenced definition >, excepto <macro definition> y <macro diagram>, tiene que haber una referencia en el <package> o la <system definition> correspondiente.

Un <identifier> está presente en una <referenced definition> después de la(s) palabra(s) clave inicial(es). Para cada referencia debe existir una <referenced definition> con la misma clase de entidad que la referencia, y cuyo <qualifier>, si lo hay, denota un trayecto, desde una unidad de ámbito que contiene la referencia, hasta la referencia. Si dos <referenced definition>s de la misma clase de entidad tienen el mismo nombre, el <qualifier> de uno de los <identifier>s no debe constituir la parte izquierda del otro <qualifier>.

No se permite especificar un <qualifier> en el <identifier> después de la(s) palabra(s) clave inicial(es) para las definiciones que no son <referenced definition>s (es decir, se debe especificar un <name> para definiciones normales).

Semántica

Antes de que pueda derivarse las propiedades de una <system definition>, cada referencia es sustituida por la <referenced definition> correspondiente. En esta sustitución, el <identifier> de la <referenced definition> se sustituye por el <name> en la referencia.

Modelo

La relación entre <referenced definition> y la *gramática abstracta* se indica en 7.

2.4.2 Sistema

Gramática abstracta

System-definition	::	*System-name*
		Block-definition-**set**
		Channel-definition-**set**
		Signal-definition-**set**
		Data-type-definition
		Syntype-definition-**set**
System-name	=	*Name*

Una *System-definition* tiene un nombre que puede utilizarse en calificadores.

Una *System-definition* debe contener por lo menos una *Block-definition*.

Las definiciones de todos los canales, géneros y sintipos utilizados en el interfaz con el entorno y entre bloques del sistema están contenidas en la *System-definition*.

Gramática textual concreta

<textual system definition> ::=
{<package reference clause>}*
{ **system** <system name><end>
{<entity in system>}+
endsystem [<system name>] <end>
| <textual typebased system definition> }

<entity in system> ::=
<block definition>
| <textual block reference>
| <textual typebased block definition>
| <channel definition>
| <signal definition>
| <signal list definition>
| <select definition>
| <macro definition>
| <remote variable definition>
| <data definition>
| <textual block type reference>
| <block type definition>
| <process type definition>
| <textual process type reference>
| <procedure definition>
| <textual procedure reference>
| <remote procedure definition>
| <service type definition>
| <textual service type reference>

<textual block reference> ::=

block <block name> **referenced** <end>

Un ejemplo de <textual system definition> se muestra en la Figura 2.10.5.

Gramática gráfica concreta

<system diagram> ::=
[<package reference area>]
is associated with
{<frame symbol> ***contains***
{<system heading>
{{<system text area>}*
{<macro diagram>}*
<block interaction area>
{<type in system area>}*} ***set*** }
| <graphical typebased system definition>}

<frame symbol> ::=

<system heading> ::=
system <system name>

<system text area> ::=
<text symbol> ***contains***
{ <signal definition>
| <signal list definition>
| <remote variable definition>
| <data definition>
| <remote procedure definition>
| <macro definition>
| <select definition>}*

<block interaction area> ::=
{ <block area> | <channel definition area>}+

<block area> ::=
<graphical block reference>
| <block diagram>
| <graphical typebased block definition>
| <existing typebased block definition>

<graphical block reference> ::=
<block symbol> ***contains*** <block name>

<block symbol> ::=

<package reference area> debe estar situada encima del símbolo de recuadro de sistema.

Block-definition-**set** en la *gramática abstracta* corresponde a las <block area>s, y *Channel-definition*-**set** corresponde al <channel definition area>s.

En la Figura 2.10.6 se muestra un ejemplo de un <system diagram>.

Semántica

Una *System-definition* es la representación SDL de una especificación o descripción de un sistema.

Un sistema está separado de su entorno por la frontera del sistema y contiene un conjunto de bloques. La comunicación entre el sistema y el entorno o entre bloques dentro del sistema sólo puede efectuarse mediante señales. Dentro de un sistema, estas señales son transportadas por canales. Los canales conectan bloques entre sí o con la frontera del sistema.

Antes de interpretar una *System-definition* se elige un subconjunto consistente (véase 3.2.1). Este subconjunto se denomina una instancia de la *System-definition*. Una instancia de un sistema es una instanciación de un tipo de sistema definido por una *System-definition*. La interpretación de una instancia de una *System-definition* es efectuada por una máquina SDL abstracta que, al hacerlo, da semántica a los conceptos SDL. Interpretar una instancia de una *System-definition* es:

a) iniciar el tiempo del sistema;

b) interpretar los bloques y sus canales conectados que están contenidos en el subconjunto de partición consistente seleccionado.

2.4.3 Bloque

Gramática abstracta

Block-definition	::	*Block-name* *Process-definition*-**set** *Signal-definition*-**set** *Channel-to-route-connection*-**set** *Signal-route-definition*-**set** *Data-type-definition* *Syntype-definition*-**set** [*Block-substructure-definition*]
Block-name	=	*Name*

A menos que una *Block-definition* contenga una *Block-substructure-definition*, debe haber por lo menos una *Process-definition* dentro del bloque.

Es posible efectuar actividades de partición sobre los bloques que especifican *Block-substructure-definition*; esta característica del lenguaje se trata en 3.2.2.

Gramática textual concreta

<block definition> ::=
block {<block name>|<block identifier>} <end>
{ <channel to route connection> | <entity in block>}*
[<block substructure definition>
| <textual block substructure reference>]
endblock [<block name> | <block identifier>] <end>

<entity in block> ::=
<signal definition>
| <signal list definition>
| <process definition>
| <textual process reference>
| <textual typebased process definition>
| <signal route definition>
| <macro definition>
| <remote variable definition>
| <data definition>
| <select definition>
| <process type definition>
| <textual process type reference>

| <block type definition>
| <textual block type reference>
| <textual procedure reference>
| <procedure definition>
| <remote procedure definition>
| <service type definition>
| <textual service type reference>

<textual process reference> ::=
process <process name> [<number of process instances>] **referenced** <end>

En la Figura 2.10.7 se muestra un ejemplo de <block definition>.

Gramática gráfica concreta

<block diagram> ::=
<frame symbol>
contains {<block heading>
{ {<block text area>}* {<macro diagram>}*
{ <type in block area> }*
[<process interaction area>] [<block substructure area>]}***set*** }
is associated with {<channel identifiers>}*

Los <channel identifiers> identifican canales conectados a rutas de señales en el <block diagram>. Los identificadores de canal están colocados fuera del <frame symbol>, próximo al punto extremo de la ruta de señales en el <frame symbol>. Si el <block diagram> no contiene una <process interaction area>, deberá contener una <block substructure area>.

<block heading> ::=
block {<block name> | <block identifier> }

<block text area> ::=
<system text area>

<process interaction area> ::=
{ <process area>
| <create line area>
| <signal route definition area>}+

<process area> ::=
<graphical process reference>
| <process diagram>
| <graphical typebased process definition>
| <existing typebased process definition>

<graphical process reference> ::=
<process symbol> ***contains***
{<process name> [<number of process instances>]}

<process symbol> ::=

<create line area> ::=
<create line symbol>
is connected to {<process area> <process area>}

<create line symbol> ::=
– ——

La punta de flecha en el <create line symbol> indica la <process area> de un proceso en el cual se efectúa una acción de crear. Los <create line symbol>s son optativos, pero si se utilizan, en el proceso en el extremo de origen del <create line symbol> debe haber una petición de crear para el proceso en la punta de flecha del <create line symbol>. Esta regla se aplica después de la transformación de <option area>.

NOTA – Esta regla puede aplicarse indistintamente antes o después de la transformación de <transition option>.

Si una <block definition> o una <block type definition>, que se utiliza en una <textual typebased block definition>, contiene <signal route definition>s y <textual typebased process definition>s, cada puerta de las <process type definition>s de las <textual typebased process definition>s debe estar conectada a una ruta de señales.

Esta misma regla se aplica a <process definition>s y <process type definition>s que contienen <signal route definition>s y <textual typebased service definition>s.

En la Figura 2.10.8 se muestra un ejemplo de <block diagram>.

Semántica

Una definición de bloque es un contenedor para una o más definiciones de proceso de un sistema y/o una subestructura de bloque.

Un bloque proporciona una interfaz de comunicación estática en virtud de la cual sus procesos pueden comunicar con otros procesos. Además, establece el ámbito para definiciones de proceso.

Interpretar un bloque es crear las instancias de procesos iniciales en el bloque.

2.4.4 Proceso

Una definición de proceso define un conjunto arbitrariamente grande de instancias de proceso.

Gramática abstracta

Process-definition	::	*Process-name* *Number-of-instances* *Process-formal-parameter* ***** *Procedure-definition*-**set** *Signal-definition*-**set** *Data-type-definition* *Syntype-definition*-**set** *Variable-definition*-**set** *View-definition*-**set** *Timer-definition*-**set** *Process-graph* | *Service-decomposition*
Number-of-instances	::	*Intg* [*Intg*]
Process-name	=	*Name*
Process-graph	::	*Process-start-node* *State-node*-**set**
Process-formal-parameter	::	*Variable-name* *Sort-reference-identifier*
Service-decomposition	::	*Service-definition*-**set** *Signal-route-definition*-**set** *Signal-route-to-route-connection*-**set**

Si *Process-definition* tiene una *Service-decomposition* no debe haber *Timer-definitions*. *Service-decomposition* debe contener por lo menos una *Service-definition*.

Gramática textual concreta

<process definition> ::=
process {<process name> | <process identifier> }
[<number of process instances>] <end>
[<formal parameters> <end>] [<valid input signal set>]
{ <entity in process>
| <signal route to route connection> }*
[<process body>]
endprocess [<process name> | <process identifier>] <end>

<entity in process> ::=
<signal definition>
| <signal list definition>
| <textual procedure reference>
| <procedure definition>
| <remote procedure definition>
| <imported procedure specification>
| <macro definition>
| <remote variable definition>
| <data definition>
| <variable definition>
| <view definition>
| <select definition>
| <imported variable specification>
| <timer definition>
| <signal route definition>
| <service definition>
| <textual service reference>
| <textual typebased service definition>
| <service type definition>
| <textual service type reference>

<textual procedure reference> ::=
<procedure preamble>
procedure <procedure name> **referenced** <end>

<textual service reference> ::=
service <service name> **referenced** <end>

<valid input signal set> ::=
signalset [<signal list>] <end>

<process body> ::=
<start> {<state> | <free action>} *

<formal parameters> ::=
fpar <parameters of sort> {**,** <parameters of sort>}*

<parameters of sort> ::=
<variable name> {**,** <variable name>}* <sort>

<number of process instances> ::=
([<initial number>][,[<maximum number>]])

<initial number> ::=
<Natural simple expression>

<maximum number> ::=
<Natural simple expression>

<service definition>, <textual service reference> o <textual typebased service definition> sólo pueden estar presentes si se omite <process body>.

Una <process definition> puede contener <signal route definition>s solamente si la <block definition> o la <textual typebased block definition> que la engloba contiene <signal route definition>s.

El número inicial de instancias y el número máximo de instancias contenidos en *Number-of-instances* se derivan de <number of process instances>. Si <initial number> se omite, el <initial number> es 1. Si <maximum number> se omite, entonces el <maximum number> no está acotado.

El <number of process instances> utilizado en la derivación es el siguiente:

a) Si no hay <textual process reference> para el proceso, se utiliza el <number of process instances> en la <process definition> o en la <textual typebased process definition>. Si no contiene un <number of process instances>, se utiliza el <number of process instances> en que se omiten el <initial number> y el <maximum number>.

b) Si se omite el <number of process instances> en <process definition> y el <number of process instances> en una <textual process reference>, se utiliza el <number of process instances> en que se omiten el <initial number> y el <maximum number>.

c) Si se omite el <number of process instances> en <process definition> o el <number of process instances> en una <textual process reference>, el <number of process instances> es el que está presente.

d) Si se especifica el <number of process instances> en <process definition> y el <number of process instances> en una <textual process reference>, los dos <number of process instances> deben ser léxicamente iguales, y se utiliza ese <number of process instances>.

Se aplica una relación similar para el <number of process instances> especificado en el <process diagram> y en la <graphical process reference>, como se define a continuación.

El <initial number> de instancias tiene que ser inferior o igual al <maximum number> y el <maximum number> tiene que ser mayor que cero.

La utilización de <valid input signal set> se define en 2.5.2, *Modelo*.

Un ejemplo de <process definition> se muestra en la Figura 2.10.9.

Gramática gráfica concreta

<process diagram> ::=

<frame symbol>

contains {<process heading>

{ {<process text area>}*

{<macro diagram>}*

{<type in process area>}*

{<process graph area> | <service interaction area> } }***set*** }

[***is associated with*** {<signal route identifiers>}*]

Los <signal route identifier>s identifican rutas externas de señales conectadas a rutas de señales en el <process diagram>. Se colocan fuera del <frame symbol>, próximo al punto extremo de las rutas de señales en el <frame symbol>.

<process text area> ::=

<text symbol> ***contains*** {
[<valid input signal set>]
{<signal definition>
| <signal list definition>
| <variable definition>
| <view definition>
| <imported variable specification>
| <imported procedure specification>
| <remote procedure definition>
| <remote variable definition>
| <data definition>
| <macro definition>
| <timer definition>
| <select definition> }* }

<process heading> ::=

process {<process name> | <process identifier>}
[<number of process instances> [<end>]]
[<formal parameters>]

<process graph area> ::=

<start area> { <state area> | <in-connector area> }*

<service interaction area> ::=

{ <service area>
| <signal route definition area> }+

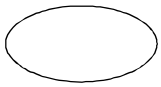
<service area> ::=

<graphical service reference>
| <service diagram>
| <graphical typebased service definition>
| <existing typebased service definition>

<graphical service reference> ::=

<service symbol> ***contains*** <service name>

<service symbol> ::=



En la Figura 2.10.10 se muestra un ejemplo de <process diagram>.

Semántica

Las consideraciones siguientes se aplican cuando la *Process-definition* contiene un *Process-graph*. La semántica adicional para el caso en que la *Process-definition* contiene *Service-definitions* se examina 2.4.5.

Una definición de proceso define un conjunto de procesos. Varias instancias del mismo conjunto de procesos pueden existir al mismo tiempo y ejecutarse asíncronamente y en paralelo entre sí y con instancias de otros conjuntos de procesos en el sistema.

En el *Number-of-instances*, el primer valor representa el número de instancias del proceso que existe cuando se crea el sistema, y el segundo valor representa el número máximo de instancias simultáneas del proceso.

Una instancia de proceso es una máquina de estados finitos ampliada, que comunica, y que realiza cierto conjunto de acciones, denominado transición, en función de la recepción de una señal determinada, cuando se encuentra en un estado. Después de efectuada una transición, la instancia de proceso se encontrará en espera en otro estado, que no tiene que ser necesariamente diferente del precedente.

El concepto de máquina de estados finitos se ha ampliado en los sentidos siguientes:

a) Variables y ramificación: el estado resultante después de una transición, distinto de la señal que origina la transición, puede verse afectado por decisiones tomadas sobre variables conocidas por el proceso.

b) Transición espontánea: puede activarse espontáneamente una transición sin que se reciba ninguna señal. Esta activación es no determinista, y por tanto hace inestable el estado al que está asociada.

c) Conservar: con el constructivo conservar puede especificarse que el orden de consumo de las señales sea diferente de su orden de llegada a la cola.

Cuando se crea un sistema, los procesos iniciales se crean en un orden arbitrario. La comunicación de señales entre los procesos sólo comienza después de que se hayan creado todos los procesos iniciales. Los parámetros formales de estos procesos iniciales no tienen valores asociados, es decir que son «indefinidos».

Las instancias de proceso existen desde el instante en que un sistema es creado, o pueden ser creadas por acciones de petición de crear los procesos que se están interpretando; su interpretación comienza cuando se interpreta la acción de arrancar; pueden dejar de existir ejecutando acciones de parada.

Las señales recibidas por instancias de proceso se denominan señales de entrada, y las señales enviadas desde instancias de proceso se denominan señales de salida.

Las señales sólo pueden ser consumidas por una instancia de proceso cuando ésta se encuentra en un estado. El conjunto completo de señales de entrada válidas es la unión del conjunto de señales en todas las rutas de señales que conducen al conjunto de instancias denotadas por la definición de proceso, el <valid input signal set>, las señales implícitas introducidas por los conceptos adicionales de 4.10 a 4.14, y las señales de temporizador.

Sólo un puerto de entrada está asociado exactamente a cada instancia de proceso. Cuando una señal de entrada llega al proceso, se aplica al puerto de entrada de la instancia de proceso.

El proceso está, bien en espera en un estado, bien efectuando una transición. Para cada estado hay un conjunto de señales de conservación (véase también 2.6.5). Cuando el proceso se encuentra en espera en un estado, la primera señal de entrada cuyo identificador no forme parte del conjunto de señales de conservación es extraída de la cola y consumida por el proceso. Una transición también se puede iniciar como una transición espontánea, independientemente de que haya señales en la cola.

El puerto de entrada puede retener cualquier número de señales de entrada, de modo que varias señales de entrada formen cola para la instancia de proceso. El conjunto de señales retenidas son ordenadas en la cola según el instante de llegada. Si dos o más señales llegan «simultáneamente» por trayectos diferentes, se ordenan arbitrariamente.

Cuando se crea el proceso, se le da un puerto de entrada vacío, y se crean variables locales.

Los parámetros formales son variables creadas, bien cuando el sistema es creado (pero no se le han pasado parámetros efectivos, por lo cual son «indefinidos»), bien cuando la instancia del proceso es creada dinámicamente.

Para todas las instancias de proceso pueden utilizarse cuatro expresiones que dan un valor de PId (véase D.10): **self, parent, offspring** y **sender**. Estas expresiones dan un resultado para:

a) la instancia de proceso (**self**);

b) la instancia del proceso creador (**parent**);

c) la última instancia de proceso creada por la instancia de proceso (**offspring**);

d) la instancia de proceso a partir de la cual se ha consumido la última señal de entrada (**sender**) (véase también 2.6.4).

Estas expresiones se explican con más detalle en 5.4.4.3.

self (**mismo**), **parent** (**progenitor**), **offspring** (**vástago**) y **sender** (**emisor**) pueden utilizarse en expresiones dentro de las instancias de proceso.

Para todas las instancias de proceso presentes en la inicialización del sistema, la expresión predefinida **parent** siempre tiene el valor Null.

Para todas las instancias de proceso de nueva creación, las expresiones predefinidas **sender** y **offspring** tienen el valor Null.

2.4.5 Servicio

El concepto de servicio ofrece una alternativa al gráfico de proceso mediante un conjunto de servicios. En muchas situaciones, las definiciones de servicio pueden reducir la complejidad global y mejorar la legibilidad en comparación con la utilización de un cuerpo de proceso. Además, cada definición de servicio puede definir un comportamiento parcial del proceso, lo que puede ser útil en algunas aplicaciones.

Gramática abstracta

Service-definition	::	*Service-name* *Procedure-definition*-**set** *Data-type-definition* *Syntype-definition*-**set** *Variable-definition*-**set** *View-definition*-**set** *Timer-definition*-**set** *Service-graph*
Service-name	=	*Name*
Service-graph	::	*Service-start-node* *State-node*-**set**
Service-start-node	::	*Transition*

Gramática textual concreta

<service definition> ::=
service {<service name> | <service identifier>} <end>
[<valid input signal set>]
{<entity in service>}*
<service body>
endservice [{<service name> | <service identifier>}] <end>

<entity in service> ::=
<variable definition>
| <data definition>
| <view definition>
| <imported variable specification>
| <imported procedure specification>
| <select definition>
| <macro definition>
| <procedure definition>
| <textual procedure reference>
| <timer definition>

<service body> ::=
<process body>

Diferentes <service definition>s o <textual typebased service definition>s en <process definition> no deben revelar el mismo nombre de variable del mismo género, exportar o importar el mismo nombre de variable, ni exportar o importar el mismo procedimiento remoto.

Si una variable o procedimiento exportados están definidos en el proceso que los contiene o una variable o procedimiento importados están especificados en el proceso que los contiene, la variable/procedimiento definidos/especificados no deben estar definidos/especificados con el mismo nombre que el <remote variable name> o <remote procedure name> correspondientes en una de las <service definition>s o <textual typebased service definition>s, y sólo puede utilizarse como exportado/importado en un servicio. Las señales implícitas para una variable

exportada se añaden a un servicio arbitrario si no se utiliza en ningún servicio. Un procedimiento exportado definido en el proceso que lo contiene debe mencionarse como exportado en un servicio exactamente.

Los conjuntos completos de señales de entrada válidas de los servicios dentro de un proceso deben estar separados. El conjunto completo de señales de entrada válidas de un servicio es la unión del <valid input signal set> de su <service definition> y el conjunto de señales transmitidas por las rutas de señales de entrada al servicio, que comprenden las señales de entrada implícitas introducidas por los conceptos adicionales de 4.10 a 4.14 y las señales de temporizador.

Gramática gráfica concreta

<service diagram> ::=

<frame symbol> ***contains***
{<service heading>
{ {<service text area> }*
{<graphical procedure reference>}*
{<procedure diagram>}*
{<macro diagram>}*
<service graph area> } ***set*** }

<service heading> ::=

service {<service name> | <service identifier>}

<service text area> ::=

<text symbol> ***contains***
{ <variable definition>
| <data definition>
| <timer definition>
| <view definition>
| <imported variable specification>
| <imported procedure specification>
| <select definition>
| <macro definition> }*

<service graph area> ::=

<process graph area>

Semántica

Dentro de una instancia de proceso hay una instancia de servicio para cada *Service-definition* en la *Process-definition*. Las instancias de servicio son componentes de la instancia de proceso y no pueden tratarse como objetos separados. Estas instancias de servicio comparten el puerto de entrada y las expresiones **self, parent, offspring** y **sender** de la instancia de proceso.

Una instancia de servicio es una máquina de estados.

Cuando la instancia de proceso es creada, los *Service-start-node*s se ejecutan en orden arbitrario. Ningún estado de ningún servicio es interpretado antes de que todos los *Service-start-node*s han sido completados. Un *Service-start-node* se considera completado cuando la instancia de servicio entra por primera vez en un *State-node* (posiblemente dentro de un procedimiento llamado) o interpreta un *Stop-node*.

Sólo un servicio a la vez ejecuta una *Transition*. Cuando el servicio ejecutante alcanza un estado, la señal siguiente en el puerto de entrada (que no es conservada por el servicio, que si no es capaz de consumirla) se da al servicio que es capaz de consumirla.

Cuando un servicio deja de existir, se descartan las señales de entrada para ese servicio. Cuando todos los servicios han dejado de existir, la instancia de proceso deja de existir.

Ejemplo

Véase 2.10.

2.4.6 Procedimiento

Los procedimientos se definen por medio de definiciones de procedimiento. El procedimiento es invocado mediante una llamada a procedimiento que identifica la definición de procedimiento. Una llamada a procedimiento tiene asociados parámetros. El mecanismo de paso de los parámetros controla qué variables son afectadas por la interpretación de un procedimiento. Las llamadas a procedimiento pueden ser acciones o parte de expresiones (sólo procedimientos de retorno de valor).

Gramática abstracta

Procedure-definition	::	*Procedure-name* *Procedure-formal-parameter** *Procedure-definition*-**set** *Data-type-definition* *Syntype-definition*-**set** *Variable-definition*-**set** *Procedure-graph*
Procedure-name	=	*Name*
Procedure-formal-parameter	=	*In-parameter* | *Inout-parameter*
In-parameter	::	*Variable-name* *Sort-reference-identifier*
Inout-parameter	::	*Variable-name* *Sort-reference-identifier*
Procedure-graph	::	*Procedure-start-node* *State-node*-**set**
Procedure-start-node	::	*Transition*

Gramática textual concreta

```
<procedure definition>  ::=
                        <procedure preamble>
                        procedure { <procedure name> | <procedure identifier>}
                              [<formal context parameters>]
                              [ <virtuality constraint>]
                              [<specialization>]  <end>
                              [<procedure formal parameters> <end>]
                              [<procedure result> <end>]
                              {       <entity in procedure>
                                 |    <select definition>
                                 |    <macro definition> }*
                              [ <procedure body> ]
                        endprocedure [<procedure name> | <procedure identifier>] <end>

<procedure preamble> ::=
                        [<virtuality>][exported  [ as <remote procedure identifier> ]]

<procedure formal parameters> ::=
                        fpar <formal variable parameters>
                              {, <formal variable parameters> }*

<formal variable parameters> ::=
                        <parameter kind>  <parameters of sort>
```

<parameter kind> ::=
[**in/out** | **in**]

<procedure result> ::=
returns [<variable name>] <sort>

<entity in procedure> ::=
<data definition>
| <variable definition>
| <textual procedure reference>
| <procedure definition>

<procedure body> ::=
<start> {<state> | <free action>}*
| {<state> | <free action>}+

Un procedimiento exportado no puede tener parámetros de contexto formales y su ámbito circundante debe ser un tipo de proceso, una definición de proceso, un tipo de servicio o una definición de servicio.

<variable definition> en una <procedure definition> no puede contener **revealed, exported, revealed exported, exported revealed** <variable name>s (véase 2.6.1).

El atributo exportado es heredado por cualquier subtipo de un procedimiento. Un procedimiento virtual exportado debe contener **exported** en todas las redefiniciones. El tipo virtual con su procedimiento se describen en 6.3.2. La cláusula optativa **as** en una redefinición debe denotar el mismo <remote procedure identifier> que en el supertipo. Si se omite en una redefinición, <remote procedure identifier> del supertipo es implícito.

Dos procedimientos exportados en un proceso, incluidos los servicios posibles, no pueden mencionar el mismo <remote procedure identifier>.

En la Figura 2.10.11 se da un ejemplo de <procedure definition>.

Gramática gráfica concreta

<procedure diagram> ::=
<frame symbol> ***contains*** {<procedure heading>
{ { <procedure text area>
| <procedure area>
| <macro diagram> }*
<procedure graph area> }***set*** }

<procedure area> ::=
<graphical procedure reference>
| <procedure diagram>

<procedure text area> ::=
<text symbol> ***contains***
{ <variable definition>
| <data definition>
| <select definition>
| <macro definition> }*

<graphical procedure reference> ::=
<procedure symbol> ***contains***
{ <procedure preamble>
<procedure name> }

<procedure symbol> ::=

<procedure heading> ::=

<procedure preamble>
procedure {<procedure name>| <procedure identifier> }
[<formal context parameters>]
[<virtuality constraint>]
[<specialization>]
[<procedure formal parameters>]
[<procedure result>]

<procedure graph area> ::=

[<procedure start area>]
{<state area> | <in-connector area> }*

<procedure start area> ::=

<procedure start symbol>
contains { [<virtuality>] }
is followed by <transition area>

<procedure start symbol> ::=

La Figura 2.10.12, muestra un ejemplo de <procedure diagram>.

Semántica

Un procedimiento es un medio de dar un nombre a un grupo de items y representar este grupo por una sola referencia. Las reglas para los procedimientos prescriben la forma de elegir el grupo de items, y limitan el ámbito del nombre de variables definidas en el procedimiento.

exported en un <procedure preamble> entraña que el procedimiento puede ser llamado como un procedimiento remoto, de acuerdo con el modelo de 4.14.

Una variable de procedimiento es una variable local dentro del procedimiento, que no puede ser ni revelada, ni vista, ni exportada, ni importada. Se crea cuando se interpreta el nodo de arranque de procedimiento, y deja de existir cuando se interpreta el nodo de retorno del gráfico de procedimiento.

La interpretación de una <procedure call> causa la creación de una instancia de procedimiento y hace que la interpretación comience de la manera siguiente:

a) Para cada *In-parameter* se crea una variable local que tiene el *Name* y el *Sort* del *In-parameter*. La variable recibe el valor de la expresión dada por el parámetro efectivo correspondiente, si está presente. Si no, la variable no recibe ningún valor, es decir que se vuelve «indefinido».

b) Un parámetro formal sin atributo explícito tiene el atributo **in** implícito.

c) Para cada *Variable-definition* en la *Procedure-definition* se crea una variable local.

d) Cada *Inout-parameter* denota una variable que se da en la expresión del parámetro efectivo. El *Variable name* contenido se utiliza durante toda la interpretación del *Procedure-graph* cuando se hace referencia al valor de la variable o cuando se asigna un nuevo valor a la variable.

e) Se interpreta la *Transition* contenida en *Procedure-start-node*.

Los nodos del gráfico de procedimiento son interpretados de la misma manera que los nodos equivalentes de un gráfico de proceso o de servicio, es decir, que el procedimiento tiene el mismo conjunto completo de señales de entrada válida que el proceso circundante, y el mismo puerto de entrada que la instancia del proceso circundante que lo ha llamado, ya sea directa o indirectamente.

Modelo

La especificación de <procedure result> resulta en <procedure formal parameters> con un parámetro de variable formal suplementario adjunto con el atributo **in/out**, un nuevo nombre distinto (a menos que esté especificado explícitamente en <procedure result>) como <variable name> y el <sort> de <procedure result> como <sort>.

Cuando un <procedure body> o una <procedure graph area> contiene <return> o <return area> con una <expression>, el procedimiento debe tener por lo menos un parámetro formal con <parameter kind> **in/out** o un <procedure result>.

Cualquier ocurrencia de <return> o <return area> con una <expression> dentro de <procedure body> o <procedure graph area> es sustituida por una <task> que contiene una asignación de la <expression> contenida de la variable **in/out** situada a la derecha, seguida por un <return> o <return area> sin ninguna <expression>.

Las transformaciones se efectúan después de *Generic systems* (véase 4.3) y antes de <value returning procedure call>.

Una <procedure start area> que contiene <virtuality> se denomina un área virtual de arranque de procedimiento, que se describe más detalladamente en 6.3.3.

2.5 Comunicación

2.5.1 Canal

Gramática abstracta

Channel-definition	*::*	*Channel-name*
		[**NODELAY**]
		Channel-path
		[*Channel-path*]
Channel-path	::	*Originating-block*
		Destination-block
		Signal-identifier-**set**
Originating-block	=	*Block-identifier* |
		ENVIRONMENT
Destination-block	=	*Block-identifier* |
		ENVIRONMENT
Block-identifier	=	*Identifier*
Signal-identifier	=	*Identifier*
Channel-name	=	*Name*

Por lo menos uno de los puntos extremos del canal tiene que ser un bloque. Si los dos puntos extremos son bloques, los *Block-identifier*s tienen que ser diferentes.

El(los) punto(s) extremo(s) de bloque deben estar definidos en la misma unidad de ámbito en que está definido el canal.

NODELAY denota que el canal no tiene retardo.

Gramática textual concreta

<channel definition> ::=
channel <channel name> [**nodelay**]
<channel path>
[<channel path>]
[<channel substructure definition>
| <textual channel substructure reference>]
endchannel [<channel name>] <end>

<channel path> ::=
from <channel endpoint>
to <channel endpoint> **with** <signal list><end>

<channel endpoint> ::=
{<block identifier> | **env** }[**via** <gate>]

Cuando se definen dos <channel path>s, deben ser de sentidos opuestos.

<gate> sólo debe especificarse si:

a) <channel endpoint> denota una conexión con una <textual typebased block definition>, en cuyo caso <gate> debe estar definida directamente en el tipo de bloque para ese bloque, o

b) **env** está especificado y el canal está definido en una subestructura de bloque de una <block type definition>, en cuyo caso <gate> debe estar definida en ese tipo de bloque.

Gramática gráfica concreta

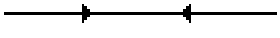
<channel definition area> ::=
<channel symbol>
is associated with {<channel name>
{ [{<channel identifiers> | <block identifier> | <gate> }]
<signal list area> [<signal list area>] }***set*** }
is connected to { <block area> {<block area> | <frame symbol>}
[<channel substructure association area>] }***set***

<channel identifiers> sólo puede especificarse si el diagrama circundante es un <block substructure diagram> directamente circundado por <block definition>. <block identifier> sólo puede especificarse si el diagrama circundante es un <channel substructure diagram>. Los <channel identifiers> identifican canales externos conectados al <block substructure diagram> delimitado por el <frame symbol>. El <block identifier> identifica un bloque externo como un punto externo de canal para el <channel substructure diagram> delimitado por el <frame symbol>. Cuando <channel symbol> está conectado a <frame symbol>, la <gate> con la cual está asociado está definida para ese tipo de bloque mediante una <gate> o una <graphical gate constraint> asociada con el diagrama de tipo de bloque.

<channel symbol> ::=
<channel symbol 1>
| <channel symbol 2>
| <channel symbol 3>
| <channel symbol 4>
| <channel symbol 5>

<channel symbol 1> ::=

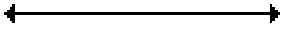
<channel symbol 2> ::=



<channel symbol 3> ::=

<channel symbol 4> ::=

<channel symbol 5> ::=



Para cada flecha en el <channel symbol> tiene que haber un <signal list area>. Un <signal list area> tiene que estar suficientemente próximo a la flecha con la que está asociado para que tal asociación sea inequívoca.

Las flechas para <channel symbol 4> y <channel symbol 5> están situadas en los extremos del canal e indican que el canal no tiene retardo.

Semántica

Un canal representa una ruta de transporte de señales. Un canal puede considerarse como uno o dos trayectos de canal unidireccionales e independientes entre dos bloques o entre un bloque y su entorno.

Signal-identifier-**set** en cada *Channel-path* en la *Channel-definition* contiene las señales que pueden ser transportadas en ese *Channel-path*.

Las señales transportadas por canales se entregan al punto extremo de destino.

Las señales son presentadas en el punto extremo de destino de un canal en el mismo orden en que fueron presentadas en el punto de origen. Si dos o más señales son presentadas simultáneamente al canal, serán ordenadas arbitrariamente.

Un canal con retardo puede demorar las señales que transporta. Esto significa que hay una cola de espera de tipo primero-en-entrar-primero-en-salir (FIFO, *first-in-first-out*) asociada a cada sentido de transmisión de un canal. Una señal presentada al canal es introducida en la cola de espera. Tras un intervalo de tiempo indeterminado y no constante, la primera instancia de señal es liberada de la cola y aplicada a uno de los canales o a una de las rutas de señales que está conectado(a) al canal.

Pueden existir varios canales entre los dos mismos puntos extremos. Canales diferentes pueden transportar señales del mismo tipo.

Si un punto extremo de un canal es <textual typebased block definition>, el otro punto extremo puede ser la misma <textual typebased block definition>. El <number of block instances> debe ser por lo tanto superior a uno para <textual typebased block definition>.

Modelo

Un canal cuyos dos puntos extremos son puertas de una <textual typebased block definition> representa canales individuales hacia o desde cada uno de los bloques del conjunto.

Si uno de los puntos extremos de un canal es una puerta de <textual typebased block definition>, ese canal representa canales individuales hacia o desde cada uno de los bloques del conjunto. Los canales individuales tienen la misma propiedad de retardo que el canal ampliado.

2.5.2 Ruta de señales

Gramática abstracta

Signal-route-definition	::	*Signal-route-name* *Signal-route-path* [*Signal-route-path*]

Signal-route-path :: *Origin*
Destination
Signal-identifier-**set**

Origin = *Process-identifier* |
Service-identifier |
ENVIRONMENT

Destination = *Process-identifier* |
Service-identifier |
ENVIRONMENT

Signal-route-name = *Name*

Process-identifier = *Identifier*

Service-identifier = *Identifier*

Por lo menos uno de los puntos extremos de la ruta de señales tiene que ser *Process-identifier* o *Service-identifier*.

Si los dos puntos extremos son procesos, los *Process-identifiers* tienen que ser diferentes. Si son servicios, los *Service-identifiers* tienen que ser diferentes.

El(los) punto(s) extremo(s) de proceso o de servicio tienen que estar definidos en la misma unidad de ámbito en que está definida la ruta de señales.

Gramática textual concreta

<signal route definition> ::=
signalroute <signal route name>
<signal route path>
[<signal route path>]

<signal route path>::=
from <signal route endpoint> **to** <signal route endpoint>
with <signal list> <end>

<signal route endpoint> ::=
{<process identifier> | <service identifier> | **env** }
[**via** <gate>]

Cuando dos <signal route path>s están definidos, deben ser de sentidos opuestos.

<gate> sólo debe especificarse si:

a) <signal route endpoint> denota una conexión con una <textual typebased process definition> o una <textual typebased service definition>, en cuyo caso <gate> debe estar definida directamente en el tipo de proceso o el tipo de servicio para ese proceso o servicio, respectivamente, o

b) **env** está especificado y la ruta de señales está definida en un tipo de bloque o tipo de proceso, en cuyo caso <gate> debe estar definida en ese tipo de bloque o tipo de proceso, respectivamente.

Gramática gráfica concreta

<signal route definition area> ::=

<signal route symbol>

is associated with {<signal route name>

{[<channel identifiers> | <external signal route identifiers> |

<gate>]

<signal list area> [<signal list area>] }***set*** }

is connected to

{{<process area> | <service area>}

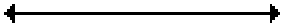
{<process area> | <service area> | <frame symbol>} }***set***

<signal route symbol> ::=

<signal route symbol 1> | <signal route symbol 2>

<signal route symbol 1> ::=

<signal route symbol 2> ::=



Un símbolo de ruta de señales incluye una punta de flecha en un extremo (un solo sentido) o una punta de flecha en cada extremo (ambos sentidos), que indica el sentido del flujo de las señales.

Para cada punta de flecha en el <signal route symbol> debe haber una <signal list area>. Una <signal list area> tiene que estar lo suficientemente próxima a la punta de flecha a que está asociada para que tal asociación sea inequívoca.

Cuando <signal route symbol> está conectado a <frame symbol>, para un <block type diagram>, <gate> con la cual está asociado es una puerta definida para ese tipo de bloque mediante una <gate> o una <graphical gate constraint> asociada con el <block type diagram>. Cuando <signal route symbol> está conectado a <frame symbol>, para un <process type diagram>, <gate> con la cual está asociado es una puerta definida para ese tipo de proceso mediante una <gate> o una <graphical gate constraint> asociada con el <process type diagram>.

Semántica

Una ruta de señales representa una ruta de transporte de señales. Una ruta de señales puede considerarse como uno o dos trayectos de ruta de señales unidireccionales e independientes entre dos conjuntos de instancias de proceso denotados cada uno por una definición de proceso, o entre un conjunto de instancias de proceso y el entorno del bloque circundante, o entre dos servicios, o entre un servicio y el entorno del proceso circundante.

Las señales transportadas por rutas de señales se entregan a los puntos extremos de destino.

Una ruta de señales no introduce ningún retardo en el transporte de las señales.

Cuando una instancia de señal es enviada a una instancia del mismo conjunto de instancias de proceso, la interpolación de *Output-node* implica que la señal se pone directamente en el puerto de entrada del proceso de destino.

Pueden existir varias rutas de señales entre los mismos dos puntos extremos. Diferentes rutas de señales pueden transportar señales del mismo tipo.

Modelo

Si una <block definition> o <block type definition> contiene <signal route definition>s, el <valid input signal set> en una <process definition>, de haberlo, no necesita contener señales en las rutas de señales que conducen al conjunto de instancias de proceso.

Si una <process type definition> o <process definition> contiene servicios, el <valid input signal set> se deriva uniendo las señales de entrada a las puertas (en caso de una <process type definition>) o las señales en las rutas de señales que conducen al proceso (en caso de una <process definition>), las señales de entrada en rutas de señales que conducen a los servicios y los <valid input signal set>s para los servicios. Este conjunto se llama <valid input signal set> implícito. Si el <valid input signal set> del proceso está especificado explícitamente, debe ser un subconjunto del <valid input signal set> implícito.

Si una <process definition> o <process type definition> contiene <signal route definition>s, no es necesario que el <valid input signal set> en una <service definition>, si lo hay, contenga señales en rutas de señales que conducen al servicio.

Si una <block definition> no contiene <signal route definition>s, todas las <process definition>s en ese ámbito deben contener (implícita o explícitamente) un <valid input signal set>. En ese caso, las <signal route definition>s y las <channel to route connection>s se derivan de los <valid input signal set>s, las <output>s y los canales que terminan en la frontera de los bloques.

Si una <block type definition> no contiene <signal route definition>s, todas las <process definition>s en ese ámbito deben contener (implícita o explícitamente) un <valid input signal set>. En ese caso, las <signal route definition>s se derivan de <valid input signal set>s, <output>s y <signal list>s en las puertas de <block type definition>.

Las señales que corresponden a un determinado sentido de flujo entre dos conjuntos de instancias de proceso en la ruta de señales implicada constituyen la intersección de las señales especificadas en el <valid input signal set> del proceso de destino y las señales mencionadas en una salida del proceso de origen. Si uno de los puntos extremos es el entorno, el conjunto de entrada/conjunto de salida para ese punto extremo está constituido por las señales transportadas por el canal o la puerta (si se trata de un tipo de bloque) en el sentido dado.

Si una definición de proceso es una <textual typebased process definition>, la derivación no se basa en los <valid input signal set>s ni las <output>s, sino en las <signal list>s de entrada y salida de las puertas del tipo de proceso.

Si <process definition> o <process type definition> no contiene <signal route definition>s, todas las <service definition>s en ese ámbito deben contener un <valid input signal set>. En ese caso las <signal route definition>s y las <signal route to route connection>s se derivan de <valid input signal set>s, <output>s y rutas de señales externas que terminan en las fronteras o puertas de proceso del tipo de proceso.

Las señales correspondientes a un sentido dado entre dos servicios en la ruta de señales implicada constituyen la intersección de las señales especificadas en <valid input signal set> del servicio de destino y las señales mencionadas en una salida del servicio de origen. Si uno de los puntos extremos es el entorno, el conjunto de entrada/conjunto de salida para ese punto extremo es la señal transmitida por la ruta de señales o la puerta (si se trata de un tipo de proceso) externas en el sentido dado.

Si una definición de servicio es una <textual typebased service definition>, la derivación no se basa en <valid input signal set>s ni en <output>s, sino en <signal list>s de entrada y salida de las puertas del tipo de servicio.

2.5.3 Conexión

Gramática abstracta

Channel-to-route-connection	::	*Channel-identifier*-**set** *Signal-route-identifier*-**set**
Signal-route-identifier	=	*Identifier*
Signal-route-to-route-connection	::	*External-signal-route-identifier*-**set** *Signal-route-identifier*-**set**
External-signal-route-identifier	=	*Identifier*

En 3 figuran otros constructivos de conexión.

Los *Channel-identifier*s en un *Channel-identifier*-**set** en una *Channel-to-route-connection* deben denotar canales conectados al bloque envolvente.

Cada *Channel-identifier* conectado al bloque circundante debe ser mencionado exactamente en una *Channel-to-route-connection*. Cada *Signal-route-identifier* en una *Channel-to-route-connection* tiene que estar definido en el mismo bloque en que está definida la *Channel-to-route-connection* y tiene que tener la frontera de ese bloque como uno de sus

puntos extremos. Cada *Signal-route-identifier* definido en el bloque circundante y que tiene el entorno como uno de sus puntos extremos, debe mencionarse en una *Channel-to-route-connection* y sólo en una.

Para un determinado sentido de flujo, la unión de los conjuntos *Signal-identifier* en las rutas de señales en una *Channel-to-route-connection* tiene que ser igual a la unión de los conjuntos *Signal-identifier* transportadas por los *Channel-identifier*s en la misma *Channel-to-route-connection* y correspondientes al mismo sentido de flujo.

Los *External-signal-route-identifier*s en un *External-signal-route-identifier***-set** en una *Signal-route-to-route-connection* deben denotar rutas de señales conectadas al proceso que los contiene.

Cada *External-signal-route-identifier* conectado al proceso que lo contiene debe estar mencionado en una sola *Signal-route-to-route-connection*. Cada *Signal-route-identifier* en una *Signal-route-to-route-connection* debe estar definido en el mismo proceso en el cual está definida la *Signal-route-to-route-connection*, y debe tener la frontera de ese proceso como uno de sus puntos extremos. Cada *Signal-route-identifier* definido en el proceso circundante que tiene su entorno como uno de sus puntos extremos debe mencionarse en una sola *Signal-route-to-route-connection*.

Para un sentido dado, la unión de los conjuntos *Signal-identifier* en las rutas de señales en una *Signal-route-to-route-connection* debe ser igual a la unión de los conjuntos *Signal-identifier* transportados por los *External-signal-route-identifier*s en la misma *Signal-route-to-route-connection* y correspondiente al mismo sentido.

Gramática textual concreta

<channel to route connection> ::=
connect <channel identifiers>
and <signal route identifiers> <end>

<channel identifiers> ::=
<channel identifier> {, <channel identifier>}*

<signal route identifiers> ::=
<signal route identifier> {, <signal route identifier>}*

<signal route to route connection> ::=
connect <external signal route identifiers>
and <signal route identifiers> <end>

<external signal route identifiers> ::=
<signal route identifier>
{, <signal route identifier>}*

Gramática gráfica concreta

Gráficamente, el constructivo de conectar está representado por <channel identifiers> y <external signal route identifiers> asociados con las rutas de señales y contenidos en <signal route definition area> (véase 2.5.2, *Gramática gráfica concreta*).

Ningún <signal route identifier> o <channel identifier> puede mencionarse más de una vez en las conexiones de un diagrama.

2.5.4 Señal

Gramática abstracta

Signal-definition	::	*Signal-name* *Sort-reference-identifier** [*Signal-refinement*]
Signal-name	=	*Name*

Gramática textual concreta

<signal definition>::=
signal
<signal definition item>
{,<signal definition item>}* <end>

<signal definition item> ::=
<signal name>
[<formal context parameters>]
[<specialization>]
[<sort list>][<signal refinement>]

<sort list> ::=
(<sort> {, <sort>}*)

<formal context parameter> en <formal context parameters> debe ser un <sort context parameter>. El <base type> que forma parte de <specialization> debe ser un <signal identifier>.

Semántica

Una instancia de señal es un flujo de información entre procesos, y también una instanciación de un tipo de señal definido por una definición de señal. Una instancia de señal puede ser enviada, bien por el entorno, bien por un proceso, y siempre está dirigida, sea hacia un proceso, sea hacia el entorno. Una instancia de señal es creada cuando un *Output-node* es interpretado y deja de existir cuando un *Input-node* es interpretado.

2.5.5 Definición de lista de señales

Un <signal list identifier> puede utilizarse en <signal list> como una notación taquigráfica para una lista de identificadores de señal y señales de temporizador.

Gramática textual concreta

<signal list definition> ::=
signallist <signal list name> **=** <signal list><end>

<signal list> ::=
<signal list item> { **,** <signal list item>}*

<signal list item> ::=
<signal identifier> | **(** <signal list identifier> **)** | <timer identifier>

La <signal list> que se construye reemplazando todos los <signal list identifier>s de la lista por la lista de los <signal identifier>s o <timer identifier>s que éstos denotan, corresponde a un *Signal-identifier*-**set** en la *gramática abstracta*.

La <signal list> no debe contener el <signal list identifier> definido por la <signal list definition>, ya sea directa o indirectamente (por otro <signal list identifier>).

Gramática gráfica concreta

<signal list area> ::=
<signal list symbol> ***contains*** <signal list>

<signal list symbol> ::=

2.6 Comportamiento

2.6.1 Variables

Una variable tiene un valor de un género asociado, o es «indefinida».

2.6.1.1 Definición de variable

Gramática abstracta

Variable-definition	::	*Variable-name* *Sort-reference-identifier* [*Ground-expression*] [**REVEALED**]
Variable-name	=	*Name*

Si está presente *Ground-expression*, debe ser del mismo género que el *Sort-reference-identifier* denotado.

Gramática textual concreta

<variable definition> ::=
dcl
[revealed | exported | revealed exported | exported revealed]
<variables of sort> {**,** <variables of sort> }*<end>

<variables of sort> ::=
<variable name> [<exported as>] {**,** <variable name> [<exported as>]}*
<sort> [:= <ground expression>]

<exported as> ::=
as <remote variable identifier>

<export as> sólo puede emplearse para una variable con el atributo **exported**. Dos variables exportadas en un proceso, incluidos los servicios posibles, no pueden mencionar el mismo <remote variable identifier>.

Semántica

Cuando una variable es creada y *Ground-expression* está presente, la variable está asociada con el valor de *Ground-expression*. En los demás casos la variable no tiene valor asociado, es decir que es «indefinida».

Si *Sort-reference-identifier* es una *Syntype-identifier*, *Ground-expression* está presente y su valor no cumple la *Range-condition*, el sistema está en error y no se especifica el comportamiento ulterior del sistema.

El atributo **revealed** permite a todos los otros procesos del mismo bloque leer el valor de una variable, siempre que tengan una <view definition> con el mismo nombre y género.

El atributo **exported** permite emplear una variable como variable exportada, como se indica en 4.13.

Modelo

Ground-expression está representada:

a) si <ground expression> viene dada en la <variable definition>, por esa <ground expression>;

b) en los demás casos, si <sort> tiene una <default initialization>, por <ground expression> de la <default initialization>.

En los demás casos *Ground-expression* no está presente.

2.6.1.2 Definición de visión

Gramática abstracta

View-definition	::	*View-name* *Sort-reference-identifier*

En la *Block-definition* circundante debe existir por lo menos una *Process-definition* que contenga una *Variable-definition* **REVEALED** con el mismo *Sort-reference-identifier* y el mismo *Variable-name* que el *Name* de *View-name*.

Gramática textual concreta

<view definition> ::=
viewed
<view name> {**,** <view name>}* <sort>
{**,**<view name>{, <view name>}*<sort>}* <end>

Semántica

El mecanismo de visión permite a una instancia de proceso ver continuamente el valor de la variable vista como si ésta estuviese definida localmente. La instancia de proceso que observa no tiene, sin embargo, derecho a modificarla.

2.6.2 Arranque

Gramática abstracta

Process-start-node	::	*Transition*

Gramática textual concreta

<start> ::=
start [<virtuality>] <end> <transition>

Gramática gráfica concreta

<start area> ::=
<start symbol>
contains { [<virtuality>] }
is followed by <transition area>

<start symbol> ::=

Semántica

Se interpreta, la *Transition* del *Process-start-node*.

Modelo

Un <start> o <start area> que contenga <virtuality> se denomina un arranque virtual, que se describe en 6.3.3.

2.6.3 Estado

Gramática abstracta

State-node	::	*State-name*
		Save-signalset
		Input-node-**set**
		Spontaneous-transition-**set**
State-name	=	*Name*

Dentro de un *Process-graph*, *Service-graph* o *Procedure-graph* los *State-node*s deben tener diferentes *State-name*s.

Para cada *State-node*, todos los *Signal-identifier*s (en el conjunto completo de señales de entrada válidas) aparecen, bien en un *Save-signalset*, bien en un *Input-node*.

Los *Signal-identifier*s en el *Input-node*-**set** tienen que ser distintos.

Gramática textual concreta

<state> ::=

state <state list> <end>
{ <input part>
| <priority input>
| <save part>
| <spontaneous transition>
| <continuous signal>}*
[**endstate** [<state name>] <end>]

<state list> ::=

{<state name> { **,** <state name> }*}
| <asterisk state list>

Cuando la <state list> contiene un solo <state name>, el <state name> representa un *State-node*. Para cada *State-node*, el *Save-signalset* se representa por la <save part> y eventuales conservaciones implícitas de señales. Para cada *State-node*, el *Input-node*-**set** se representa por la <input part> y las eventuales señales de entrada implícitas. Para cada *State-node*, una *Spontaneous-transition* está representada por una <spontaneous transition>.

El <state name> opcional que termina un <state> sólo puede especificarse si la <state list> en el <state> consiste en un solo <state name>, en cuyo caso debe ser ese mismo <state name>.

Gramática gráfica concreta

<state area> ::=
<state symbol> ***contains*** <state list> ***is associated with***
{ <input association area>
| <priority input association area>
| <continuous signal association area>
| <spontaneous transition association area>
| <save association area> }*

<state symbol> ::=

<input association area> ::=
<solid association symbol> ***is connected to*** <input area>

<save association area> ::=
<solid association symbol> ***is connected to*** <save area>

<spontaneous transition association area> ::=
<solid association symbol> ***is connected to***
<spontaneous transition area>

Un <state area> representa uno o más *State-node*s.

Los <solid association symbol>s que tienen su origen en un <state symbol> pueden tener un trayecto de origen común.

Un <state area> debe contener <state name> si coincide con un <nextstate area>.

Semántica

Un estado representa una condición particular en la cual un proceso puede consumir una instancia de señal, lo que causa una transición. Si el estado no tiene transiciones espontáneas ni señales continuas, y no hay instancias de señal en el puerto de entrada que no sean las mencionadas en una conservación, el proceso espera en el estado hasta que se reciba una instancia de señal.

En cualquier instante en un estado que contenga *Spontaneous-transition*s, el proceso puede pasar a la *Transition* de una de las *Spontaneous-transition*s.

Modelo

Cuando la <state list> de cierto <state> contiene más de un <state name>, se crea una copia del <state> para cada uno de estos <state name>. Entonces el <state> es reemplazado por estas copias.

2.6.4 Entrada

Gramática abstracta

Input-node :: *Signal-identifier*
[*Variable-identifier*]*
Transition

Variable-identifier = *Identifier*

La longitud del [*Variable-identifier*]* debe ser igual al número de *Sort-reference-identifier*s de la *Signal-definition* denotada por el *Signal-identifier*.

Los géneros de las variables deben corresponder en posición a los géneros de los valores que pueden ser transportados por la señal.

Gramática textual concreta

<input part> ::=
<basic input part>
| <remote procedure input transition>

<basic input part> ::=
input [<virtuality>] <input list> <end>
[<enabling condition>]<transition>

<input list> ::=
<asterisk input list>
| <stimulus> { **,**<stimulus> }*

<stimulus> ::=
{ <signal identifier> | <timer identifier>}
[**(** [<variable>] {**,**[<variable>]}* **)**]

Cuando la <input list> contiene un solo <stimulus>, la <input part> representa un *Input-node*. En la *gramática abstracta*, las señales de temporizador (<timer identifier>) son también representadas por *Signal-identifier*. Las señales de temporizador y las señales ordinarias sólo se distinguen cuando proceda, pues en muchos aspectos tienen propiedades similares. Las propiedades exactas de las señales de temporizador se definen en 2.8.

Las comas pueden omitirse después de la última <variable> en <stimulus>.

Gramática gráfica concreta

<input area> ::=
<basic input area>
| <remote procedure input area>

<basic input area> ::=
<input symbol> ***contains*** { [<virtuality>] <input list> }
is followed by {[<enabling condition area>] <transition area>}

<input symbol> ::=
<plain input symbol>
| <internal input symbol>

<plain input symbol> ::=

Un <input area> cuya <input list> contiene un solo <stimulus> corresponde a un *Input-node*. Cada uno de los <signal identifier>s o <timer identifier>s contenidos en un <input symbol> da el nombre de uno de los *Input-node*s que este <input symbol> representa.

Semántica

Una entrada permite el consumo de la instancia de señal de entrada especificada. El consumo de la señal de entrada pone a la disposición del proceso la información transportada por la señal. A las variables asociadas con la entrada se asignan valores transportados por la señal consumida.

Los valores se asignarán a las variables de izquierda a derecha. Si no hay una variable asociada con la entrada para un género especificado en la señal, se descarta el valor de ese género. Si no hay un valor asociado con un género especificado en la señal, la variable correspondiente se convierte en «indefinida».

A la expresión **sender** del proceso consumidor se le da el valor PId del proceso originador, transportado por la instancia de señal.

Las instancias de señal que pasan del entorno a una instancia de proceso dentro del sistema transportarán siempre un valor PId diferente de cualquiera de los del sistema.

Modelo

Cuando la lista de <stimulus>s de cierta <input part> contiene más de un <stimulus>, se crea una copia de la <input part> para cada uno de esos <stimulus>. La <input part> se reemplaza entonces por estas copias.

Cuando una o varias <variable>s de un determinado <stimulus> son <indexed variable>s o <field variable>s, todas las <variable>s son sustituidas por nuevos <variable identifier>s únicos e implícitamente declarados. Directamente después de <input part>, se inserta una <task> que contiene en su <task body> un <assignment statement> para cada una de las <variable>s, asignando el valor de la nueva variable correspondiente a la <variable>. Los valores se asignarán siguiendo el orden de izquierda a derecha de la lista de <variable>s. Esta <task> se convierte en la primera <action> en la <transition>.

Una <basic input part> o <basic input area> que contiene <virtuality> se denomina transición de entrada virtual. La transición de entrada virtual se describe en 6.3.3.

2.6.5 Conservación

Una conservación especifica un conjunto de identificadores de señal cuyas instancias no son relevantes para el proceso en el estado al cual se ha asociado la conservación, y que debe conservarse para un procesamiento futuro.

Gramática abstracta

Save-signalset :: *Signal-identifier*-**set**

Gramática textual concreta

```
<save part>  ::=
                        <basic save part>
                |       <remote procedure save>

<basic save part> ::=
                        save [<virtuality>] <save list> <end>

<save list>  ::=
                        {<signal list> | <asterisk save list>}
```

Una <save list> representa el *Signal-identifier*-**set**. La <asterisk save list> es una notación taquigráfica que se explica en 4.7.

Gramática gráfica concreta

```
<save area> ::=
                        <basic save area>
                |       <remote procedure save area>

<basic save area> ::=
                        <save symbol> contains  { [<virtuality>] <save list> }
```

<save symbol> ::=

Semántica

Las señales conservadas se retienen en el puerto de entrada en el orden de su llegada.

El efecto de la conservación es válido solamente para el estado al cual está asociada la conservación. En el estado siguiente, las instancias de señal que han sido «conservadas» se tratan como instancias de señal normales.

Modelo

Una <basic save part> o <basic save area> que contiene <virtuality> se denomina una conservación virtual, que se describe en 6.3.3.

2.6.6 Transición espontánea

Una transición espontánea especifica una transición de estado sin ninguna recepción de señal.

Gramática abstracta

Spontaneous-transition :: *Transition*

Gramática textual concreta

<spontaneous transition> ::=
input [<virtuality>] <spontaneous designator> <end>
[<enabling condition>] <transition>

<spontaneous designator> ::=
none

Gramática gráfica concreta

<spontaneous transition area> ::=
<input symbol> ***contains***
{ [<virtuality>] <spontaneous designator> }
is followed by
{ [<enabling condition area>] <transition area> }

Semántica

Una transición espontánea permite la activación de una transición sin que ningún estímulo sea presentado al proceso. La activación de una transición espontánea es independiente de la presencia de instancias de señal en el puerto de entrada del proceso. No hay orden de prioridad entre transiciones activadas por la recepción de una señal y transiciones espontáneas.

La expresión **sender** del proceso es **self** después de la activación de una transición espontánea.

Modelo

Una <spontaneous transition> o <spontaneous transition area> que contiene <virtuality> se llama transición espontánea virtual. La transición espontánea virtual se describe en 6.3.3.

2.6.7 Etiqueta

Gramática textual concreta

<label> ::=

<connector name> :

<free action> ::=

connection

<transition>

[**endconnection** [<connector name>] <end>]

<body> se utiliza como un no terminal conveniente para reglas que se aplican a procesos, servicios y procedimientos. <body> no forma parte de la gramática textual concreta.

Todos los <connector name>s definidos en un <body> tienen que ser distintos.

Una etiqueta representa el punto de entrada de una transferencia de control desde las uniones correspondientes con los mismos <connector name>s en el mismo <body>.

Sólo se permite transferencia de control a etiquetas dentro del mismo <body>. La regla para <body>s de un supertipo y sus especializaciones se indica en 6.3.1.

Si la <transition string> de la <transition> en <free action> no está vacía, el primer <action statement> debe tener una <label>, o en caso contrario el <terminator statement> debe tener <label>.

Si está presente, el <connector name> que termina la <free action> debe ser el mismo que el <connector name> en esa <label>.

Gramática gráfica concreta

<in-connector area> ::=

<in-connector symbol> ***contains*** <connector name> ***is followed by***

<transition area>

<in-connector symbol> ::=

Un <in-connector area> representa la continuación de un <flow line symbol> desde un <out-connector area> correspondiente, con el mismo <connector name>, en la misma <process graph area> o <procedure graph area>.

Semántica

La <transition> o <transition area> contenida está en la gramática abstracta representada aplicando <join> al <connector name> (véase 2.6.8.2.2).

2.6.8 Transición

2.6.8.1 Cuerpo de transición

Gramática abstracta

Transition	::	*Graph-node* * (*Terminator* | *Decision-node*)
Graph-node	::	*Task-node* | *Output-node* | *Create-request-node* | *Call-node* | *Set-node* | *Reset-node*
Terminator	::	*Nextstate-node* | *Stop-node* | *Return-node*

Gramática textual concreta

```
<transition> ::=
                    {<transition string> [<terminator statement>] }
              |     <terminator statement>

<transition string> ::=
                    {<action statement>}+

<action statement> ::=
                    [<label>] <action> <end>

<action> ::=
                    <task>
              |     <output>
              |     <create request>
              |     <decision>
              |     <transition option>
              |     <set>
              |     <reset>
              |     <export>
              |     <procedure call>
              |     <remote procedure call>

<terminator statement> ::=
                    [<label>] <terminator> <end>

<terminator> ::=
                    <nextstate>
              |     <join>
              |     <stop>
              |     <return>
```

Si se omite el <terminator> de una <transition>, la última acción en la <transition> debe contener una <decision> de terminación (véase 2.7.5) o una <transition option> de terminación, salvo cuando una <transition> está contenida en una <decision> o una <transition option>.

Gramática gráfica concreta

<transition area> ::=
[<transition string area>] ***is followed by***
{ <state area>
| <nextstate area>
| <decision area>
| <stop symbol>
| <merge area>
| <out-connector area>
| <return area>
| <transition option area> }

<transition string area> ::=
{ <task area>
| <output area>
| <set area>
| <reset area>
| <export area>
| <create request area>
| <procedure call area>
| <remote procedure call area>}
[***is followed by*** <transition string area>]

Una transición consiste en una secuencia de acciones que serán ejecutadas por el proceso.

El <transition area> corresponde a *Transition* y el <transition string area> corresponde a *Graph-node**.

Semántica

Una transición efectúa una secuencia de acciones. Durante una transición los datos de un proceso pueden manipularse y las señales pueden presentarse a la salida. Una transición terminará por la entrada del proceso en un estado, por una parada o por un retorno, o por la transferencia de control a otra transición.

2.6.8.2 Terminador de transición

2.6.8.2.1 Estado siguiente

Gramática abstracta

Nextstate-node :: *State-name*

El *State-name* especificado en un estado siguiente tiene que ser el nombre de un estado dentro del mismo *Process-graph*, *Service-graph* o *Procedure-graph*.

Gramática textual concreta

<nextstate>::=
nextstate <nextstate body>

<nextstate body> ::=
{<state name> | <dash nextstate>}

Gramática gráfica concreta

<nextstate area> ::=
<state symbol> ***contains*** <nextstate body>

Semántica

Un estado siguiente representa un terminador de una transición. Especifica el estado del proceso, servicio o procedimiento cuando termine la transición.

2.6.8.2.2 Unión

Una unión cambia el flujo en un <body> expresando que el <action statement> siguiente a interpretar es el que contiene el mismo <connector name>.

Gramática textual concreta

<join> ::=

join <connector name>

Tiene que haber un <connector name>, y sólo uno, correspondiente a una <join> dentro del mismo <body>. La regla para <process type body> se indica en 6.3.1.

Gramática gráfica concreta

<merge area> ::=
<merge symbol> ***is connected to*** <flow line symbol>

<merge symbol> ::=
<flow line symbol>

<flow line symbol> ::=
———————

<out-connector area> ::=
<out-connector symbol> ***contains*** <connector name>

<out-connector symbol> ::=
<in-connector symbol>

Para cada <out-connector area> en un <process graph area> debe haber una <in-connector area>, y sólo una, en esa <process graph area>. La regla para <process type graph area> se indica en 6.3.1.

Una <out-connector area> corresponde a una <join> en la *Gramática textual concreta*. Si una <merge area> está incluida en una <transition area>, ello equivale a especificar una <out-connector area> en la <transition area>que contiene un <connector name> único y asociar una <in-connector area>, con el mismo <connector name>, al <flow line symbol> en la <merge area>.

Semántica

En la sintaxis abstracta una <join> o una <out-connector area> se derivan de la <transition string> en la cual el primer <action statement> o área de acción tiene asociado el mismo <connector name>.

2.6.8.2.3 Parada

Gramática abstracta

Stop-node :: ( )

Un *Stop-node* no podrá estar contenido en un *Procedure-graph*.

Gramática textual concreta

<stop> ::=

stop

Gramática gráfica concreta

<stop symbol> ::=

Semántica

La parada causa la detención inmediata del proceso o servicio que lo interpreta.

En el caso de un proceso, significa que las señales retenidas en el puerto de entrada se descartan y que las variables y temporizadores creados para el proceso, el puerto de entrada y el proceso dejarán de existir.

En el caso de un servicio, significa que las señales para ese servicio serán descartadas hasta que la instancia de proceso deje de existir.

2.6.8.2.4 Retorno

Gramática abstracta

Return-node :: ( )

n *Return-node* debe estar contenido en un *Procedure-graph*.

Gramática textual concreta

<return> ::=

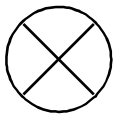
return [<expression>]

Gramática gráfica concreta

<return area> ::=

<return symbol>
[***is associated with*** <expression>]

<return symbol> ::=



<expression> en <return> o <return area> se admite solamente si el ámbito circundante es un operador o un procedimiento que tiene <procedure result>.

Semántica

Un *Return-node* se interpreta de la forma siguiente:

a) Todas las variables creadas por la interpretación del *Procedure-start-node* dejarán de existir.

b) La interpretación del *Procedure-graph* está completa y la instancia de procedimiento deja de existir.

c) En lo sucesivo, la interpretación del proceso, servicio (o procedimiento) llamante continúa en el nodo que sigue a la llamada.

Modelo

El modelo de retorno con una expresión se define en 2.4.6.

2.7 Acción

2.7.1 Tarea

Gramática abstracta

Task-node :: *Assignment-statement* |
Informal-text

Gramática textual concreta

<task> ::=
task <task body>

<task body> ::=
{<assignment statement>{**,**<assignment statement>}*}
| {<informal text> {,<informal text>}*}

Gramática gráfica concreta

<task area> ::=
<task symbol> ***contains*** <task body>

<task symbol> ::=

Semántica

La interpretación de un *Task-node* es la interpretación del *Assignment-statement* o la interpretación del *Informal-text*.

Modelo

Un <task body> puede contener varios <assignment statement>s o <informal text>. En ese caso es sintaxis derivada para especificar unas secuencia de <task>s, una para cada <assignment statement> o <informal text> de modo que se mantenga el orden original en que fueron especificados en el <task body>.

Esta notación taquigráfica se expande antes de expandir notaciones taquigráficas en las expresiones contenidas.

2.7.2 Crear

Gramática abstracta

Create-request-node :: *Process-identifier*
[*Expression*]*

La longitud de [*Expression*]* tiene que ser igual al número de *Process-formal-parameter*s en la *Process-definition* del *Process-identifier*.

Cada *Expression* correspondiente en posición a un *Process-formal-parameter* tiene que tener el mismo género que el *Process-formal-parameter* en la *Process-definition* denotada por el *Process-identifier*.

Gramática textual concreta

<create request> ::=
create <create body>

<create body> ::=
{ <process identifier> | **this** } [<actual parameters>]

<actual parameters> ::=
([<expression>] {,[<expression>]}*)

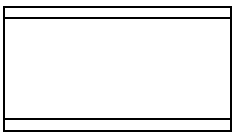
Pueden omitirse las comas después de la última <expression> en <actual parameters>.

this sólo puede especificarse en una <process type definition> y en ámbitos contenidos en una <process type definition>.

Gramática gráfica concreta

<create request area> ::=
<create request symbol> ***contains*** <create body>

<create request symbol> ::=



<create request area> representa un *Create-request-node*.

Semántica

La acción crear causa la creación de una instancia de proceso en el mismo bloque. El **parent** del proceso creado tiene el mismo valor PId que el **self** del proceso creador. Las expresiones **self** del proceso creado y **offspring** del proceso creador tienen ambas el mismo valor nuevo de PId único (véase D.10.1).

Cuando se crea una instancia de proceso se le da un puerto de entrada vacío, se crean variables y las expresiones de parámetros efectivos se interpretan en el orden dado y se asignan (como se define en 5.4.3) a los parámetros formales correspondientes. Seguidamente, el proceso arranca interpretando el nodo de arranque en el gráfico de proceso.

El proceso creado actúa entonces asíncronamente y en paralelo con otros procesos.

Si se intenta crear un número mayor de instancias de proceso que el especificado por el número máximo de instancias en la definición de proceso, no se crea ninguna nueva instancia, la expresión **offspring** del proceso creador tiene el valor Null, y la interpretación continúa.

Si se omite <expression> en <actual parameters>, el parámetro formal correspondiente no tiene valor asociado, es decir, es «indefinido».

Modelo

La indicación **this** es una sintaxis derivada que denota el <process identifier> implícito para el conjunto de instancias del cual es miembro el proceso que ejecuta la creación.

2.7.3 Llamada a procedimiento

Gramática abstracta

Call-node	::	*Procedure-identifier* [*Expression*]*
Procedure-identifier	=	*Identifier*

La longitud de la [*Expression*]* tiene que ser igual al número de los *Procedure-formal-parameter*s en la *Procedure-definition* del *Procedure-identifier*.

Cada *Expression* correspondiente en posición a un *Procedure-formal-parameter* **in** tiene que tener el mismo género que el *Procedure-formal-parameter*.

Cada *Expression* correspondiente por posición a un *Procedure-formal-parameter* **in/out** tiene que ser un *Variable-identifier* con el mismo *Sort-reference-identifier* que el *Procedure-formal-parameter*.

Gramática textual concreta

<procedure call> ::=
call <procedure call body>

<procedure call body> ::=
[**this**] <procedure identifier> [<actual parameters>]

Un ejemplo de <procedure call> se muestra en la Figura 2.10.13.

Una <expression> en <actual parameters> correspondiente a un parámetro **in**/**out** formal no puede omitirse y debe ser un <variable identifier>.

Una vez aplicado el *Modelo* para **this**, <procedure identifier> debe denotar un procedimiento que contiene una transición de arranque.

Si se utiliza **this**, <procedure identifier> debe denotar un procedimiento circundante.

Gramática gráfica concreta

<procedure call area> ::=
<procedure call symbol> ***contains*** <procedure call body>

<procedure call symbol> ::=

<procedure call area> representa el *Call-node.*

Un ejemplo de <procedure call area> se muestra en la Figura 2.10.14.

Semántica

La interpretación de un nodo de llamada a procedimiento transfiere la interpretación a la definición de procedimiento referenciada en el nodo de llamada, y se interpreta ese gráfico de procedimiento.

La interpretación de la transición que contiene la llamada a procedimiento continúa cuando ha terminado la interpretación del proceso llamado.

Las expresiones de parámetros efectivos se interpretan en el orden dado.

Se necesitas semánticas especiales en lo que respecta a la interpretación de datos y parámetros (véase la explicación en 2.4.6).

Si se omite <expression> en <actual parameters>, el parámetro formal correspondiente no tiene valor asociado, es decir, es «indefinido».

Modelo

Si el <procedure identifier> no está definido en el servicio o proceso circundante, la llamada a procedimiento se transforma en una llamada a un subtipo del procedimiento local, implícitamente creado.

this implica que cuando el procedimiento está especializado, el <procedure identifier> es sustituido por el identificador del procedimiento especializado.

2.7.4 Salida

Gramática abstracta

Output-node	::	*Signal-identifier*
		[*Expression*]*
		[*Signal-destination*]
		Direct-via
Signal-destination	=	*Expression* | *Process-identifier*
Direct-via	=	(*Signal-route-identifier* | *Channel-identifier*)-**set**

La longitud de la [*Expression*]* tiene que ser igual al número de *Sort-reference-identifier*s en la *Signal-definition* denotada por el *Signal-identifier*.

Cada *Expression* tiene que ser el mismo género que la *Sort-identifier-reference* correspondiente (por posición) en la *Signal-definition*.

Para cada subconjunto coherente posible (véase 3) tiene que existir por lo menos un trayecto de comunicación (ya sea implícito para el propio tipo de proceso, ya sea explícito vía rutas de señales y, posiblemente, canales) hacia el entorno o hacia un conjunto de instancias de proceso o un servicio que tenga *Signal-identifier* en su conjunto de señales de entrada válidas y que se origine a partir del conjunto de instancias de proceso o el servicio en que se utilizó el *Output-node*.

Para cada *Signal-route-identifier* en *Direct-via*, el *Origin* en (uno de) los *Signal-route-path*(s) de la ruta de señales debe denotar:

a) un conjunto de instancias de proceso que contiene el proceso que interpreta el *Output-node* o que contiene un servicio que interpreta el *Output-node*, o

b) un servicio dentro del proceso originador que interpreta el *Output-node*.

Además, el *Signal-route-path* debe incluir *Signal-identifier* en su conjunto *Signal-identifier*s. Si *Origin* denota un conjunto de instancias de proceso y la señal es enviada desde un servicio, debe existir dentro del proceso una ruta de señales capaz de transmitir la señal del servicio emisor a la ruta de señales mencionada en *Direct-via*.

Para cada *Channel-identifier* en *Direct-via* debe existir una o dos rutas de señales y cero o más canales de modo que el canal por ese trayecto pueda alcanzarse con *Signal-identifier* desde el proceso o servicio, y el *Channel-path* procedente del proceso debe incluir *Signal-identifier* en su conjunto de *Signal-identifier*s.

Gramática textual concreta

<output> ::=
output <output body>

<output body> ::=
<signal identifier>
[<actual parameters>]{**,** <signal identifier> [<actual parameters>]}*
[**to** <destination>] [**via** [**all**] <via path>]

<destination> ::=
<PId expression> | <process identifier> | **this**

<via path> ::=
<via path element> {, <via path element>}*

<via path element> ::=
<signal route identifier>
| <channel identifier>
| <gate identifier>

<PId expression> o el <process identifier> en <destination> representa la *Signal-destination*. Hay una ambigüedad sintáctica entre <PId expression> y <process identifier> en <destination>. Si <destination> puede interpretarse como una <PId expression> sin violar ninguna condición estática, se interpreta como <PId expression>, o si no, como <process identifier>. <process identifier> debe denotar un proceso, que puede alcanzarse desde el proceso originador.

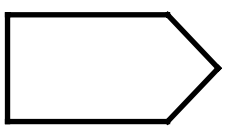
El constructivo **via** representa el *Direct-via*. <via path> se considera una lista de <via path element>s.

via all sólo se puede especificar cuando no hay *Signal-destination*. **via all** representa la multidistribución explicada en *Modelo*.

this sólo se puede especificar en una <process type definition> y en ámbitos circundados por <process type definition>.

Gramática gráfica concreta

<output area> ::=
<output symbol> ***contains*** <output body>
<output symbol> ::=
<plain output symbol>
| <internal output symbol>
<plain output symbol> ::=



Semántica

Process-identifier en *Signal-destination* indica que *Signal-destination* es cualquier instancia existente del conjunto de instancias de proceso indicadas por *Process-identifier*. Si no hay instancias, se descarta la señal.

Si ningún *Signal-route-identifier* está especificado en *Direct-via* y ningún *Signal-destination* está especificado, cualquier proceso para el cual existe un trayecto de comunicación puede recibir la señal.

Los valores transportados por la instancia de señal son los valores de los parámetros reales en la salida. Si se omite <expression> en <actual parameters>, ningún valor es transportado con el lugar correspondiente de la instancia de señal, es decir, el lugar correspondiente es «indefinido».

El valor PId del proceso originador también es transportado por la instancia de señal.

Si se especifica un sintipo en la definición de señal y se especifica una expresión en la salida, se aplica a la expresión la verificación de intervalo definida en 5.3.1.9.1.

La instancia de señal se pasa entonces a un trayecto de comunicación capaz de transportarla. El conjunto de trayectos de comunicación capaces de transportar la instancia de señal puede limitarse, mediante la cláusula **via,** al conjunto de trayectos mencionados en *Direct-via*.

Si *Signal-destination* es *Expression*, la instancia de señal se entrega a la instancia de proceso denotada por *Expression*. Si esta instancia no existe o no puede alcanzarse desde el proceso originador, se descarta la instancia de señal.

Si *Signal-destination* es *Process-identifier*, la instancia de señal se entrega a una instancia arbitraria del conjunto de instancias de proceso denotado por *Process-identifier*. Si no existe esa instancia, se descarta la instancia de señal.

Por ejemplo, si *Signal-destination* está especificada como Nula en *Output-node*, la instancia de señal se descarta cuando se interprete el *Output-node*.

Si no está especificado ningún *Signal-destination*, el receptor se selecciona en dos pasos. En el primero, la señal se envía a un conjunto de instancias de proceso, al que se puede llegar por los trayectos de comunicación que pueden transportar la instancia de señal. Este conjunto de instancias de proceso se elige arbitrariamente. En el segundo paso, cuando la instancia de señal llega al final del trayecto de comunicación, es entregada a una instancia arbitraria del conjunto de instancias de proceso. La instancia se selecciona arbitrariamente. Si no se puede seleccionar ninguna instancia, se descarta la instancia de señal.

Obsérvese que el hecho de que se especifique el mismo *Channel-identifier* o *Signal-route-identifier* en *Direct-via* de dos *Output-node*s no significa automáticamente que las señales están en cola en el puerto de entrada en el mismo orden en que se interpretan los *Output-node*s. Sin embargo, este orden se mantiene si las dos señales son transportadas por canales de retardo idénticos, o solamente transportadas por canales sin retardo, o si el proceso o servicio originador o el proceso o servicio de destino se definen en el mismo bloque.

Modelo

Si se especifican varios pares de (<signal identifier> <actual parameters>) en un <output body>, esto es sintaxis derivada para especificar una secuencia de <output>s o <output area>s en el mismo orden especificado en el <output body> inicial, cada uno de los cuales contiene un solo par de (<signal identifier> <actual parameters>). La cláusula **to** y la cláusula **via** se repiten en cada una de las <output>s o <output area>s.

La indicación **via all** es sintaxis derivada para la multidistribución de la señal por los trayectos de comunicación mencionados en <via path>, de modo que las señales se envían en el orden en que los <via path element>s aparecen en <via path>, una por cada <via path element>. La multidistribución denota una secuencia de salidas de la misma señal. Los valores transportados por cada una de las instancias de señal resultantes sólo se evalúan una vez antes de interpretar la primera salida, después se utilizan variables implícitas para almacenar los valores que se utilizarán en cada salida. Si el mismo <via path element> aparece varias veces en un <via path>, se envía una señal para cada aparición.

La indicación **this** es sintaxis derivada para denotar como <process identifier>, el <process identifier> implícito para el conjunto de instancias del cual es miembro el proceso que ejecuta la salida.

2.7.5 Decisión

Gramática abstracta

Decision-node	::	*Decision-question* *Decision-answer*-**set** [*Else-answer*]
Decision-question	=	*Expression* | *Informal-text*
Decision-answer	::	(*Range-condition* | *Informal-text*) *Transition*
Else-answer	::	*Transition*

*Range-condition*s de *Decision-answer*s deben ser mutuamente excluyentes, y *Ground-Expression*s de *Range-condition*s deben ser del mismo género.

Si la *Decision-question* es una *Expression*, la *Range-condition* de las *Decision-answer*s tiene que ser del mismo género que la *Decision-question*.

Gramática textual concreta

<decision> ::=
decision <question> <end> <decision body> **enddecision**

<decision body> ::=
{<answer part> <else part>}
| {<answer part> {<answer part>}$^{+}$ [<else part>] }

<answer part> ::=
([<answer>]) : [<transition>]

<answer> ::=
<range condition> | <informal text>

<else part> ::=
else : [<transition>]

<question> ::=
<question expression> | <informal text> | **any**

Una <decision> o <transition option> es una decisión de terminación y una opción de terminación, respectivamente, si cada <answer part> y <else part> en el <decision body> contiene una <transition> en que se especifica una <terminator statement>, o contiene una <transition string> cuyo último <action statement> contiene una decisión de terminación o una opción.

La <answer> de <answer part> debe omitirse si <question> consta de la palabra clave **any**. En este caso ninguna <else part> puede estar presente.

Hay una ambigüedad sintáctica entre <informal text> y <character string> en <question> y <answer>. Si la <question> y todas las <answer>s son <character string>, todas éstas se interpretan como <informal text>. Si la <question> o alguna <answer> es una <character string> que no encaja en el contexto de la decisión, la <character string> denota <informal text>.

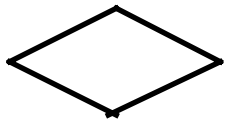
El contexto de la decisión (es decir, el género) se determina sin considerar <answer>s que son <character string>.

Si <character string> de <question> o cualquier <answer> contiene caracteres de control, la cadena se interpreta como <informal text>.

Gramática gráfica concreta

<decision area> ::=
<decision symbol> ***contains*** <question>
is followed by
{ {<graphical answer part> <graphical else part>} ***set***
| {<graphical answer part> {<graphical answer part>}+
[<graphical else part>] } ***set*** }

<decision symbol> ::=



<graphical answer> ::=
[<answer>] | ([<answer>])

<graphical answer part> ::=
<flow line symbol> ***is associated with*** <graphical answer>
is followed by <transition area>

<graphical else part> ::=
<flow line symbol> ***is associated with*** **else**
is followed by <transition area>

La <graphical answer> y **else** pueden situarse a lo largo del <flow line symbol> asociado, o en el <flow line symbol> interrumpido.

Los <flow line symbol>s que se originan en un <decision symbol> pueden tener un trayecto de origen común.

Un <decision area> representa un *Decision-node*.

<answer> de <graphical answer> sólo debe omitirse si <question> consiste en la palabra clave **any**. En este caso, ninguna <graphical else part> puede estar presente.

Semántica

Una decisión transfiere la interpretación al trayecto de salida cuya condición de intervalo contiene el valor dado por la interpretación de la pregunta. Se define un conjunto de respuestas posibles a la pregunta, cada una de las cuales especifica el conjunto de acciones a interpretar para esa elección de trayecto.

Una de las respuestas puede ser el complemento de las otras. Esto se consigue especificando la *Else-answer*, que indica el conjunto de acciones a realizar cuando el valor de la expresión sobre la cual se plantea la pregunta no está cubierto por los valores o el conjunto de valores especificados en las otras respuestas.

En todos aquellos casos en que no se especifica la *Else-answer*, el valor resultante de la evaluación de expresión de pregunta debe corresponder con una de las respuestas.

Modelo

Si una <decision> no es de terminación, es entonces sintaxis derivada para una <decision> en la cual todas las <answer part>s y la <else part> han insertado en su <transition> una <join> con el primer <action statement> que sigue a la decisión o, si la decisión es el último <action statement> en una <transition string>, con el siguiente <terminator statement>.

La utilización de **any** solamente en una <decision> es una notación taquigráfica para utilizar <anyvalue expression> en la decisión. Suponiendo que <decision body> va seguido por N <answer part>s, **any** en <decision> es entonces una notación taquigráfica para escribir **any**(data_type_N), donde data_type_N es un sintipo anónimo definido como:

```
syntype data_type_N =
   package Predefined Integer constants 1:N
endsyntype;
```

Las <answer>s omitidas son notaciones taquigráficas para escribir los literales 1 a N como las <constant>s de <range condition>s en N <answer>s.

2.8 Temporizador

Gramática abstracta

Timer-definition	::	*Timer-name* *Sort-reference-identifier**
Timer-name	=	*Name*
Set-node	::	*Time-expression* *Timer-identifier* *Expression**
Reset-node	::	*Timer-identifier* *Expression**
Timer-identifier	=	*Identifier*
Time-expression	=	*Expression*

Los géneros de la *Expression** en el *Set-node* y *Reset-node* deben corresponder en posición con el *Sort-reference-identifier** que sigue directamente al *Timer-name* identificado por el *Timer-identifier*.

Gramática textual concreta

<timer definition> ::=
timer
<timer definition item> {,<timer definition item>}* <end>

<timer definition item> ::=
<timer name> [<sort list>] [:= <Duration ground expression>]

<reset> ::=
reset (<reset statement> { , <reset statement> }*)

<reset statement> ::=
<timer identifier> [(<expression list>)]

<set> ::=
set <set statement> { , <set statement> }*

<set statement> ::=
([<Time expression> ,] <timer identifier> [(<expression list>)])

Un <set statement> puede omitir <Time expression>, si <timer identifier> denota un temporizador que tiene <Duration ground expression> en su definición.

Un <reset statement> representa un *Reset-node*; un <set statement> representa un *Set-node*.

Gramática gráfica concreta

<set area> ::=
<task symbol> ***contains*** <set>

<reset area> ::=

<task symbol> ***contains*** <reset>

Semántica

Una instancia de temporizador es un objeto que puede estar activo o inactivo. Dos ocurrencias de un identificador de temporizador seguido por una lista de expresiones se refieren a la misma instancia de temporizador únicamente si el operador «=» aplicado a todas las expresiones correspondientes en la lista da el valor verdadero (es decir, las dos listas de expresiones tienen los mismos valores).

Cuando un temporizador inactivo es inicializado, se le asocia un valor Tiempo. Si este temporizador no es reinicializado, o si no es inicializado de nuevo, antes de que el tiempo de sistema llegue a este valor Tiempo, se aplica al puerto de entrada del proceso una señal con el mismo nombre que el temporizador. La misma acción se efectúa si el temporizador es inicializado con un valor Tiempo inferior o igual a **now**. Después del consumo de una señal de temporizador, la expresión **sender** da el mismo valor que la expresión **self**. Si se da una lista de expresiones cuando el temporizador está inicializado, los valores de estas expresiones (o expresión) están contenidos en la señal de temporizador en el mismo orden. Un temporizador está activo desde el momento de la inicialización hasta el momento del consumo de la señal de temporizador.

Si un género especificado en una definición de temporizador es un sintipo, la verificación de intervalo definida en 5.3.19.1 aplicada a la expresión correspondiente en una inicialización o reinicialización debe ser Verdadero; de no ser así el sistema está en error y su comportamiento ulterior está indefinido.

Cuando un temporizador inactivo es reinicializado, sigue estando inactivo.

Cuando un temporizador activo es reinicializado, la asociación con el valor Tiempo se pierde; si hay una señal de temporización correspondiente retenida el puerto de entrada se suprime y el temporizador pasa a inactivo.

La inicialización de un temporizador activo equivale a reinicializarlo e inicializarlo inmediatamente después. Entre los instantes de reinicialización e inicialización del temporizador, éste permanece activo.

Una instancia de temporizador está inactiva antes de ser inicializada por primera vez.

*Expression*s en *Set-node* o *Reset-node* se evalúan en el orden dado.

Modelo

Un <set statement> sin <Time expression> es sintaxis derivada para <set statement> en el cual <Time expression> es «**now** + <Duration ground expression>», donde <Duration ground expression> se deriva de la definición del temporizador.

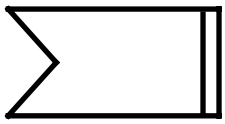
Un <reset> o un <set> puede contener varios <reset statement>s o <set statement>s respectivamente. Esto es sintaxis derivada para especificar una secuencia de <reset>s o <set>s, uno para cada <reset statement> o <set statement>, de modo que se mantenga el orden original en el cual se han especificado en <reset> o <set>. Esta notación taquigráfica se amplía antes de que se amplíen las notaciones taquigráficas en las expresiones contenidas.

2.9 Entrada y salida internas

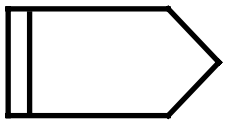
Los símbolos definidos en esta subcláusula se introducen para la compatibilidad con los diagramas existentes. No se recomiendan para nuevas descripciones SDL y tienen la misma semántica que <plain input symbol> y <plain output symbol> respectivamente.

Gramática gráfica concreta

<internal input symbol> ::=



<internal output symbol> ::=



2.10 Ejemplos

```
task t3 := 0/*example*/;
```

FIGURE 2.10.1/Z.100

Exemple de note (SDL/PR)

```
---------
task 'task1' comment 'example';
task 'task2';
---------
```

FIGURE 2.10.2/Z.100

Exemple de commentaire (SDL/PR)

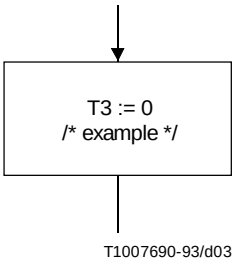


FIGURA 2.10.3/Z.100

Mismo ejemplo que la Figura 2.10.1 en SDL/GR

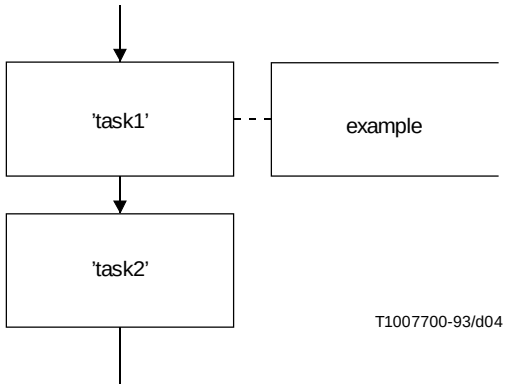


FIGURA 2.10.4/Z.100

Mismo ejemplo que la Figura 2.10.2 en SDL/GR

```
system Daemongame;

     signal Newgame, Probe, Result, Endgame, Gameid, Win, Lose,
        Score (Integer);
     channel Gameserver.in
        from env to Game
        with Newgame, Probe, Result, Endgame;
     endchannel Gameserver.in;
     channel Gameserver.out
        from Game to env
        with Gameid, Win, Lose, Score;
     endchannel Gameserver.out;
     block Game referenced;
endsystem Daemongame;
```

FIGURA 2.10.5/Z.100

Ejemplo de especificación de sistema (SDL/PR)

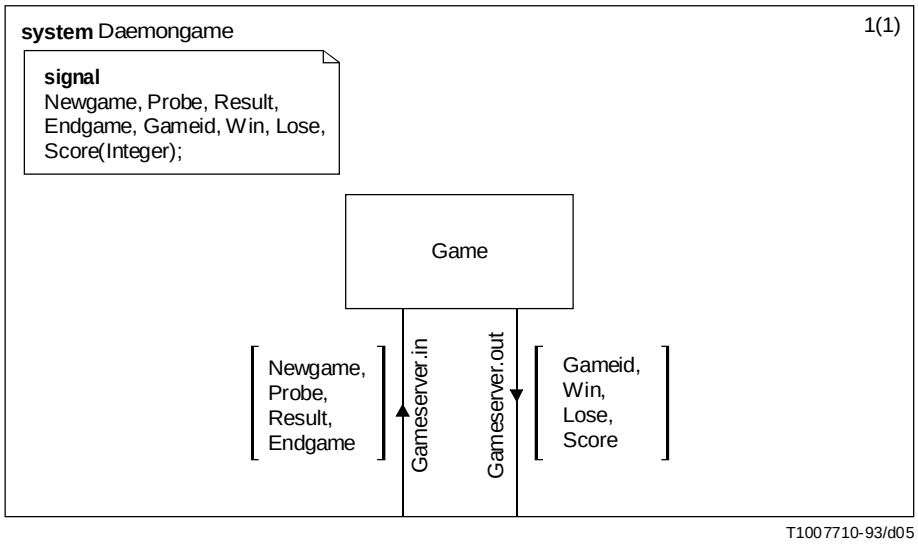


FIGURA 2.10.6/Z.100

Ejemplo de especificación de sistema (SDL/GR)

```
block Game;
     signal Gameover(PId);
     connect Gameserver.in and R1,R2;
     connect Gameserver.out and R3;
     signalroute R1 from env to Monitor with Newgame;
     signalroute R2 from env to Game with Probe, Result, Endgame;
     signalroute R3 from Game to env
        with Gameid, Win, Lose, Score;
     signalroute R4  from Game to Monitor with Gameover;

     process Monitor (1,1) referenced;

     process Game (0,) referenced;
endblock Game;
```

FIGURA 2.10.7/Z.100

Ejemplo de especificación de bloque (SDL/PR)

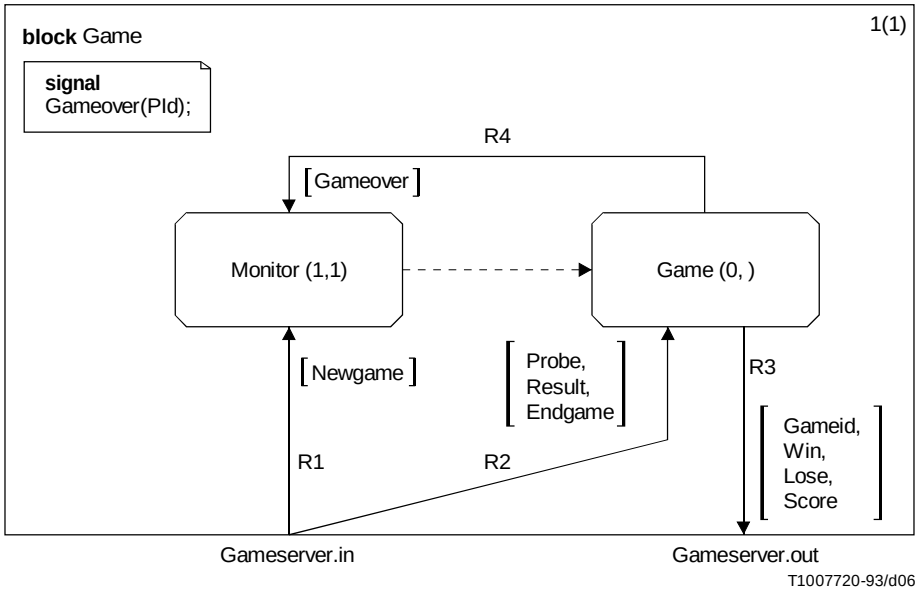


FIGURA 2.10.8/Z.100

Ejemplo de especificación de bloque (SDL/GR)

```
process Game (0, ); fpar player Pid;

dcl count Integer := 0 ; /*counter to keep track of score */

start;
     output Gameid to player;
        nextstate even;

state even;
     input none;
        nextstate odd;
  input Probe;
        output Lose to player;
        task count:= count-1;
        nextstate -;

state odd;
  input Probe;
        output Win to player;
        task count:= count+1;
        nextstate -;
  input none;
        nextstate even;

state even, odd;
  input Result;
        output Score(count) to player;
        nextstate -;
  input Endgame;
        output Gameover(player);
        stop;

endprocess Game;
```

FIGURA 2.10.9/Z.100

Ejemplo de especificación de proceso (SDL/PR)

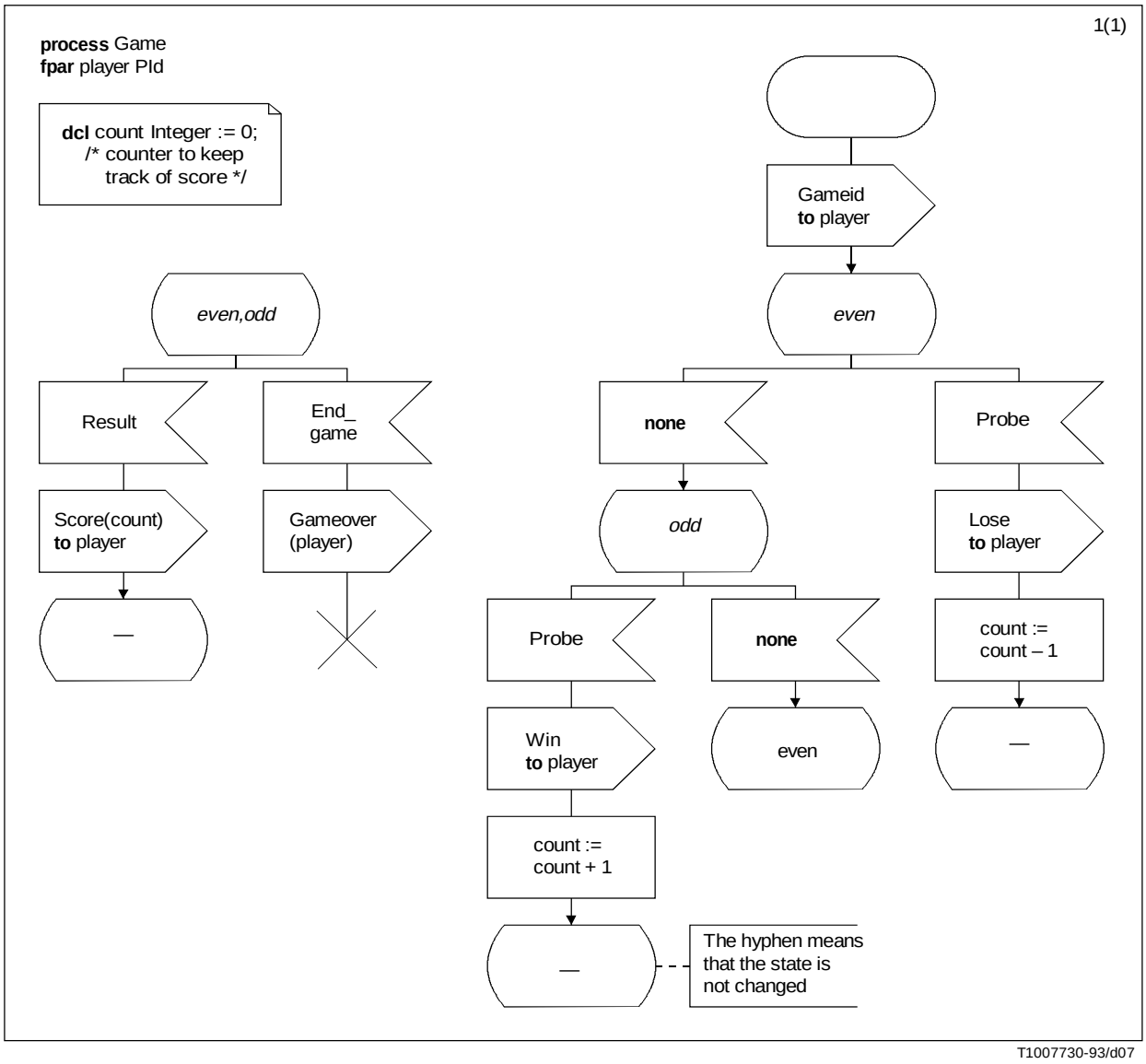


FIGURA 2.10.10/Z.100

Ejemplo de especificación de proceso (SDL/GR)

```
procedure Check;
     fpar in/out x, y Integer;
     /* The following signal definitions are assumed:
     signal sig1(Boolean), sig2, sig3(Integer,PId);    */
     dcl  sum, index Integer,
          nice Boolean;
     start;
          task sum := 0, index := 1;
        nextstate idle;
     state idle;
        input sig1(nice);
        decision nice;
          (True): task 'Calculate sum';
            output sig3(sum, sender);
            return;
          (False): nextstate problems;
        enddecision;
        input sig2;
        join 1;
     .........
endprocedure Check;
```

FIGURA 2.10.11/Z.100

Ejemplo de un fragmento de una especificación de proceso (SDL/PR)

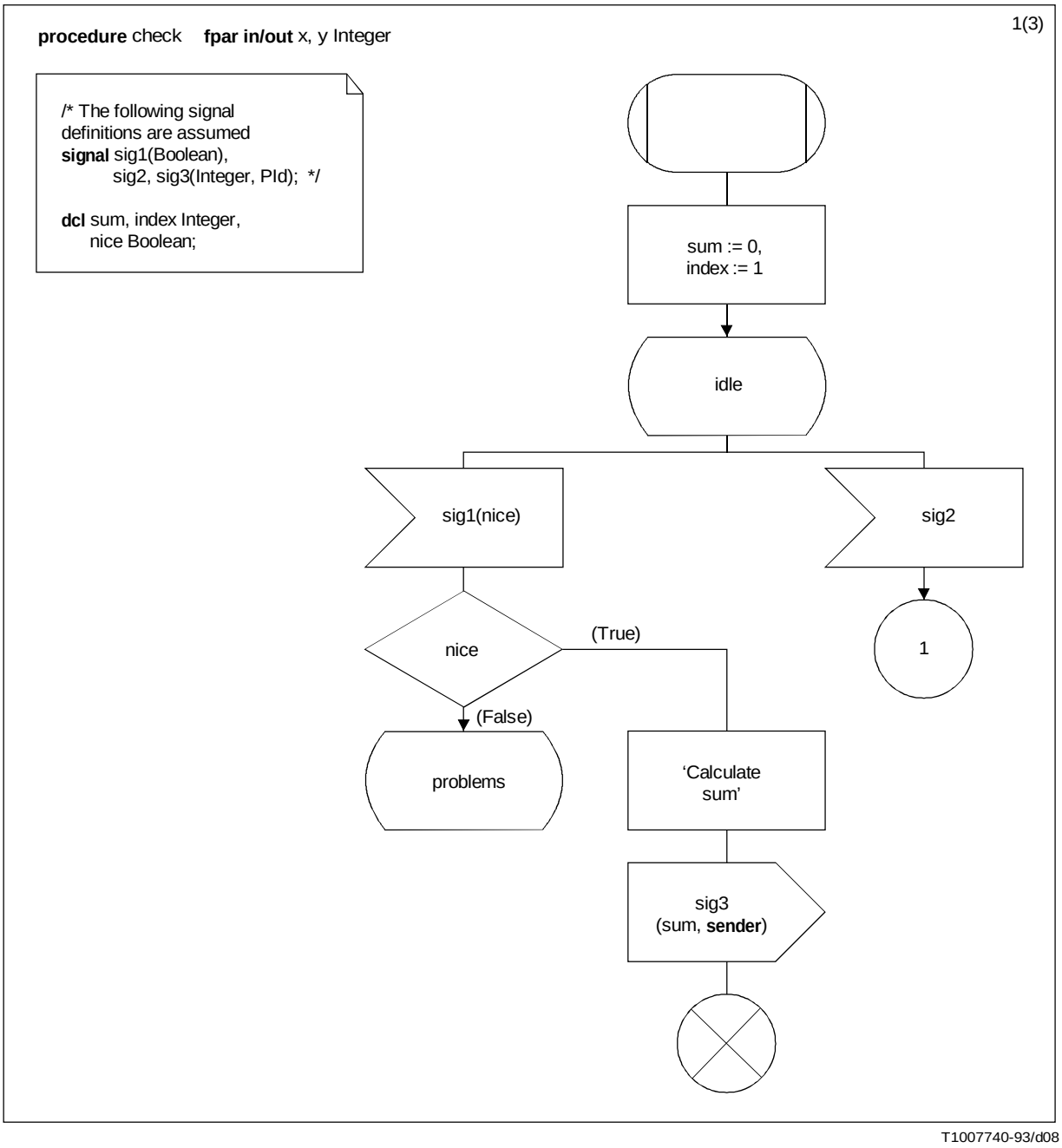


FIGURA 2.10.12/Z.100

Ejemplo de un fragmento de una especificación de procedimiento (SDL/GR)

```
/* The following signal definition is assumed:
signal inquire(Integer,Integer,Integer);  */
process alfa;
     dcl a,b,c Integer;
        .....
        ......
     input inquire (a,b,c);
        call check (a,b);
     ......
endprocess;
```

FIGURA 2.10.13/Z.100

Ejemplo de llamada a procedimiento en un fragmento de una definición de proceso (SDL/PR)

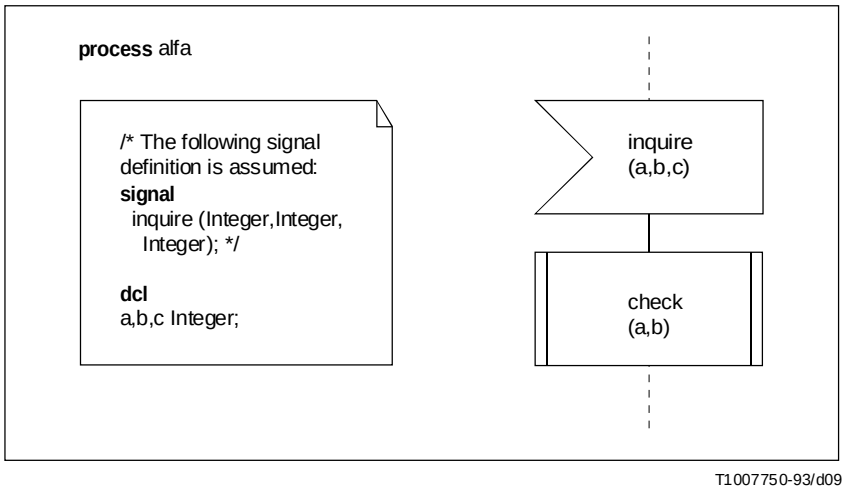


FIGURA 2.10.14/Z.100

Ejemplo de llamada a procedimiento en un fragmento de una definición de proceso (SDL/GR)

En las Figuras 2.10.15 a 2.10.20 se da un ejemplo de <process definition> que contiene <service definition>s, con las correspondientes <service definition>s. Este proceso tiene el mismo comportamiento que el indicado en las Figuras 2.10.9 y 2.10.10.

```
process Game;
     fpar player Pid;
     signal Proberes (Integer);

     signalroute IR1 from Game_handler to env with Score,Gameid;
     signalroute IR2 from Game_handler to env with Gameover;
     signalroute IR3 from env to Game_handler with Result,Endgame;
     signalroute IR4 from env to Probe_handler with Probe;
     signalroute IR6 from Probe_handler to env with Lose,Win;
     signalroute IR7 from Probe_handler to Game_handler with Proberes;

     connect R2 and IR3,IR4;
     connect R3 and IR1,IR6;
     connect R4 and IR2;

     service Game_handler referenced;
     service Probe_handler referenced;

endprocess Game;
```

FIGURA 2.10.15/Z.100

Ejemplo de proceso (igual a la Figura 2.10.9) descompuesto en servicios (SDL/PR)

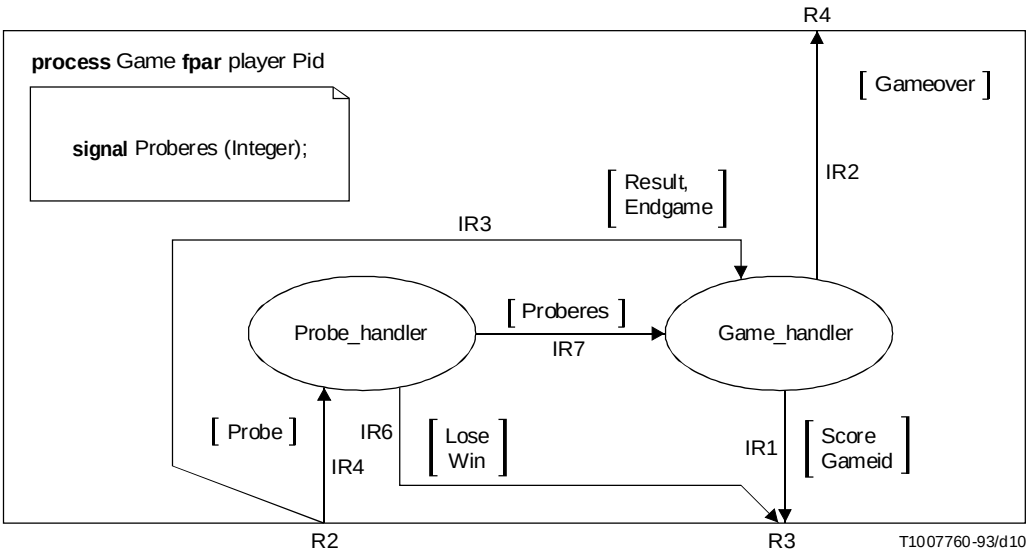


FIGURA 2.10.16/Z.100

Ejemplo de proceso (igual a la Figura 2.10.10) descompuesto en servicios (SDL/GR)

```
service Game_handler;

     /* The service handles a game with actions to start a game, to end
     a game, to keep track of the score and to communicate the score */

     dcl  count Integer,    /*Counter to keep track of the score*/
          a Integer;        /* counter increment */

     start;
        output Subscr;
        output Gameid to player;
        task count:=0;
        nextstate started;
     state started;
        priority input Proberes(a);
           task count:=count+a;
           nextstate -;
        input Result;
           output Score(count) to player;
           nextstate -;
         input Endgame;
           output Gameover;
           stop;
     endstate started;
endservice Game_handler;
```

FIGURA 2.10.17/Z.100

Ejemplo de servicio (SDL/PR)

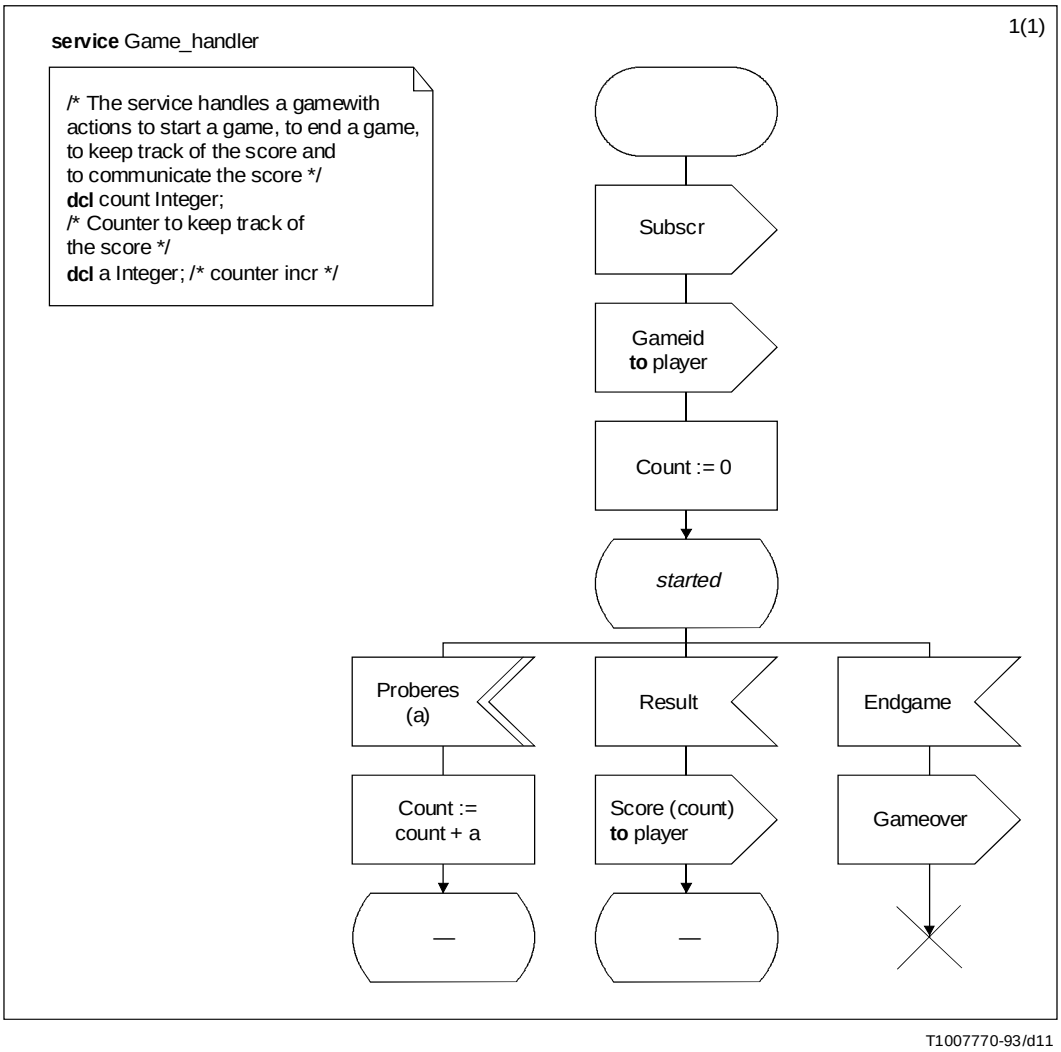


FIGURA 2.10.18/Z.100

Ejemplo de servicio (igual a la Figura 2.10.17) (SDL/PR)

```
service Probe_handler;

/* The service has actions to register the bumps and to handle probes from the player.
     The probe result is sent to the player but also to the service Game_handler */
     start;
        nextstate even;
     state even;
        input Probe;
           output Lose to player;
           output Proberes(-1) to self;
           nextstate -;
        input none;
           nextstate Odd;
     endstate even;
     state odd;
        input none;
           nextstate even;
        input Probe;
           output Win to player;
           output Proberes(1) to self;
           nextstate -;
     endstate Odd;
endservice Bump_handler;
```

FIGURA 2.10.19/Z.100

Ejemplo de servicio (SDL/PR)

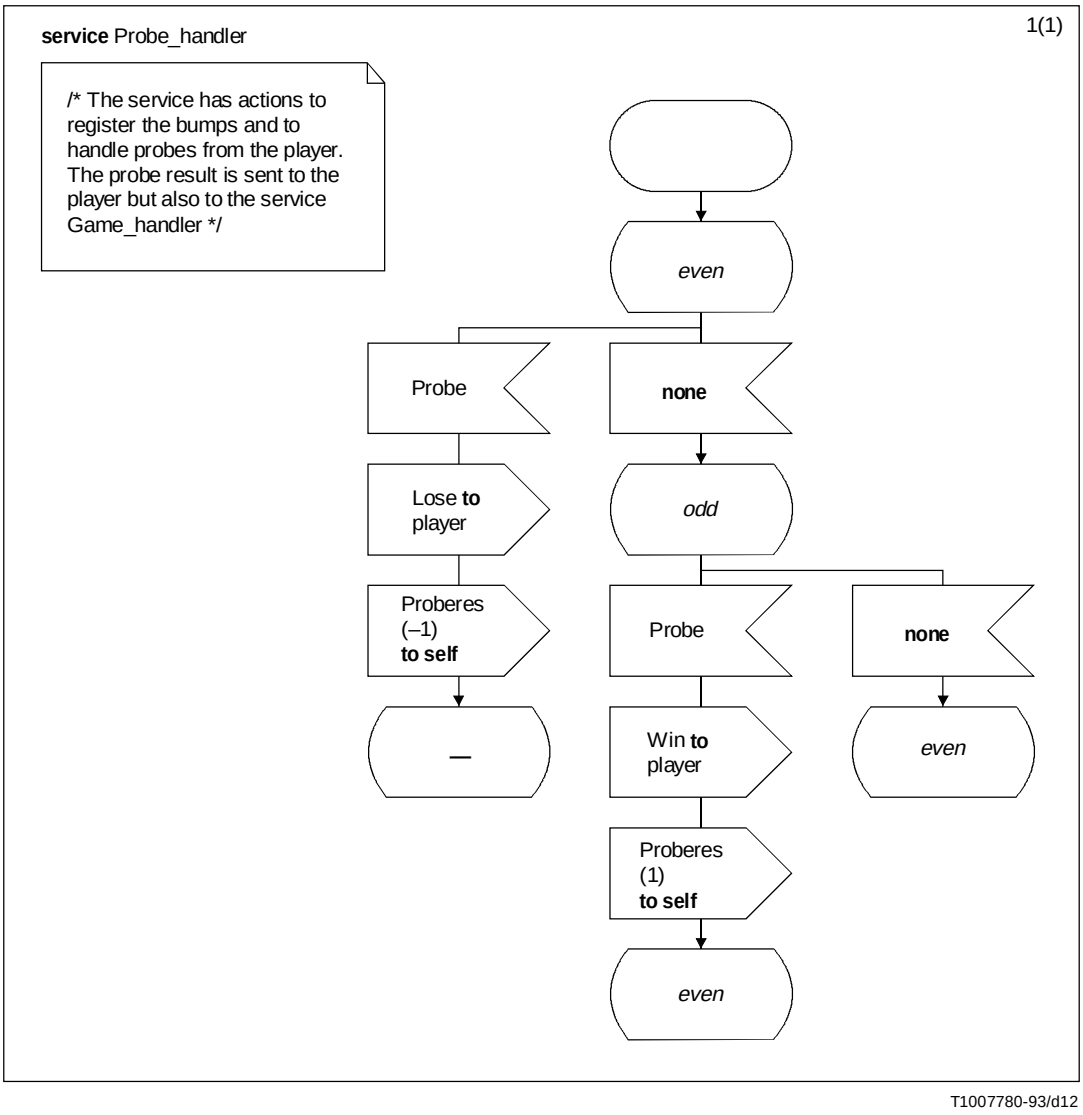


FIGURA 2.10.20/Z.100

Ejemplo de servicio (igual a la Figura 2.10.19) (SDL/GR)

3 Conceptos de descomposición estructural en SDL

3.1 Introducción

Esta cláusula define cierto número de conceptos necesarios para manejar estructuras jerárquicas en SDL. La base de estos conceptos se define en 2 y los conceptos definidos son adiciones estrictas a los definidos en 2.

Los conceptos introducidos en esta cláusula tienen por finalidad proporcionar al usuario del SDL medios para especificar sistemas grandes y/o complejos. Los conceptos definidos en 2 son adecuados para especificar sistemas relativamente pequeños que pueden ser comprendidos y manejados a un nivel único de bloques. Cuando se especifican sistemas grandes, o complejos, es necesario particionar la especificación del sistema en unidades manejables, que puedan ser tratadas y comprendidas independientemente. A menudo conviene efectuar la partición en varios pasos con lo que se obtiene una estructura jerárquica de unidades que especifican el sistema. Además de los conceptos presentados en esta cláusula, el concepto de paquete presentado en 2 y los conceptos de tipificación estructural presentados en 6 son particularmente útiles para describir sistemas grandes.

El término partición significa división de una unidad en subunidades más pequeñas, que son sus componentes. La partición no afecta a la interfaz estática de una unidad. Además de efectuar la partición, es necesario también añadir nuevos detalles al comportamiento de un sistema cuando se desciende a niveles inferiores en la estructura jerárquica de la especificación del sistema. Esto se denota por el término refinamiento.

3.2 Partición

3.2.1 Generalidades

Un bloque puede ser dividido en un conjunto de subbloques, canales y subcanales. De manera similar, un canal puede ser dividido en un conjunto de bloques, canales y subcanales. En las sintaxis concretas, cada definición de bloque (o tipo de bloque) y definición de canal puede tener dos versiones: una versión no dividida y una versión dividida. Sin embargo, las subestructuras de canal son transformadas cuando se establece una relación de correspondencia con la sintaxis abstracta, es decir, un canal con una subestructura nunca se interpreta, pero sí se interpreta la subestructura de canal. Una definición de subbloque es una definición de bloque, y una definición de subcanal es una definición de canal.

En la definición de un sistema concreto, así como en la definición de un sistema abstracto, puede aparecer tanto la versión no dividida como la versión dividida de una definición de bloque. En este caso, una definición de sistema concreto contiene varios subconjuntos de partición coherentes, correspondiendo cada subconjunto a una instancia de sistema. Un subconjunto de partición coherente, es una selección de las *block definitions* en una *system definition* de tal modo que:

a) Si contiene una *Block-definition*, tiene entonces que contener la definición de la unidad de ámbito envolvente, si existe.

b) Debe contener todas las *Block-definition*s definidas al nivel del sistema y, si contiene una *Sub-block-definition* de una *Block-definition*, tendrá entonces que contener todas las otras *Sub-block-definition*s de esa *Block-definition*.

c) Todas las *Block-definition*s «constitutivo de hoja» en la estructura resultante contienen *Process-definition*s.

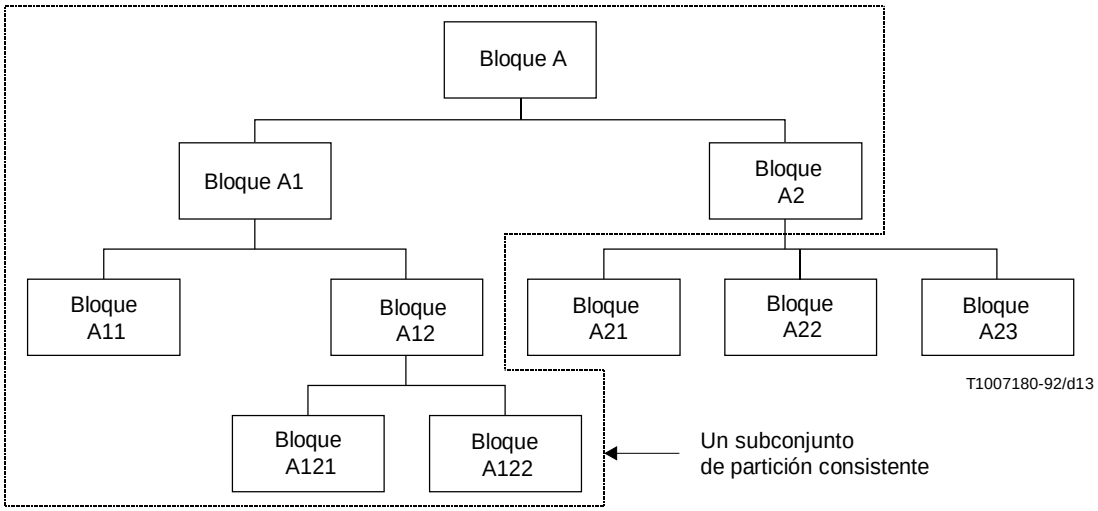


FIGURA 3.2.1/Z.100

Subconjunto de partición coherente ilustrado en un diagrama auxiliar

En el momento de la interpretación del sistema, se interpreta un subconjunto de partición coherente. Se interpretan los procesos en cada uno de los bloques constitutivos de hoja en el subconjunto de partición coherente. Si estos bloques constitutivos de hoja contienen también subestructuras, no producen efecto alguno. Las subestructuras en los bloques no constitutivos de hoja producen efecto sobre la visibilidad, y los procesos en estos bloques no se interpretan. Véase la Figura 3.2.1.

3.2.2 Partición de bloque

Gramática abstracta

Block-substructure-definition	::	*Block-substructure-name* *Sub-block-definition*-**set** *Channel-connection*-**set** *Channel-definition*-**set** *Signal-definition*-**set** *Data-type-definition* *Syntype-definition*-**set**
Block-substructure-name	=	*Name*
Sub-block-definition	=	*Block-definition*
Channel-connection	::	*Channel-identifier*-**set** *Sub-channel-identifier*-**set**
Sub-channel-identifier	=	*Channel-identifier*
Channel-identifier	=	*Identifier*

La *Block-substructure-definition* tiene que contener por lo menos una *Sub-block-definition*. A menos que se indique otra cosa, queda entendido que, en lo sucesivo, un término de sintaxis abstracta está contenido en la *Block-substructure-definition*.

Un *Block-identifier* contenido en una *Channel-definition* tiene que denotar una *Sub-block-definition*. Una *Channel-definition* que conecta una *Sub-block-definition* a la frontera de la *Block-substructure-definition* se denomina definición de subcanal.

Cada *Channel-definition* en la definición de bloque circundante que está conectado a una *Block-substructure-definition* debe ser miembro de una sola *Channel-connection*. Los *Channel-identifier*s en la *Channel-connection* deben identificar

*Channel-definition*s en la definición de bloque circundante. Cada *Sub-channel-identifier* debe aparecer en una sola *Channel-connection*.

Para señales que salen de la *Block-substructure-definition*, la unión de los *Signal-identifier*-**set**s asociados al *Sub-channel-identifier*-**set** contenido en una *Channel-connection* debe ser idéntica a la de los *Signal-identifier*-**set**s asociados al *Channel-identifier*-***set*** contenido en la *Channel-connection*. La misma regla es válida para las señales que entran en la *Block-substructure-definition*. Sin embargo, esta regla se modifica en el caso de refinamiento de señal (véase 3.3).

Puesto que una *Sub-block-definition* es una *Block-definition*, puede ser fraccionada. Esta partición puede repetirse cualquier número de veces, de lo que resulta una estructura jerárquica en árbol de *Block-definitions* y sus *Sub-block-definitions*. Se dice que las *Sub-block-definitions* de una *Block-definition* existen en el nivel inmediato inferior del árbol de bloques, véase también la Figura 3.2.2.

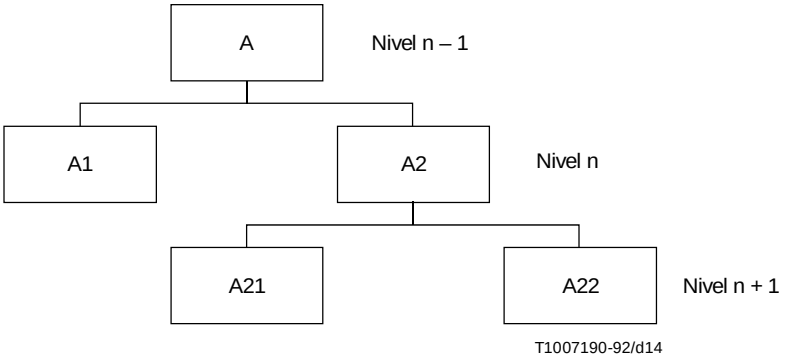


FIGURA 3.2.2/Z.100

Diagrama de árbol de bloques

El diagrama de árbol de bloques es un diagrama auxiliar.

Gramática textual concreta

<block substructure definition> ::=
substructure
{[<block substructure name>]
| <block substructure identifier> } <end>
{ <entity in system> | <channel connection> }+
endsubstructure
[{ <block substructure name> | <block substructure identifier>}] <end>

Si se omite el <block substructure name> después de la palabra clave **substructure,** es igual que el nombre de la <block definition> o <block type definition> circundante.

<textual block substructure reference> ::=
substructure <block substructure name> **referenced** <end>

<channel connection> ::=
connect <channel identifiers>
and <subchannel identifiers> <end>

<subchannel identifiers> ::=
<channel identifiers>

Si una <block substructure definition> contiene <channel definition>s y <textual typebased block definition>s, cada puerta de las <block type definition>s de las <textual typebased block definition>s debe estar conectada a un canal.

Para una <block substructure definition> en una <block type definition>, no pueden darse <channel connection>s. Estas se derivan para las <textual typebased block definition>s resultantes definidas en 6.1.4.

Gramática gráfica concreta

<block substructure diagram> ::=
<frame symbol>
contains {<block substructure heading>
{ {<block substructure text area>}*
{<macro diagram>}*
<block interaction area>
{<type in system area> }*}***set*** }
is associated with {<channel identifiers>}*

Los <channel identifiers> identifican canales conectados a subcanales en el <block substructure diagram>. Se colocan fuera del <frame symbol> próximo al punto extremo de los subcanales en el <frame symbol>.

Un <channel symbol> dentro del <frame symbol> y conectado a éste indica un subcanal.

<block substructure heading> ::=
substructure
{<block substructure name> | <block substructure identifier>}

<block substructure text area> ::=
<system text area>

<block substructure area> ::=
<graphical block substructure reference>
| <block substructure diagram>
| <open block substructure diagram>

<graphical block substructure reference> ::=
<block substructure symbol> ***contains*** <block substructure name>

<block substructure symbol> ::=
<block symbol>

<open block substructure diagram> ::=
{ {<block substructure text area>}*
{<macro diagram>}*
<block interaction area>} ***set***

Cuando un <block substructure area> es un <open block substructure diagram>, el <block diagram> o <block type diagram> envolvente no puede contener otras definiciones que parámetros de contexto formales, puertas y la definición de subestructura.

Semántica

Véase 3.2.1.

Modelo

Un <open block substructure diagram> se transforma en un <block substructure diagram> de tal manera que en el <block substructure heading> el <block substructure name> es el mismo que el <name> del <block diagram> o el <block type diagram> envolvente.

Ejemplo

En la Figura 3.3.1 se da un ejemplo de <open block substructure diagram>.

A continuación se presenta un ejemplo de <block substructure definition>.

```
substructure A ;
   signal s5(nat), s6, s8, s9(min);
   block a1 referenced;
   block a2  referenced;
   block a3  referenced;
   channel c1   from a2 to env with s1, s2; endchannel c1;
   channel c2   from env to a1 with s3;
                from a1 to env with s1; endchannel c2;
   channel d1   from a2 to env with s7; endchannel d1;
   channel d2   from a3 to env with s10; endchannel d2;
   channel e1   from a1 to a2 with s5, s6; endchannel e1;
   channel e2   from a3 to a1 with s8; endchannel e2;
   channel e3   from a2 to a3 with s9; endchannel e3;
   connect c and c1, c2 ;
   connect d and d1, d2 ;
endsubstructure  A;
```

El <block substructure diagram> para el mismo ejemplo se presenta en la Figura 3.2.3.

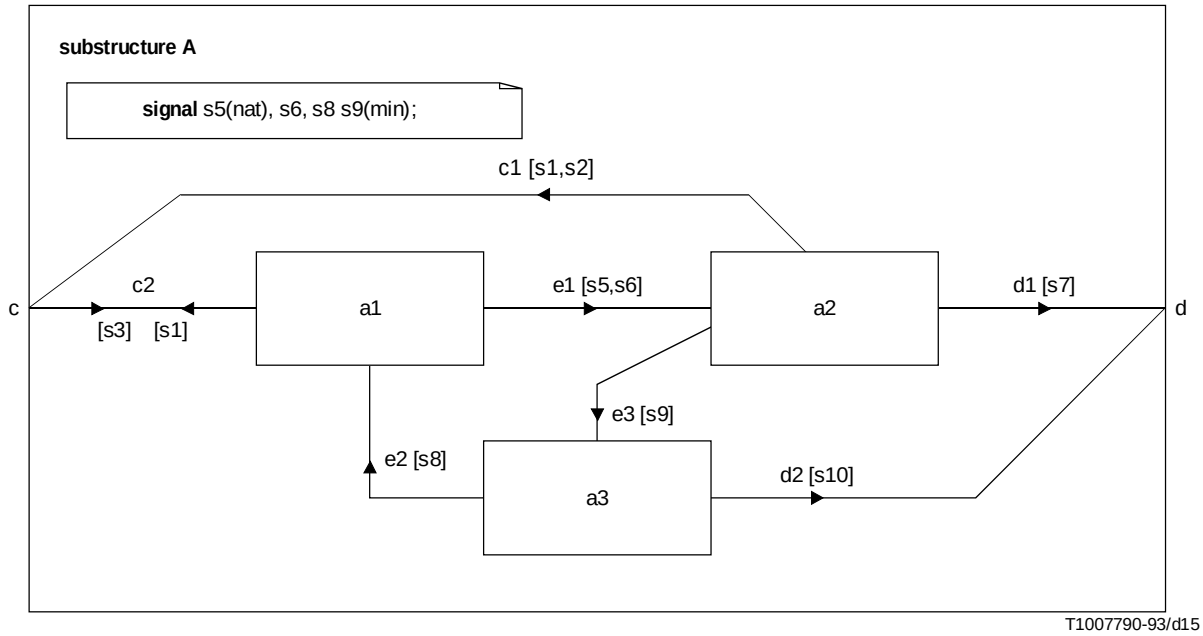


FIGURA 3.2.3/Z.100

Diagrama de subestructura de bloque para un bloque A

3.2.3 Partición de canal

Todas las condiciones estáticas se enuncian mediante una gramática textual concreta. Para la gramática gráfica concreta se cumplen condiciones análogas.

Gramática textual concreta

<channel substructure definition> ::=
substructure {[<channel substructure name>]
| <channel substructure identifier> } <end>
{ <entity in system>
| <channel endpoint connection> }+
endsubstructure [{ <channel substructure name>
| <channel substructure identifier>}] <end>

Si <channel substructure name> que sigue a la palabra clave **substructure** se omite, es el mismo que el <channel name> en la <channel definition> envolvente.

<textual channel substructure reference> ::=
substructure <channel substructure name> **referenced** <end>

<channel endpoint connection> ::=
connect {<block identifier> | **env**}
and <subchannel identifiers> <end>

Los dos puntos extremos de la <channel definition> tienen que ser diferentes, y para cada uno debe haber exactamente una <channel endpoint connection>. El <block identifier> o **env** en una <channel endpoint connection> tiene que identificar uno de los puntos extremos de la <channel definition> fraccionada.

Las condiciones estáticas adicionales de una <channel substructure definition> se definen en función de la <block definition> que resulta de la transformación descrita en *Modelo*.

Gramática gráfica concreta

<channel substructure diagram> ::=
<frame symbol>
contains {<channel substructure heading>
{ {<channel substructure text area>}*
{<macro diagram>}*
{<type in system area>}*
<block interaction area> } ***set*** }
is associated with {<block identifier> | **env**}+

El <block identifier> o **env** identifica un punto extremo del canal fraccionado. El <block identifier> se coloca fuera del <frame symbol> próximo al punto extremo del subcanal asociado en el <frame symbol>. El <channel symbol> dentro del <frame symbol> y conectado a éste indica un subcanal.

<channel substructure heading> ::=
substructure
{ <channel substructure name> | <channel substructure identifier>}

<channel substructure text area> ::=
<system text area>

<channel substructure association area> ::=
<dashed association symbol>
is connected to <channel substructure area>

<channel substructure area> ::=
<graphical channel substructure reference>
| <channel substructure diagram>

<graphical channel substructure reference> ::=
<channel substructure symbol> ***contains*** <channel substructure name>

<channel substructure symbol> ::=

<block symbol>

Modelo

Una <channel definition> que contiene una <channel substructure definition> se transforma en una <block definition> y dos <channel definition>s de tal modo que:

a) Cada una de las dos <channel definition>s está conectada al bloque y a un punto extremo del canal original. Las <channel definition>s tienen nombres nuevos distintos y toda referencia al canal original en los constructivos **via** se reemplaza por una referencia al nuevo canal apropiado. Los dos canales implícitos causan retardo únicamente si el canal subestructurado causa retardo.

b) La <block definition> tiene un nombre nuevo distinto y contiene solamente una <block substructure definition> que tiene el mismo nombre y contiene las mismas definiciones que la <channel substructure definition> original. Los calificadores en la nueva <block definition> se cambian para incluir el nombre de bloque. Las dos <channel endpoint connection>s provenientes de la <channel substructure definition> original se representan por dos <channel connection>s en las cuales el <block identifier> o **env** se reemplaza por el nuevo canal apropiado.

c) <output>s dentro de la subestructura de canal que menciona el canal en un <via path> su cláusula via que menciona <channel identifier> es sustituida por una cláusula via que contiene uno o dos de los canales implícitos en su <via path>, de modo que el canal que tiene el <signal identifier> en su <signal list> para un sentido procedente de <block definition> está en <via path>. Si <channel identifier> es sustituido por dos canales implícitos en <via path>, el cambio se produce en orden arbitrario.

Ejemplo

A continuación se presenta un ejemplo de <channel substructure definition>.

```
channel C from A to B  with s1;
       from B to A with s2;
       substructure C;
        signal s3(hel), s4(boo), s5;
        block b1 referenced;
        block b2 referenced;
        channel c1 from env to b1 with s1;
                    from b1 to env with s2; endchannel c1;
        channel c2   from b2 to env with s1;
                    from env to b2 with s2; endchannel c2;
        channel e1   from b1 to b2 with s3; endchannel e1;
        channel e2   from b2 to b1 with s4, s5; endchannel e2;
        connect A and c1;
        connect B and c2;
       endsubstructure C;
 endchannel C;
```

El <channel substructure diagram> para el mismo ejemplo se presenta en la Figura 3.2.4.

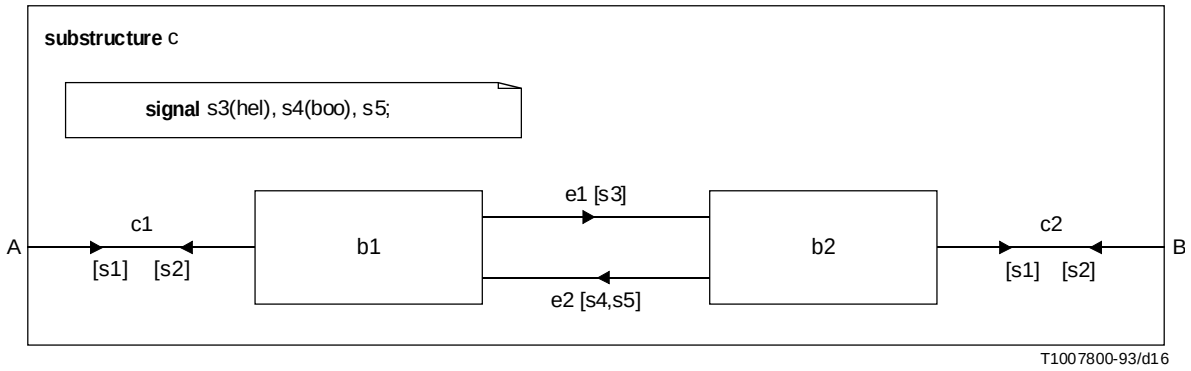


FIGURA 3.2.4/Z.100

Diagrama de subestructura de canal para un canal C

3.3 Refinamiento

El refinamiento se efectúa refinando una definición de señal para formar un conjunto de definiciones de subseñal. Una definición de subseñal es una definición de señal y, a su vez, puede ser refinada. Este refinamiento puede repetirse cualquier número de veces, y como resultado de ello se obtiene una estructura jerárquica de definiciones de señal y sus definiciones (directas o indirectas) de subseñal. Una definición de subseñal de una definición de señal no se considera un componente de la definición de señal.

Gramática abstracta

Signal-refinement :: *Subsignal-definition*-**set**

Subsignal-definition :: [**REVERSE**] *Signal-definition*

Para cada *Channel-connection* se debe cumplir que para cada *Signal-identifier* asociado a cada *Channel-identifier*, bien el *Signal-identifier* está asociado por lo menos a uno de los *Sub-channel-identifiers*, bien cada uno de sus identificadores de subseñal está asociado a por lo menos uno de los *Sub-channel-identifiers*. Este es un cambio de las reglas de partición correspondientes.

En el conjunto completo de señales de entrada válidas de un conjunto de instancias de proceso denotado por una definición de proceso o de los *Output-node*s de una definición de proceso no puede haber dos señales en dos niveles diferentes de refinamiento de la misma señal.

Gramática textual concreta

<signal refinement> ::=
refinement
{<subsignal definition>}+
endrefinement

<subsignal definition> ::=
<reverse> <signal definition>

<reverse> ::=
[**reverse**]

Una <signal definition> en una <subsignal definition> no puede tener parámetros de contexto formales.

Semántica

Cuando se define que una señal sea transportada por un canal, el canal será automáticamente el portador de todas las subseñales de esa señal. Puede haber un refinamiento cuando el canal es dividido o conectado a subcanales en una subestructura. En tal caso, los subcanales transportarán las subseñales en lugar de la señal refinada. El sentido de flujo de una subseñal está determinado por el subcanal portador. Una subseñal puede tener un sentido opuesto al de la señal refinada, lo que se indica por la palabra clave **reverse**. Las señales sólo pueden refinarse en <channel connection>s y <channel endpoint connection>s.

Cuando una definición de sistema contiene un refinamiento de señal, el concepto de subconjunto de partición consistente queda restringido. Se dice que tal definición de sistema contiene varios subconjuntos de refinamiento consistentes.

Un subconjunto de refinamiento consistente es un subconjunto de partición consistente (como se indica en 3.2.1) restringido por la regla adicional siguiente:

d) Al seleccionar el subconjunto de partición coherente, las señales deben utilizarse en el mismo nivel de refinamiento en los procesos de comunicación. Es decir, para cada conjunto de instancias de proceso por el cual los trayectos de comunicación pueden transportar una señal a otro conjunto de instancias de proceso, la señal se debe utilizar en el mismo nivel de refinamiento en ambos conjuntos.

Ejemplo

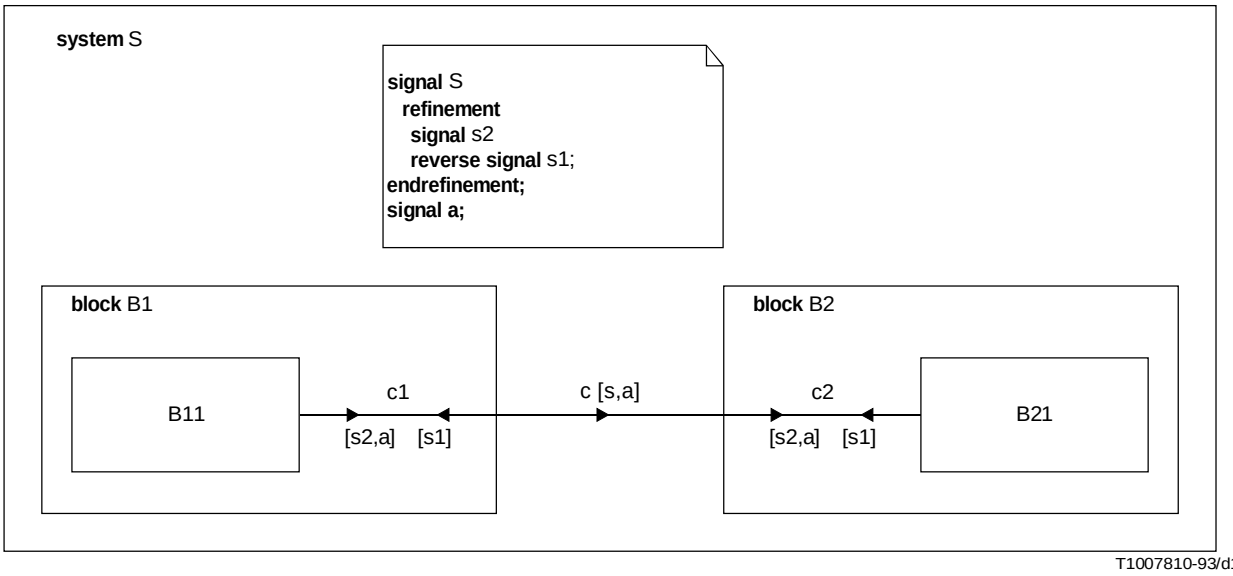


FIGURA 3.3.1/Z.100

Diagrama de sistema que contiene un refinamiento de señal

En el ejemplo de la Figura 3.3.1, la señal s es refinada en las definiciones de bloque B1 y B2, pero la señal a no es refinada. En el nivel de refinamiento más elevado, los procesos B1 y B2 comunican utilizando señales s y a. En el nivel inferior inmediato, los procesos B11 y B21 comunican utilizando las señales s1, s2 y a.

No está autorizado el refinamiento en una sola de las definiciones de bloque B1 y B2, pues no hay una transformación dinámica entre una señal y sus subseñales, sino solamente una relación estática.

4 Conceptos adicionales del SDL básico

4.1 Introducción

Esta cláusula define cierto número de conceptos adicionales. Se presentan por razones de conveniencia para los usuarios de SDL, además de las notaciones taquigráficas definidas en otras cláusulas de la Recomendación.

Las propiedades de las notaciones taquigráficas se derivan de la forma en que se modelan en términos de (o se transforman en) los conceptos primitivos. A fin de asegurar una utilización cómoda e inequívoca de las notaciones taquigráficas, y de reducir los efectos secundarios cuando se combinan varias notaciones taquigráficas, estos conceptos se transforman en un orden especificado definido en 7. El orden de transformación también se sigue cuando se definen los conceptos en esta cláusula.

4.2 Macro

En el texto que sigue, los términos definición de macro y llamada a macro se utilizan en un sentido general, que abarca tanto el SDL/GR como el SDL/PR. Una definición de macro contiene una colección de símbolos gráficos y/o unidades léxicas, que pueden incluirse en uno o más lugares en la <sdl specification>. Cada uno de estos lugares se indica por una llamada a macro. Antes de que pueda analizarse una <sdl specification>, cada llamada a macro deberá reemplazarse por la correspondiente definición de macro.

4.2.1 Reglas léxicas

<formal name> ::=
[<name>%] <macro parameter>
{[%<name>]%<macro parameter>}*
[%<name>]

4.2.2 Definición de macro

Gramática textual concreta

<macro definition> ::=
macrodefinition <macro name>
[<macro formal parameters>] <end>
<macro body>
endmacro [<macro name>] <end>

<macro formal parameters> ::=
fpar <macro formal parameter> {, <macro formal parameter>}*

<macro formal parameter> ::=
<name>

<macro body> ::=
{<lexical unit>|<formal name>}*

<macro parameter> ::=
<macro formal parameter>
| **macroid**

Los <macro formal parameter>s tienen que ser distintos. Los <macro actual parameter>s de una llamada a macro tienen que tener una concordancia de uno a uno con sus correspondientes <macro formal parameter>s.

El <macro body> no debe contener las palabras clave **endmacro** y **macrodefinition**.

Gramática gráfica concreta

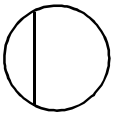
<macro diagram> ::=
<frame symbol> ***contains*** {<macro heading> <macro body area>}

<macro heading> ::=
macrodefinition <macro name> [<macro formal parameters>]

<macro body area> ::=
{ {<any area> }*
<any area> [***is connected to*** <macro body port1>] }***set***
| { <any area> ***is connected to*** <macro body port2>
<any area> ***is connected to*** <macro body port2>
{ <any area> [***is connected to*** <macro body port2>]}*}***set***

<macro inlet symbol>::=

<macro outlet symbol> ::=



<macro body port1> ::=
<outlet symbol> ***is connected to*** {<frame symbol>
[***is associated with*** <macro label>]
| { <macro inlet symbol> | <macro outlet symbol> }
[{ ***contains*** <macro label>
| ***is associated with*** <macro label> }] }

<macro body port2> ::=
<outlet symbol> ***is connected to*** {<frame symbol>
is associated with <macro label>
| { <macro inlet symbol> | <macro outlet symbol> }
{ ***contains*** <macro label>
| ***is associated with*** <macro label> }}

<macro label> ::=
<name>

<outlet symbol> ::=
<dummy outlet symbol>
| <flow line symbol>
| <channel symbol>
| <signal route symbol>
| <solid association symbol>
| <dashed association symbol>
| <create line symbol>

<dummy outlet symbol> ::=
<solid association symbol>

<any area> ::=
<block area>
| <block interaction area>
| <block substructure area>
| <block substructure text area>
| <block text area>
| <block type reference>
| <channel definition area>
| <channel substructure area>

| <channel substructure association area>
| <channel substructure text area>
| <comment area>
| <continuous signal area>
| <continuous signal association area>
| <create line area>
| <create request area>
| <decision area>
| <enabling condition area>
| <existing typebased block definition>
| <export area>
| <graphical block reference>
| <graphical typebased block definition>
| <graphical procedure reference>
| <graphical process reference>
| <in-connector area>
| <input area>
| <input association area>
| <macro call area>
| <merge area>
| <nextstate area>
| <operator heading>
| <operator text area>
| <option area>
| <out-connector area>
| <output area>
| <package reference area>
| <package text area>
| <priority input area>
| <priority input association area>
| <procedure area>
| <procedure call area>
| <procedure graph area>
| <procedure start area>
| <procedure text area>
| <process area>
| <process graph area>
| <process interaction area>
| <process text area>
| <process type graph area>
| <process type reference>
| <remote procedure call area>
| <remote procedure input area>
| <remote procedure save area>
| <reset area>
| <return area>
| <save area>
| <save association area>
| <service area>
| <service interaction area>
| <service graph area>
| <service text area>
| <service type reference>
| <set area>
| <signal list area>
| <signal route definition area>
| <spontaneous transition association area>
| <spontaneous transition area>
| <start area>
| <state area>
| <stop symbol>
| <system text area>
| <task area>

| <text extension area>
| <transition area>
| <transition option area>
| <transition string area>
| <type in system area>
| <type in block area>
| <type in process area>

Un <dummy outlet symbol> lo único que puede tener asociado es una <macro label>.

Para un <outlet symbol> que no sea un <dummy outlet symbol>, el correspondiente <inlet symbol> en la llamada a macro tiene que ser un <dummy inlet symbol>.

Un <macro body> puede aparecer en cualquier texto a que se hace referencia en <any area>.

Semántica

Una <macro definition> contiene unidades léxicas, mientras que un <macro diagram> contiene unidades sintácticas. Así, la relación de correspondencia entre construcciones de macro en sintaxis textual y sintaxis gráfica generalmente no es posible. Por la misma razón se aplican reglas detalladas separadas para sintaxis textual y para sintaxis gráfica, aunque hay algunas reglas comunes.

El <macro name> es visible en la totalidad de la definición de sistema, cualquiera que sea la definición de macro que aparezca. Una llamada a macro puede aparecer antes de la definición de macro correspondiente.

Una definición de macro puede contener llamadas a macro, pero no podrá llamarse a sí misma directamente, ni tampoco indirectamente a través de llamadas a macro en otras definiciones de macro.

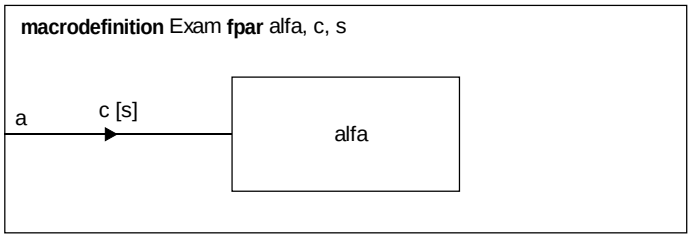
La palabra clave **macroid** puede utilizarse como un pseudo parámetro formal de macro dentro de cada definición de macro. No se le podrá dar ningún <macro actual parameter>s y se remplaza por un <name> único para cada expansión de una definición de macro (dentro de una expansión se utiliza el mismo <name> para cada ocurrencia de **macroid**).

Ejemplo

A continuación se presenta un ejemplo de una <macro definition>:

```
macrodefinition Exam
fpar alfa, c, s, a;
     block alfa referenced;
     channel c  from a to alfa with s; endchannel c;
endmacro Exam;
```

El <macro diagram> para el mismo ejemplo se presenta a continuación. Sin embargo, el <macro formal parameter>, a, no se requiere en este caso



T1007820-93/d18

4.2.3 Llamada a marco

Gramática textual concreta

<macro call> ::=
macro <macro name> [<macro call body>] <end>

<macro call body> ::=
(<macro actual parameter> {, <macro actual parameter>}*)

<macro actual parameter> ::=

{<lexical unit>}*

La <lexical unit> no puede ser una coma «,» ni un paréntesis derecho «)». Si uno cualquiera de estos caracteres debe ser utilizado en un <macro actual parameter>, el <macro actual parameter> tiene que ser una <character string>. Si el <macro actual parameter> es una <character string>, el valor de la <character string> se utiliza cuando el <macro actual parameter> reemplaza un <macro formal parameter>.

Una <macro call> puede aparecer en cualquier lugar en que esté autorizada una <lexical unit>.

Gramática gráfica concreta

<macro call area> ::=
<macro call symbol> ***contains*** {<macro name> [<macro call body>]}
[***is connected to***
{<macro call port1> | <macro call port2> {<macro call port2>}+}]

<macro call symbol> ::=

<macro call port1> ::=
<inlet symbol> [***is associated with*** <macro label>]
is connected to <any area>

<macro call port2> ::=
<inlet symbol> ***is associated with*** <macro label>
is connected to <any area>

<inlet symbol> ::=
<dummy inlet symbol>
| <flow line symbol>
| <channel symbol>
| <signal route symbol>
| <solid association symbol>
| <dashed association symbol>
| <create line symbol>

<dummy inlet symbol> ::=

<solid association symbol>

Un <dummy inlet symbol> lo único que puede tener asociado es <macro label>. Para cada <inlet symbol> tiene que haber un <outlet symbol> en el correspondiente <macro diagram>, asociado con la misma <macro label>. Para un <inlet symbol> que no sea un <dummy inlet symbol>, el <outlet symbol> correspondiente tiene que ser un <dummy outlet symbol>.

Salvo en el caso de <dummy inlet symbol>s y <dummy outlet symbol>s, es posible tener múltiples <lexical unit>s (textuales) asociadas con un <inlet symbol> o <outlet symbol>. En este caso la <lexical unit> más próxima al <macro

call symbol> o el <frame symbol> del <macro diagram> se toma por la <macro label> asociada con el <inlet symbol> o <outlet symbol>.

El <macro call area> puede aparecer en cualquier lugar en que esté autorizada un área. Sin embargo, se requiere cierto espacio entre el <macro call symbol> y cualquier otro símbolo gráfico cerrado. Si de acuerdo con las reglas sintácticas tal espacio no puede estar vacío, el <macro call symbol> se conecta al símbolo gráfico cerrado con un <dummy inlet symbol>.

Semántica

Una definición de sistema puede contener definiciones de macro y llamadas a macro. Antes de que puedan analizarse estas definiciones de sistema, todas las llamadas a macro deben ser expandidas. La expansión de una llamada a macro significa que una copia de la definición de macro que tiene el mismo <macro name> que el dado en la llamada a macro reemplaza la llamada a macro.

Cuando se llama a una definición de macro, ésta es expandida. Esto significa que se crea una copia de la definición de macro y cada ocurrencia de los <macro formal parameter>s de la copia es reemplazada por los correspondientes <macro actual parameter>s de la llamada a macro, después de lo cual las llamadas a macro en la copia, si existen, son expandidas. Cuando se reemplazan los <macro formal parameter>s por los <macro actual parameter>s, se suprimen todos los caracteres «%» en los <formal name>s.

Debe haber una correspondencia de uno a uno entre <macro formal parameter> y <macro actual parameter>.

Modelo

Para reemplazar el <macro call area> por una copia del <macro diagram> se procede de la siguiente manera. Todos los <macro inlet symbol>s y <macro outlet symbol>s se suprimen. Un <dummy outlet symbol> se reemplaza por el <inlet symbol> que tiene la misma <macro label>. Un <dummy inlet symbol> se reemplaza por el <outlet symbol> que tiene la misma <macro label>. Después se suprimen las <macro label>s asociadas a <inlet symbol>s y <outlet symbol>s. También se suprimen los <macro body port1> y <macro body port2> que no tienen <macro call port1> o <macro call port2> correspondientes.

Ejemplo

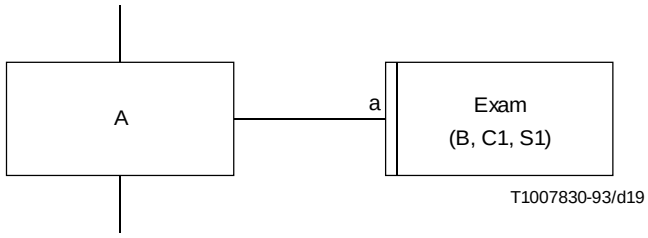
A continuación se presenta un ejemplo de <macro call>, dentro de un fragmento de una <block definition>.

```
.........
block A referenced;
macro Exam (B, C1, S1, A);
.........
```

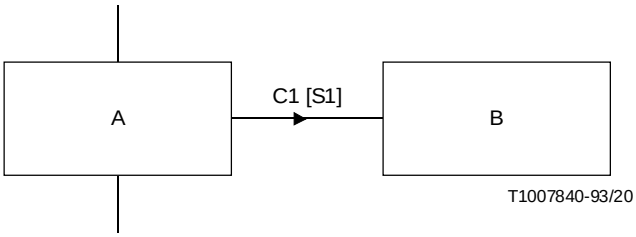
La expansión de esta llamada a macro, utilizando el ejemplo 4.2.2, da el siguiente resultado.

```
.........
block A referenced;
block B referenced;
channel C1 from A to B with S1; endchannel C1;
.........
```

Seguidamente se presenta la <macro call area> para el mismo ejemplo, dentro de un fragmento de una <block interaction area>.



La expansión de esta llamada a macro da el siguiente resultado.



4.3 Definición de sistema genérica

Una especificación de sistema debe tener partes facultativas y parámetros de sistema con valores no especificados para satisfacer diversas necesidades. Tal especificación de sistema se denomina genérica. Su propiedad genérica se especifica por medio de sinónimos externos (que son análogos a parámetros formales de una definición de procedimiento). Una especificación de sistema genérica se concretiza seleccionando un subconjunto adecuado de la misma y proporcionando un valor para cada uno de los parámetros de sistema. La especificación de sistema resultante no contiene sinónimos externos, y se denomina especificación de sistema específica.

4.3.1 Sinónimo externo

Gramática textual concreta

<external synonym definition>::=

synonym <external synonym definition item>

{, <external synonym definition item> }*

<external synonym definition item> ::=

<external synonym name> <predefined sort> = **external**

<external synonym> ::=

<external synonym identifier>

Una <external synonym definition> puede aparecer en cualquier lugar en que se permita una <synonym definition> (véase 5.3.1.13). Un <external synonym> puede utilizarse en cualquier lugar en que se permita un <synonym> (véase 5.3.3.3). Los géneros predefinidos son: Boolean (booleano), Character (carácter), Charstring (cadena-de-caracteres), Integer (entero), Natural, Real, PId, Duration (duración) y Time (tiempo).

Semántica

Un <external synonym> es un <synonym> cuyo valor no está especificado en la definición de sistema. Esto se indica por la palabra clave **external** que se utiliza en lugar de una <simple expression>.

Una definición de sistema genérico es una definición de sistema que contiene <external synonym>s o <informal text> en una opción de transición (véase 4.3.4). Una definición de sistema específico se crea a partir de una definición de sistema genérico proporcionando valores para los <external synonym>s, y transformando <informal text> en construcciones formales. La manera de efectuar esto, y la relación con la gramática abstracta, no forma parte de la definición del lenguaje.

4.3.2 Expresión simple

Gramática textual concreta

<simple expression> ::=

<ground expression>

La *Ground-expresion* representada por una <simple expression> contendrá solamente operadores, sinónimos y literales definidos en el paquete Predefined, como se indica en el Anexo D.

Semántica

Una expresión simple es una *Ground-expression*.

4.3.3 Definición facultativa

Gramática textual concreta

```
<select definition> ::=
                    select if (<Boolean simple expression> ) <end>
                    {       <system type definition>
                        |   <textual system type reference>
                        |   <block type definition>
                        |   <textual block type reference>
                        |   <block definition>
                        |   <textual block reference>
                        |   <textual typebased block definition>
                        |   <channel definition>
                        |   <signal definition>
                        |   <signal list definition>
                        |   <remote variable definition>
                        |   <remote procedure definition>
                        |   <data definition>
                        |   <process type definition>
                        |   <textual process type reference>
                        |   <process definition>
                        |   <textual process reference>
                        |   <textual typebased process definition>
                        |   <service type definition>
                        |   <textual service type reference>
                        |   <service definition>
                        |   <textual service reference>
                        |   <textual typebased service definition>
                        |   <timer definition>
                        |   <channel connection>
                        |   <channel endpoint connection>
                        |   <variable definition>
                        |   <view definition>
                        |   <imported variable specification>
                        |   <procedure definition>
                        |   <textual procedure reference>
                        |   <imported procedure specification>
                        |   <signal route definition>
                        |   <channel to route connection>
                        |   <signal route to route connection>
                        |   <select definition>
                        |   <macro definition> }+
                    endselect <end>
```

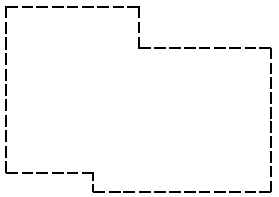
Los únicos nombres visibles en una <Boolean simple expression> de <select definition> son nombres de sinónimos externos definidos fuera de cualquier <select definition>s u <option area>s y literales y operadores de los géneros definidos dentro del paquete Predefined definido en el Anexo D.

Una <select definition> puede contener solamente las definiciones que estén sintácticamente permitidas en ese lugar.

Gramática gráfica concreta

<option area> ::=
<option symbol> ***contains***
{ **select if** (<Boolean simple expression>)
{ <system type diagram>
| <system type reference>
| <block type diagram>
| <block type reference>
| <block area>
| <channel definition area>
| <system text area>
| <block text area>
| <process text area>
| <procedure text area>
| <block substructure text area>
| <channel substructure text area>
| <service text area>
| <macro diagram>
| <process type diagram>
| <process type reference>
| <process area>
| <service type diagram>
| <service type reference>
| <service area>
| <procedure area>
| <signal route definition area>
| <create line area>
| <option area> } + }

El <option symbol> es un polígono de trazo discontinuo con esquinas de trazo continuo, por ejemplo:



Un <option symbol> contiene lógicamente la totalidad de cualquier símbolo gráfico unidimensional cortado por su frontera (es decir, con un punto extremo en su exterior).

Una <option area> puede aparecer en cualquier lugar, salvo dentro de <process graph area> y <process type graph area>. Una <option area> puede contener solamente las áreas y diagramas que están sintácticamente permitidos en ese lugar.

Semántica

Si el valor de la <Boolean simple expression> es Falso, los constructivos contenidos en la <select definition> o en el <option symbol> no se seleccionan. En el otro caso se seleccionan los constructivos.

Modelo

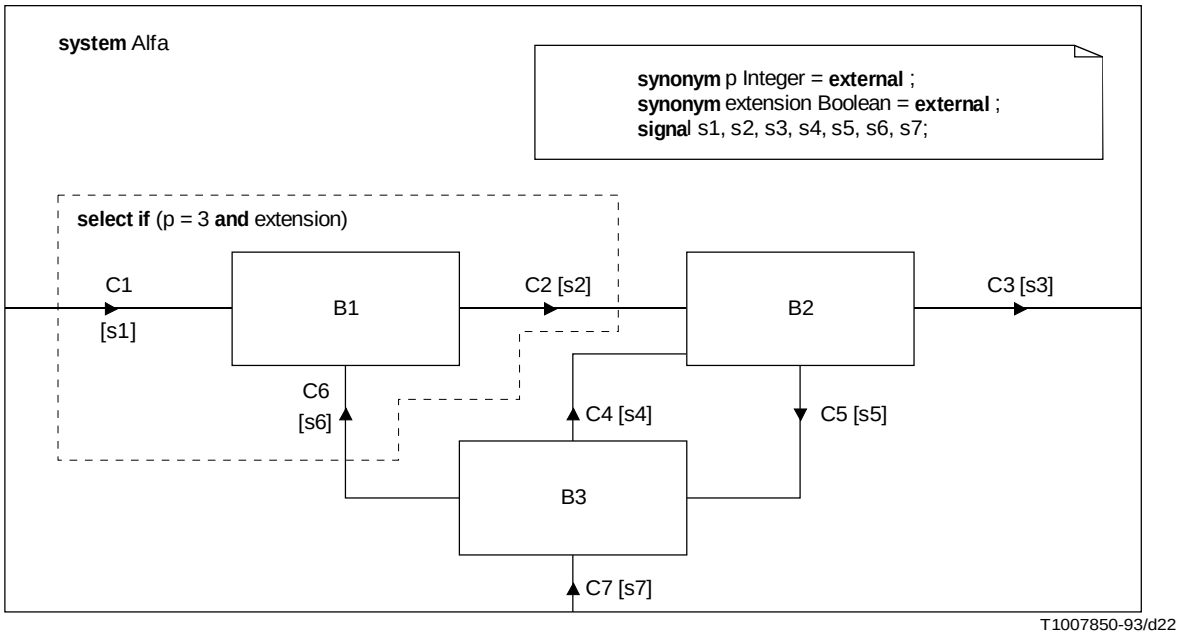
En una transformación la <select definition> y la <option area> se reemplazan por los constructivos seleccionados contenidos, si existen. También se suprimen todos los conectores conectados a un área dentro de <option area>s no seleccionadas.

Ejemplo

En el sistema Alfa hay tres bloques: B1, B2, y B3. El bloque B1 y los canales conectados a él son opcionales, y dependen de los valores de los sinónimos externos p y ampliación. En SDL/PR, este ejemplo se representa como sigue.

```
system Alfa;
     synonym p Integer = external;
     synonym extension Boolean = external;
     signal s1,s2,s3,s4,s5,s6,s7;
     select  if (p = 3 and extension);
        block B1 referenced;
        channel C1 from env to B1 with s1 ; endchannel C1;
        channel C2 from B1 to B2 with s2 ; endchannel C2;
        channel C6 from B3 to B1 with s6; endchannel C6;
     endselect;
     channel C3 from B2 to env with s3 ; endchannel C3;
     channel C4 from B3 to B2 with s4 ; endchannel C4;
     channel C5 from B2 to B3 with s5; endchannel C5;
     channel C7 from env to B3 with s7 ; endchannel C7;
     block B2 referenced;
     block B3 referenced;
endsystem Alfa;
```

El mismo ejemplo en sintaxis SDL/GR se representa como sigue.



4.3.4 Cadena de transición facultativa

Gramática textual concreta

<transition option> ::=
alternative <alternative question> <end>
{ <answer part> <else part>
| <answer part> { <answer part> }+ [<else part>] }
endalternative

<alternative question>::=
<simple expression>
| <informal text>

Toda <ground expression> en <answer> tiene que ser una <simple expression>. Las <answer>s en una <transition option> tienen que ser mutuamente exclusivas. Si la <alternative question> es una <expression>, la *Range-condition* de las <answer>s tienen que ser del mismo género que el de la <alternative question>.

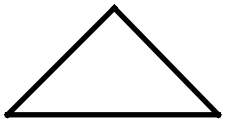
No puede omitirse ninguna <answer> en <answer part>s de <transition option>.

Gramática gráfica concreta

<transition option area> ::=

<transition option symbol> ***contains*** {<alternative question>}
is followed by {<option outlet1>
{<option outlet1> | <option outlet2> }
{ <option outlet1> }* } ***set***

<transition option symbol> ::=



<option outlet1> ::=

<flow line symbol> ***is associated with*** <graphical answer>
is followed by <transition area>

<option outlet2> ::=

<flow line symbol> ***is associated with*** **else**
is followed by <transition area>

El <flow line symbol> en <option outlet1> y <option outlet2> está conectado a la parte inferior del <transition option symbol>. Los <flow line symbol>s que salen de un <transition option symbol> pueden tener un trayecto de origen común. La <graphical answer> y **else** pueden colocarse a lo largo del <flow line symbol> asociado, o en el <flow line symbol> interrumpido.

Las <graphical answer>s en un <transition option area> tienen que ser mutuamente exclusivas.

Semántica

Se seleccionan constructivos en una <option outlet1> si la <answer> contiene el valor de la <alternative question>. Si ninguna de las <answer>s contiene el valor de la <alternative question>, se seleccionan los constructivos en la <option outlet2>.

Si no existe una <option outlet2> y no se selecciona ninguno de los trayectos salientes, la selección es inválida.

Modelo

Si una <transition option> no es terminadora, es una sintaxis derivada para una <transition option> en la cual todas las <answer part>s y la <else part> han insertado en su <transition>:

a) si la opción de transición es el último <action statement> en una <transition string>, un <join> hacia el <terminator statement> siguiente; o

b) un <join> hacia el primer <action statement> que sigue a la opción de transición.

<transition option> de terminación se define en 2.7.5.

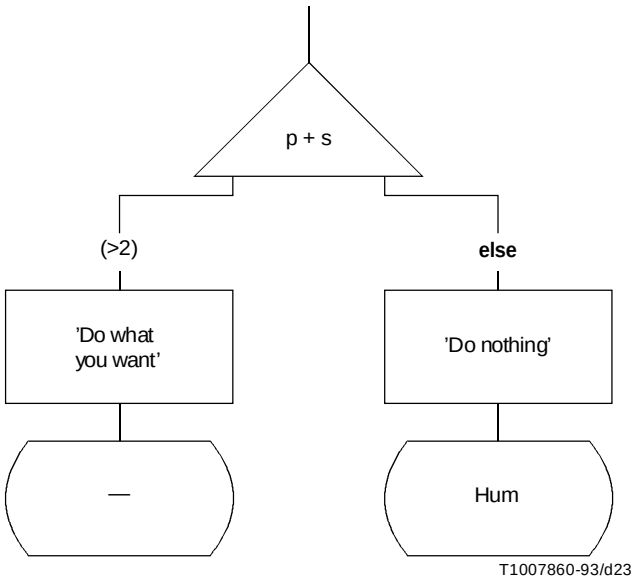
En una transformación, la <transition option> y el <transition option area> se suprimen y se reemplazan por los constructivos seleccionados contenidos.

Ejemplo

A continuación se presenta un fragmento de una <process definition> que contiene una <transition option>. *p* y *s* son sinónimos.

```
.............
alternative p + s;
     (>2) : task 'Do what you want';
        nextstate -;
     else: task 'Do nothing';
        nextstate Hum;
endalternative;
.............
```

Seguidamente se presenta el mismo ejemplo en sintaxis gráfica concreta.



4.4 Estado asterisco

Gramática textual concreta

<asterisk state list> ::=

<asterisk> [(<state name> {, <state name>}*)]

<asterisk> ::=

*

Los <state name>s en una <asterisk state list> deben ser distintos y deben estar contenidos en otras <state list>s en el <body> circundante o en el <body> de un supertipo.

Modelo

Un <state> con una <asterisk state list> se transforma en una lista de <state>s, uno para cada <state name> del <body> en cuestión, salvo en lo que concierne a los <state name>s contenidos en la <asterisk state list>.

4.5 Múltiple aparición de estado

Gramática textual concreta

Un <state name> puede aparecer en más de un <state> de un <body>.

Modelo

Cuando varios <state>s contienen el mismo <state name>, estos <state>s se concatenan para formar un <state> que tiene ese <state name>.

4.6 Entrada asterisco

Gramática textual concreta

<asterisk input list> ::=

<asterisk>

Un <state> puede contener cuando más una <asterisk input list>. Un <state> no contendrá a la vez <asterisk input list> y <asterisk save list>.

Modelo

Una <asterisk input list> se transforma en una lista de <input part>s a razón de una para cada miembro del conjunto completo de señales de entrada válidas de <process definition> o <service definition> circundante, salvo para <signal identifier>s de señales de entrada implícitas introducidas por los conceptos adicionales de 4.10 a 4.14, y para <signal identifier>s contenidos en las otras <input list>s y <save list>s de <state>.

4.7 Conservación asterisco

Gramática textual concreta

<asterisk save list> ::=

<asterisk>

Un <state> puede contener cuando más una <asterisk save list>. Un <state> no contendrá a la vez <asterisk input list> y <asterisk save list>.

Modelo

Una <asterisk save list> se transforma en una lista de <stimulus>s que contiene el conjunto completo de señales de entrada válidas de la <process definition> o <service definition> envolvente, salvo para <signal identifier>s de señales de entrada implícitas introducidas por los conceptos adicionales de 4.10 a 4.14, y de <signal identifier>s contenidos en las otras <input list>s y <save list>s de <state>.

4.8 Transición implícita

Gramática textual concreta

Un <signal identifier> contenido en el conjunto completo de señales de entrada válidas de una <process definition> o <service definition> puede omitirse en el conjunto de <signal identifier>s contenido en las <input list>s, <priority input list>s y la <save list> de un <state>.

Modelo

Para cada <state> hay una <input part> implícita que contiene una <transition> que sólo contiene un <nextstate> que hace retornar al mismo <state>.

4.9 Estado siguiente indicado por guión

Gramática textual concreta

<dash nextstate> ::=
<hyphen>

<hyphen>::=
-

La <transition> contenida en un <start> no podrá conducir, directa ni indirectamente, a un <dash nextstate>.

Modelo

En cada <nextstate> de un <state> el <dash nextstate> se remplaza por el <state name> del <state>.

4.10 Entrada prioritaria

En algunos casos conviene expresar que la recepción de una señal tiene prioridad sobre la recepción de otras señales. Esta situación puede expresarse mediante una entrada prioritaria.

Gramática textual concreta

<priority input> ::=
priority input [<virtuality>]
<priority input list> <end> <transition>

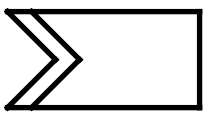
<priority input list> ::=
<stimulus> {, <stimulus>}*

Gramática gráfica concreta

<priority input association area> ::=
<solid association symbol> ***is connected to*** <priority input area>

<priority input area> ::=
<priority input symbol> ***contains***
{ [<virtuality>] <priority input list> }
is followed by <transition area>

<priority input symbol> ::=



Modelo

Una <priority input> o <priority input area> que contiene <virtuality> es denominada una entrada prioritaria virtual, que se describe más detalladamente en 6.3.3.

La entrada prioritaria se transforma de la manera siguiente.

Un estado con el nombre state_name que contiene <priority input>s se divide en dos estados. La transformación requiere dos variables implícitas, n y newn. La variable n se inicializa a 0. Además se requiere una señal implícita X_cont que transporta un valor Entero. Para cada servicio se necesita un solo X_cont si el proceso contiene servicios. Las entradas prioritarias al estado original están conectadas al primer estado, mientras que todas las demás entradas están conectadas al segundo estado (State2) y conservadas en el primer estado. Las transiciones espontáneas, las entradas de procedimiento remoto y las conservaciones de procedimiento remoto del estado original están conectadas a ambos estados. La cadena de transición que conduce al estado original conduce ahora al primer estado (State1). A la cadena de transición se añade la cadena de acción siguiente:

1) todos los <nextstate>s que mencionan el state_name son sustituidos por **join 1**;

2) Se inserta la transición siguiente:

l: **task** n := n+1;
output X_cont(n) **to self**;
nextstate State1;

3) Al primer estado se añade:

input X_cont(newn);
decision (newn = n);
(True): **nextstate** State2;
(False): **nextstate** - ;
enddecision;

4.11 Señal continua

Al describir sistemas puede surgir una situación en que un usuario desearía mostrar que una transición es causada directamente por un valor Verdadero de una expresión booleana. El modelo para conseguir esto consiste en evaluar la expresión durante el estado e iniciar la transición si la expresión da Verdadero. Una notación taquigráfica para esto es la denominada señal continua, que permite iniciar directamente una transición cuando se cumple cierta condición.

Gramática textual concreta

<continuous signal> ::=
provided [<virtuality>]
<Boolean expression> <end>
[**priority** <Integer literal name> <end>] <transition>

Gramática gráfica concreta

<continuous signal association area> ::=
<solid association symbol> ***is connected to*** <continuous signal area>

<continuous signal area> ::=
<enabling condition symbol>
contains{ [<virtuality>] <Boolean expression>
[[<end>]***priority***<Integer literal name>]}
is followed by <transition area>

Semántica

La <Boolean expression> de la <continuous signal> se evalúa al entrar en el estado al cual está asociada, y durante la espera en ese estado, siempre que no haya <stimulus> de una <input list> asociada aplicados en el puerto de entrada. Si el valor de la <Boolean expression> es Verdadero, se inicia la transición. Cuando el valor de la <Boolean expression> es Verdadero en más de una <continuous signal>s, la transición a iniciar es determinada por la <continuous signal> que tiene la prioridad más elevada, es decir, el valor más bajo para <Integer literal name>. Si más <continuous signal>s tienen la misma prioridad, debe hacerse una elección no determinista entre ellas. Si el valor es Falso para todas las

<continuous signal>s con una prioridad explícita, las <Boolean expression>s de <continuous signal>s sin prioridad se consideran en orden arbitrario.

Modelo

Una <continuous signal> o <continuous signal area> que contiene <virtuality> se llama señal continua virtual. La transición continua virtual se describe en 6.3.3.

El estado con el nombre state_name que contiene <continuous signal>s se transforma como se indica a continuación. Esta transformación requiere dos variables implícitas n y newn. La variable n se inicializa a 0. Se requiere además una señal implícita sx.emptyQ que transporta un valor Entero. En este caso, sx denota un servicio, si lo hay, que contiene el estado.

1) Todos los <nextstate>s que mencionan el state_name se reemplazan por **join 1**;

2) Se inserta la siguiente transición:

```
1: task  n:= n+1;
   output  sx.emptyQ(n) to self;
   nextstate  state_name;
```

3) Se añade la siguiente <input part> al <state> state_name:

```
input  sx.emptyQ (newn);
```

y una de decisión <decision> que contiene la <question>

(newn=n)

4a) La <answer part> Falsa contiene

```
nextstate state_name;
```

4b) La <answer part> Verdadera contiene una secuencia de <decision>s que corresponde a las <continuous signal>s en orden de prioridad (una prioridad más elevada se indica por un valor menor del <Integer literal name>). Si más <continuous signal>s tienen la misma prioridad se evalúan todas en orden arbitrario antes del nivel de prioridad siguiente. Cuando todas las <continuous signal>s con prioridad explícita han sido evaluadas, las <continuous signal>s sin prioridad son evaluadas en orden arbitrario.

La <answer part> Falsa contiene la <decision> siguiente, salvo la última <decision> para la cual esta <answer part> contiene: **join 1**;

Cada <answer part> Verdadera de estas <decision>s conduce a la <transition> de la <continuous signal> correspondiente.

El orden arbitrario en el cual se evalúan las <continuous signal>s puede obtenerse utilizando la notación taquigráfica de decisión no determinista, es decir que se efectúa una elección no determinista entre la evaluación de las <continuous signal>s posibles, y si una <continuous signal> es evaluada como Falsa se marca para que no sea evaluada si es sometida a una nueva elección no determinista. En cada caso se registra si quedan por evaluar más <continuous signal>s.

Ejemplo

Véase 4.12.

4.12 Condición habilitante

En SDL, la recepción de una señal o de una transición espontánea en un estado inicia inmediatamente una transición. El concepto de condición habilitante hace posible imponer una condición adicional para el inicio de una transición.

Gramática textual concreta

<enabling condition> ::=

provided <Boolean expression> <end>

Gramática gráfica concreta

<enabling condition area> ::=

<enabling condition symbol> ***contains*** <Boolean expression>

<enabling condition symbol> ::=

Semántica

La <Boolean expression> en la <enabling condition> se evalúa antes de pasar al estado en cuestión, y en cualquier momento en que se reentra en el mismo estado por la llegada de un <stimulus>. En caso de más de una condición habilitante, éstas se evalúan secuencialmente siguiendo un orden arbitrario, antes de acceder al estado. El modelo de transformación garantiza una reevaluación repetida de la expresión enviando <stimulus>s adicionales a través del puerto de entrada. Una señal denotada en la <input list> que precede a la <enabling condition> puede comenzar una transición solamente si el valor de la <Boolean expression> correspondiente es Verdadero. En cambio, si este valor es Falso la señal se conserva. Una <spontaneous transition> precedida por una <enabling condition> sólo puede ser activada si el valor de la <Boolean expression> es Verdadero.

Modelo

El estado con el nombre state_name que contiene <enabling condition>s se transforma como se indica a continuación. Esta transformación requiere dos variables implícitas n y newn. La variable n se inicializa a 0. Se requiere también una señal implícita sx.emptyQ que transporta un valor entero. En este caso, sx denota un servicio, si lo hay, que contiene el estado.

1) Todos los <nextstate>s que mencionan el nombre state_name se reemplazan por **join 1**;

2) Se inserta la siguiente transición:

1: **task** n:= n+1;

output sx.emptyQ (n) **to self**;

Se añaden jerárquicamente, siguiendo un orden arbitrario, cierto número de decisiones, cada una de las cuales contiene sólo una <Boolean expression> correspondiente a alguna <enabling condition> vinculada al estado, de modo que sea posible evaluar todas las combinaciones de valores verdaderos para todas las condiciones habilitantes vinculadas al estado. Cada una de esas combinaciones conduce a un estado nuevo distinto.

3) Cada uno de estos nuevos estados tiene un conjunto de <input part>s que consisten en una copia de las <input part>s del estado sin condiciones habilitantes más las <input part>s para las cuales las <Boolean expression>s de la <enabling condition>'s dieron un valor Verdadero para este estado y un conjunto de <spontaneous transition>s cuyas <Boolean expression>s de la <enabling condition>'s dieron un valor Verdadero para este estado.

Los <stimulus>s y las <remote procedure identifier list>s para las <input part>s restantes constituyen la <save list> y <remote procedure save> para una nueva <save part> asociada a este estado. Las <save part>s y <spontaneous transition>s del estado original se copian también para este nuevo estado.

4) A cada uno de los nuevos estados se añade:

input sx.emptyQ (newn);

Una <decision> que contiene la <question>

(newn=n);

5a) La <answer part> Falsa contiene un <nextstate> que retorna a este mismo nuevo estado.

5b) La <answer part> Verdadera contiene un **join 1**;

6) Si se utilizan <continuous signal>s y <enabling condition>s en el mismo <state>, las evaluaciones de las <Boolean expression>s a partir de las <continuous signal>s se efectúan reemplazando el paso 5b del modelo para <enabling condition> por el paso 4b del modelo para <continuous signal>.

Ejemplo

A continuación se presenta en la Figura 4.12.1 un ejemplo que ilustra la transformación de señal continua y condición habilitante que aparecen en un estado.

Obsérvese en el ejemplo que el conector ec se ha introducido por razones de conveniencia y no forma parte del modelo de transformación.

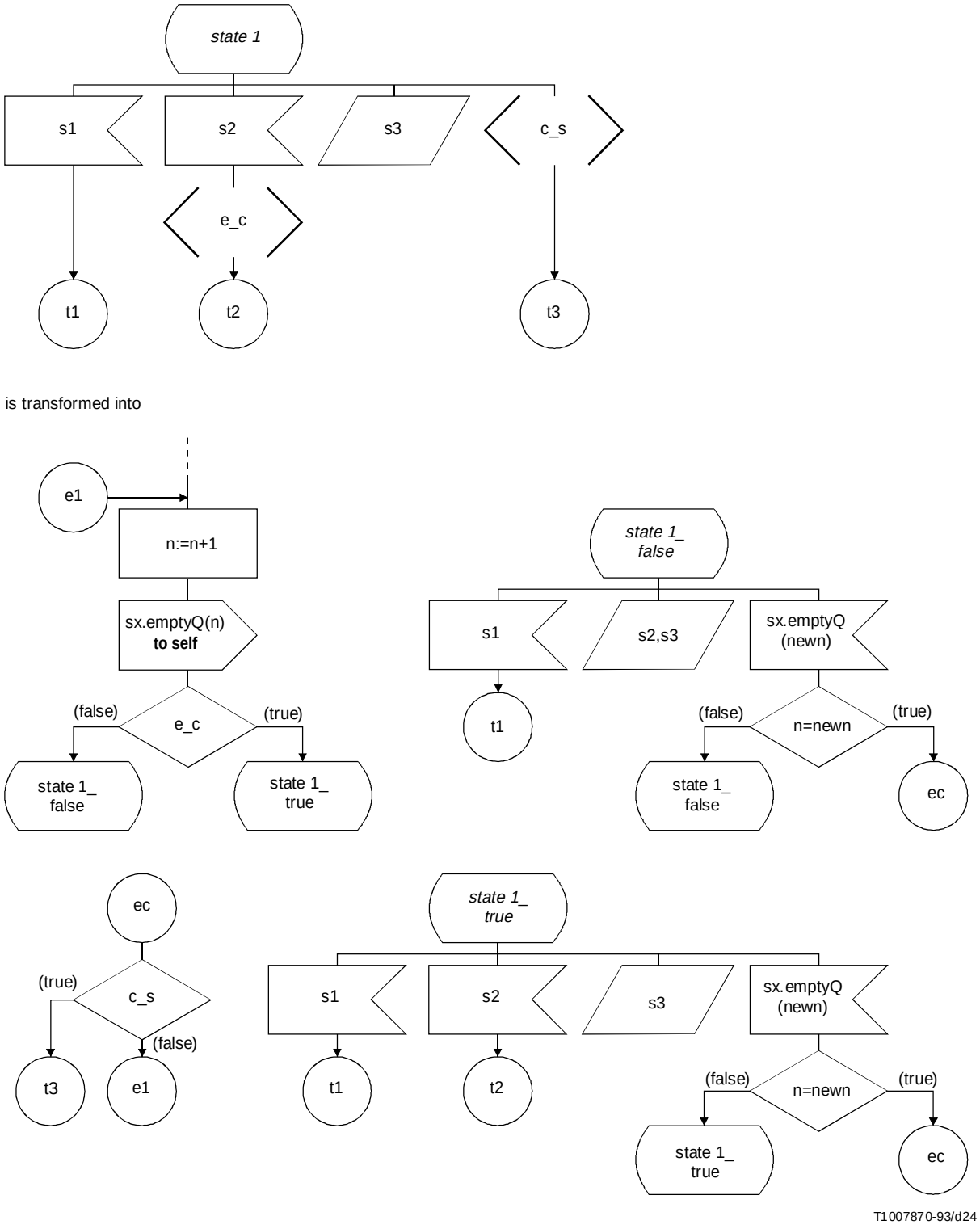


FIGURA 4.12.1/Z.100

Transformación de señal continua y condición habilitante en el mismo estado

4.13 Valor importado y exportado

En SDL una variable siempre pertenece a una instancia de proceso, para la cual es local. Normalmente la variable sólo es visible para la instancia de proceso que la posee, aunque puede ser declarada como un valor revelado (véase 2.6.1.1), lo que permite a otras instancias de proceso en el mismo bloque tener acceso al valor de la variable. Si una instancia de proceso en otro bloque necesita acceder al valor de una variable, es necesario un intercambio de señales con la instancia de proceso que posee la variable.

Esto puede conseguirse mediante la siguiente notación taquigráfica denominada valor importado y exportado. La notación taquigráfica puede utilizarse también para exportar valores a otras instancias de proceso dentro del mismo bloque, en cuyo caso proporciona una alternativa al uso de valores revelados.

Gramática textual concreta

<remote variable definition>::=
remote
<remote variable name> {,<remote variable name>}* <sort> [**nodelay**]
{, <remote variable name> {,<remote variable name>}* <sort> [**nodelay**]}*
<end>

<imported variable specification> ::=
imported
<remote variable identifier> {, <remote variable identifier> }* <sort>
{, <remote variable identifier> {, <remote variable identifier> }* <sort>}* <end>

<import expression> ::=
import (<remote variable identifier> [, <destination>])

<export> ::=
export (<variable identifier> {, <variable identifier> }*)

nodelay denota canales implícitos sin retardo para el intercambio de señales resultante de la transformación descrita en *Modelo*.

El identificador de una <imported variable specification> debe denotar una <remote variable definition> posiblemente implícita (véase a continuación) del mismo género.

El <remote variable identifier> que sigue a **as** en una definición de variable exportada debe denotar una <remote variable definition> del mismo género que la definición de variable exportada. Si no hay cláusula **as** se denota la definición de variable remota en la unidad de ámbito circundante más próxima con el mismo nombre y género que la definición de variable exportada.

Para cada <import expression> debe existir una <imported variable specification> del identificador de variable remota en un tipo de proceso, definición de proceso, tipo de servicio o definición de servicio circundantes.

Para cada variable importada en un proceso debe existir por lo menos un proceso de exportación.

Pueden omitirse definiciones de variable remotas, y en ese caso hay una definición de variable remota implícita para cada variable exportada. La definición de variable remota tiene el mismo nombre y género que la variable exportada y se inserta en la unidad de ámbito en la cual se define el género. La definición de variable remota implícita no tiene la propiedad **nodelay**.

El <variable identifier> en <export> debe denotar una variable definida con **exported**.

Gramática gráfica concreta

<export area>::=
<task symbol> ***contains*** <export>

Semántica

Una <remote variable definition> introduce el nombre y género para variables importadas y exportadas.

Una definición de variable exportada es una definición de variable con la palabra clave **exported**.

La asociación entre una <imported variable specification> y una <exported variable definition> se establece refiriéndolas a la misma <remote variable definition>.

La instancia de proceso que posee una variable cuyos valores son exportados a otras instancias de proceso se denomina el exportador de la variable. Otras instancias de proceso que utilizan estos valores se denominan importadores de la variable. La variable se denomina variable exportada.

Una instancia de proceso puede ser importador y exportador a la vez, pero no puede importar desde, ni exportar hacia, el entorno.

a) *Operación de exportación*

Las variables exportadas tienen la palabra clave **exported** en sus <variable definition>s, y tienen una copia implícita para ser utilizada en operaciones de importación.

Una operación de exportación es la ejecución de una <export> por la cual un exportador revela el valor vigente de una variable exportada. Una operación de exportación causa el almacenamiento del valor vigente de la variable exportada en su copia implícita.

b) *Operación de importación*

Para cada <imported variable specification> en un importador hay un conjunto de variables implícitas, todas las cuales tienen un nombre único y el género dado en la <imported variable specification>. Estas variables implícitas se utilizan para el almacenamiento de valores importados.

Una operación de importación es la ejecución de una <import expression> por la cual un importador accede al valor de una variable exportada. El valor se almacena en una variable implícita denotada por el <import identifier> en la <import expression>. El exportador que contiene la variable importada es especificado por <destination> en <import expression>. Si no se especifica ningún <destination>, la importación procede de una instancia de proceso arbitraria que exporta la misma variable remota. La asociación entre la variable exportada en el exportador y la variable implícita en el importador se especifica refiriéndolas a la misma variable remota en la definición de variable de exportación y en la <imported variable specification>.

Modelo

Una operación de importación se modela mediante intercambio de señales. Estas señales son implícitas y se transportan por canales y rutas de señales implícitos. El importador envía una señal al exportador, y espera la contestación. En respuesta a esta señal el exportador devuelve al importador una señal con el valor contenido en la copia implícita de la variable exportada.

Si a la variable de exportación se vincula una inicialización por defecto, o si la variable de exportación es inicializada cuando es definida, se inicializa también la copia implícita, con el mismo valor que la variable de exportación.

Hay dos <signal definition>s implícitas para cada <remote variable definition> en una definición de sistema. Los <signal name>s en esas <signal definition>s se denotan respectivamente por *xQUERY* y *xREPLY* donde *x* denota el <name> de la <remote variable definition>. Las señales se definen en la misma unidad de ámbito que la <remote variable definition>. La copia implícita de la variable exportada se denota por *imcx*.

a) *Importador*

La <import expression> '**import** (x, destination)' se transforma en lo siguiente, donde se omite la cláusula **to** si no viene dado el destino:

output xQUERY **to** destination;
Wait in state xWAIT, saving all other signals;
input xREPLY (x);

En los demás estados se conserva, xREPLY.

Reemplazar la <import expression> por x, (el <name> de la variable implícita);

b) *Exportador*

A todos los <state>s del exportador, excluidos los estados implícitos derivados de import, se añade la <input part> siguiente:

input xQUERY;
output xREPLY (imcx) **to sender**/* next state the same*/

La <export> '**export** (x)' se transforma en lo siguiente:

task imcx := x,

NOTAS

1 Existe una posibilidad de bloqueo empleando el constructivo de importación, especialmente si no se indica <destination>, o si <destination> no denota una <PId expression> de un proceso cuya existencia está garantizada por la especificación en el momento de recibir la señal xQUERY.

2 El carácter optativo de las definiciones de variable remotas se introduce para la compatibilidad con el SDL-88, y no se recomienda para las nuevas descripciones SDL.

4.14 Procedimientos remotos

Un proceso de cliente puede llamar a un procedimiento definido en otro proceso mediante una petición al proceso servidor a través de una llamada a procedimiento remoto de un procedimiento en el proceso servidor.

Gramática textual concreta

<remote procedure definition> ::=
remote procedure <remote procedure name> [**nodelay**] <end>
[<procedure signature> <end>]

<imported procedure specification> ::=
imported procedure <remote procedure identifier> <end>
[<procedure signature> <end>]

<remote procedure call> ::=
call <remote procedure call body>

<remote procedure call body> ::=
<remote procedure identifier> [<actual parameters>]
[**to** <destination>]

<remote procedure input transition>::=
input [<virtuality>] **procedure**
<remote procedure identifier list> <end>
[<enabling condition>]
<transition>

<remote procedure save> ::=
save [<virtuality>] **procedure**
<remote procedure identifier list> <end>

<remote procedure identifier list> ::=
<remote procedure identifier> {,<remote procedure identifier>}*

nodelay denota canales implícitos sin retardo para el intercambio de señales resultante de la transformación descrita en *Modelo*.

La signatura de <imported procedure specification> debe ser la misma que la signatura en la <remote procedure definition> identificada por <remote procedure identifier>.

El <remote procedure identifier> que sigue a **as** en una definición de procedimiento exportado debe denotar una <remote procedure definition> con la misma signatura que el procedimiento exportado. En una definición de procedimiento exportado sin cláusula **as**, el nombre del procedimiento exportado está implícito y la <remote procedure definition> en el ámbito circundante más próximo con el mismo nombre está implícita.

Para cada <remote procedure call> debe existir una <imported procedure specification> del identificador de procedimiento remoto en un tipo de proceso, definición de proceso, tipo de servicio o definición de servicio circundante con el mismo <remote procedure name>.

Para cada especificación de procedimiento importado en un proceso debe existir por lo menos un procedimiento exportado en algún proceso.

Gramática gráfica concreta

<remote procedure call area> ::=

<procedure call symbol> ***contains*** <remote procedure call body>

<remote procedure input area>::=

<input symbol> ***contains***

{ [<virtuality>] **procedure** <remote procedure identifier list> }

is followed by { [<enabling condition area>] <transition area> }

<remote procedure save area> ::=

<save symbol> ***contains***

{[<virtuality>] **procedure** <remote procedure identifier list> <end> }

Semántica

Una <remote procedure definition> introduce el nombre y signatura para procedimientos importados y exportados.

Un procedimiento exportado es un procedimiento con la palabra clave **exported**.

La asociación entre una <imported procedure specification> y un procedimiento exportado se establece refiriéndolos a la misma <remote procedure definition>.

Una llamada a procedimiento remoto por un proceso *requesting* hace que el proceso de petición espere hasta que el proceso *server* haya ejecutado el procedimiento. Se conservan las señales enviadas al proceso de petición mientras espera. El proceso servidor ejecutará el procedimiento solicitado en el estado siguiente en el cual no está especificada la conservación del procedimiento, sujeto a la ordenación normal de la recepción de señales. Si no se especifica <remote procedure save> ni <remote procedure input transition> para un estado, se añade una transición implícita que consiste en la llamada a procedimiento únicamente y conduce de nuevo al mismo estado. Si una <remote procedure input transition> está especificada para un estado, se añade una transición implícita que consiste en la llamada a procedimiento seguida de <transition>. Si se especifica una <remote procedure save> para un estado, se añade una conservación implícita de la señal para el procedimiento solicitado.

Modelo

La llamada a procedimiento remoto es modelada por un intercambio de señales. Estas señales son implícitas y se transportan por canales y rutas de señales implícitos. El proceso *requesting* envía al proceso *server* una señal que contiene los parámetros reales de la llamada a procedimiento, y espera la respuesta. En respuesta a esa señal, el proceso servidor ejecuta el procedimiento remoto correspondiente, devuelve una señal al proceso solicitante con el valor de todos los parámetros **in**/**out**, y ejecuta la transición.

Hay dos <signal definition>s implícitas para cada <remote procedure definition>s en una <system definition>. Los <signal name>s en esas <signal definition>s son denotados por *pCALL* y *pREPLY* respectivamente, donde *p* es determinada de manera única. Las señales se definen en la misma unidad de ámbito que la <remote procedure definition>.

a) Proceso de petición

La <remote procedure call> '**call** Proc(apar) **to** destination' donde apar es la lista de parámetros reales, se transforma en lo siguiente, donde se omite la cláusula **to** si no se indica el destino:

output pCALL(apar) **to** destination;

Wait in state pWAITe, saving all other signals;

input pREPLY (aINOUTpar);

donde aINOUTpar es la lista modificada de parámetros **in**/**out** reales.

En todos los demás estados se conserva pREPLY.

b) Proceso servidor

Hay una declaración de una variable implícita, denominada ivar de género PId.

A todos los <state>s con una transición de entrada de procedimiento remoto, se añade la siguiente <input part>:

input pCALL(fpar);

ivar := **sender**

call Proc(fpar);

output pREPLY (fINOUTpar) **to** ivar;

<transition>

donde fpar y fINOUTpar son análogos a apar y aINOUT del caso anterior.

A todos los <state>s con una conservación de procedimiento remoto se añade la siguiente <save part>:

save pCALL;

A todos los demás <state>s, excluidos los estados implícitos derivados de import, se añade la siguiente <input part>:

input pCALL(fpar);

ivar := **sender**

call Proc(fpar);

output pREPLY (fINOUTpar) **to** ivar;

/* next state the same */

Una <remote procedure input transition> o <remote procedure input area> que contiene <virtuality> se denomina una transición de entrada virtual de procedimiento remoto. Una <remote procedure save> o <remote procedure save area> que contiene <virtuality> se denomina una conservación virtual de procedimiento remoto. La transición de entrada y la conservación de procedimiento remoto virtuales se describen en 6.3.3.

NOTA – Existe la posibilidad de bloqueo empleando el constructivo de procedimiento remoto, especialmente si no se indica <destination> o si <destination> no denota una <PId expression> de un proceso cuya existencia está garantizada por la especificación en el momento de recibirse la señal pCALL.

5 Datos en SDL

5.1 Introducción

Esta introducción presenta un bosquejo del modelo formal utilizado para definir tipos de datos e información sobre la forma en que está estructurado el resto de 5.

En un lenguaje de especificación es esencial permitir que los tipos de datos sean descritos formalmente en términos de su comportamiento, más bien que componiéndolos a partir de primitivas proporcionadas, como en algunos lenguajes de programación. Este último método implica invariablemente una realización particular del tipo de datos, y por tanto restringe la libertad de que dispone el realizador para elegir representaciones apropiadas del tipo de datos. El método del tipo abstracto de datos permite cualquier realización a condición de que sea factible y correcta con respecto a la especificación.

5.1.1 Abstracción en tipos de datos

Todos los datos utilizados en SDL se basan en tipos abstractos de datos que se definen en términos de sus propiedades abstractas más bien que en términos de alguna realización concreta. Ejemplos de definición de tipos abstractos de datos se presentan en el Anexo D, que define las facilidades predefinidas de datos del lenguaje.

Aunque todos estos tipos de datos son abstractos, y las facilidades predefinidas de datos pueden ser contraordenadas por el usuario, SDL intenta proporcionar un conjunto de facilidades predefinidas de datos que son familiares tanto en su comportamiento como en su sintaxis. Están predefinidos:

a) Booleano (Boolean)
b) Carácter (Character)
c) Cadena (String)
d) Cadena-de-caracteres (Charstring)
e) Entero (Integer)
f) Natural (Natural)
g) Real (Real)
h) Matriz (Array)
i) Conjuntista (Powerset)
j) PId (PId)
k) Duración (Duration)
l) Tiempo (Time).

El concepto de género estructurado (**struct**) puede utilizarse para formar objetos compuestos.

5.1.2 Bosquejo de formalismos utilizados para modelar datos

Los datos son modelados por un álgebra inicial. El álgebra tiene géneros designados y un conjunto de operadores que establecen relaciones de correspondencia entre los géneros. Cada género define la colección de todos los valores posibles que pueden ser generados por el conjunto conexo de operadores. Cada valor puede denotarse por al menos un término en el lenguaje que contiene solamente literales y operadores. Los literales constituyen un caso especial de operadores sin argumentos.

Los géneros y operadores, junto con el comportamiento del tipo de datos, forman las propiedades del tipo de datos. Un tipo de datos es introducido en cierto número de definiciones parciales de tipo, cada una de las cuales define un género y operadores, así como reglas algebraicas asociadas con ese género.

La palabra clave **newtype** introduce una definición parcial de tipo que define un género nuevo distinto. Se puede crear un género con propiedades heredadas de otro género, pero con identificadores diferentes para el género y los operadores.

La introducción de un sintipo nomina un subconjunto de los valores de un género.

Un generador es una descripción **newtype** incompleta: antes de asumir la condición de géneros, debe ser transformado en una definición parcial de tipo proporcionando la información que falta.

Algunos de los operadores introducidos por la definición parcial de tipo tienen una relación de correspondencia con el género, produciendo así valores (posiblemente nuevos) del género. Otros operadores dan significado al género mediante una relación de correspondencia con otros géneros definidos. Muchos operadores tienen una relación de correspondencia con el género booleano a partir de otros géneros, pero está estrictamente prohibido que estos operadores amplíen el género booleano.

5.1.3 Terminología

La terminología utilizada en 5 se elige de modo que esté en armonía con los trabajos publicados sobre álgebras iniciales. En particular «tipo de datos», se utiliza para hacer referencia a una colección de géneros más una colección de operadores asociados con esos géneros y la definición de las propiedades de estos géneros y operadores por ecuaciones algebraicas. Un «género» es un conjunto de valores con características comunes. Un «operador» es una relación entre géneros. Una «ecuación» es una definición de equivalencia entre términos de un género. Un «valor» es un conjunto de términos equivalentes. Un «axioma» es una ecuación que define un «término» booleano como equivalente a Verdadero. Sin embargo, «axiomas» se utiliza para referirse a un conjunto de «axioma»s o «ecuación»s.

5.1.4 División de texto en datos

El modelo de álgebra inicial utilizado para datos en SDL se describe en una forma que permite definir la mayor parte de los conceptos de datos en términos de un núcleo (kernel) de datos del lenguaje abstracto de datos SDL.

El texto de la cláusula 5 se divide en esta introducción (5.1), el lenguaje núcleo de datos (5.2), el uso pasivo de datos (5.3) y el uso activo de datos (5.4).

El lenguaje núcleo (kernel) de datos define la parte de datos en SDL que corresponde directamente con el método de álgebra inicial subyacente. El texto sobre álgebra inicial del Anexo C da una introducción más detallada a la base matemática de este método. Una formulación matemática más precisa se presenta en Apéndice I al Anexo C.

El uso pasivo de datos incluye las características implícitas y taquigráficas de datos SDL que permiten su uso para la definición de tipos abstractos de datos. Incluye también la interpretación de expresiones que no contienen valores asignados a variables. Estas expresiones «pasivas» corresponden a un uso funcional del lenguaje.

El uso activo de datos amplía el lenguaje de modo que incluye asignación. Esto a su vez incluye asignación a, uso de, e inicialización de variables. Cuando el SDL se utiliza para asignar valores a variables o acceder a valores de variables, se dice que se utiliza activamente. La diferencia entre expresiones activas y pasivas es que el valor de una expresión pasiva es independiente del momento en que se efectúa su interpretación, en tanto que la interpretación de una expresión activa puede dar valores diferentes según los valores asociados con las variables o el estado del sistema en cada instante.

Los datos predefinidos de SDL se definen en el Anexo D, que también define los géneros enumerados en 5.1.1.

5.2 El lenguaje núcleo de datos

El núcleo de datos puede utilizarse para definir tipos abstractos de datos.

5.2.1 Definiciones de tipos de datos

En un punto cualquiera de una especificación SDL hay una definición aplicable de tipo de datos. La definición de tipo de datos define la validez de expresiones y la relación entre expresiones. La definición introduce operadores y conjuntos de valores (géneros).

No hay una correspondencia simple entre la sintaxis concreta y la abstracta para definiciones de tipos de datos pues la sintaxis concreta introduce incrementalmente la definición de tipo de datos con énfasis en los géneros (véase también el Anexo C).

Las definiciones en la sintaxis concreta son a menudo independientes y no pueden dividirse en unidades de ámbito diferentes. Por ejemplo:

```
newtype even literals 0;
   operators
     plusee: even, even   -> even;
     plusoo: odd, odd  -> even;
   axioms
        plusee(a,0)  == a;
        plusee(a,b)  == plusee(b,a);
        plusoo(a,b)  == plusoo(b,a);
endnewtype even; /* even "numbers" with plus–depends on odd */

newtype odd literals 1;
   operators
     plusoe: odd, even -> odd;
     pluseo: even, odd -> odd;
   axioms
        plusoe(a,0)  == a;
        pluseo(a,b)  == plusoe(b,a);
endnewtype odd; /*odd "numbers" with plus - depends on even*/
```

Cada definición de tipo de datos es una definición completa. No hay referencias a géneros u operadores que no estén incluidos en la definición de tipo de datos que se aplica en un punto dado. Asimismo, una definición de tipo de datos no podrá invalidar la semántica de la definición de tipo de datos en la unidad de ámbito inmediatamente circundante. Un tipo de datos en una unidad de ámbito encerrada sólo enriquece operadores de géneros definidos en la unidad de ámbito exterior. Un «valor» de un género definido en una unidad de ámbito puede utilizarse libremente y pasarse entre unidades de ámbito jerárquicamente inferiores o a partir de estas unidades. Puesto que los datos predefinidos están definidos en un paquete predefinido y utilizado implícitamente, los géneros predefinidos (por ejemplo booleano y entero) pueden utilizarse libremente en todo el sistema.

Gramática abstracta

Data-type-definition	::	*Sorts* *Signature*-**set** *Equations*
Sorts	=	*Sort-name*-**set**
Sort-name	=	*Name*
Equations	=	*Equation*-**set**

Una *data type definition* no agregará nuevos valores a ningún *sort* del *data type* de la <scope unit kind> circundante.

Si un *term* (véase 5.2.3) es inequivalente a otro *term* de acuerdo con el tipo de datos que se aplica en la <scope unit kind> circundante, estos *terms* no serán definidos como equivalentes por la *data type definition*.

Gramática textual concreta

<partial type definition> ::=
newtype <sort name>
[<formal context parameters>]
[<extended properties>] <properties expression>
endnewtype [<sort name>]

<properties expression> ::=
<operators>
{ <internal properties>
| <external properties> }
[<default initialization>]

<internal properties> ::=
[<operator definitions>]
[**axioms** <axioms>] [<literal mapping>]

Un <formal context parameter> de <formal context parameters> debe ser un <sort context parameter> o un <synonym context parameter>.

<formal context parameters>, <extended properties>, <operator definitions>, <external properties>, <literal mapping> y <default initialization> no forman parte del núcleo de datos.

La *data type definition* se representa por la colección de todas las <partial type definition>s en la <scope unit kind> vigente combinada con la *data type definition* de la <scope unit kind> circundante.

Cada una de las siguientes <scope unit kind>s representa un ítem en la sintaxis abstracta que contiene una *data type definition*: <system definition>, <block definition>, <process definition>, <service definition>, <procedure definition>, <channel substructure definition> o <block substructure definition> o los diagramas correspondientes en sintaxis gráfica. La <partial type definition> en una <system type definition>, <block type definition>, <process type definition> o <service type definition> se transforma en una <partial type definition> en la (el conjunto de) instancia(s), resultante(s) cuando el tipo es instanciado.

Para una <scope unit kind> los *sorts* se representan por el conjunto de <sort name>s introducido por el conjunto de <partial type definition>s de la <scope unit kind>.

Para una <scope unit kind> el conjunto *signature* y las *equations* se representan por las <properties expression>s de las <partial type definition>s de la <scope unit kind>.

Los <operators> de una <properties expression> representan parte del conjunto *signature* en la sintaxis abstracta. El conjunto *signature* completo es la unión de los conjuntos *signature* definidos por las <partial type definition>s en la <scope unit kind>.

Los <axioms> de una <properties expression> representan parte del conjunto e*quation* en la sintaxis abstracta. Las *equations* son la unión de los conjuntos *equation* definidos por las <partial type definition>s en la <scope unit kind>.

Los géneros de datos predefinidos tienen sus <partial type definition>s implícitas en el paquete Predefined definido en el Anexo D.

Semántica

La definición de tipo de datos define un tipo de datos. Un tipo de datos tiene un conjunto de propiedades de tipo, es decir: un conjunto de géneros, un conjunto de operadores y un conjunto de ecuaciones.

Las propiedades de tipos de datos se definen en la sintaxis concreta mediante definiciones parciales de tipo. Una definición parcial de tipo no introduce todas las propiedades de un tipo de datos sino, solamente, define parcialmente algunas de las propiedades. Las propiedades completas de un tipo de datos se hallan examinando la combinación de todas las definiciones parciales de tipo que se aplican dentro de la unidad de ámbito que contiene la definición de tipo de datos.

Un género es un conjunto de valores de datos. Dos géneros diferentes no tienen valores en común.

La definición de tipo de datos se forma a partir de la data de tipo de datos de la unidad de ámbito que define la unidad de ámbito vigente tomada conjuntamente con los géneros, operadores y ecuaciones definidos en la unidad de ámbito vigente.

Salvo dentro de una <partial type definition> o un <signal refinement>, la definición de tipo de datos que se aplica en cualquier punto es el tipo de datos definido para la unidad de ámbito que circunda inmediatamente ese punto. Dentro de una <partial type definition> o un <signal refinement>, la definición de tipo de datos que se aplica es la definición de tipo de datos de la unidad de ámbito que circunda a la <partial type definition> o <signal refinement> respectivamente.

El conjunto de géneros de un tipo de datos es la unión del conjunto de géneros introducido en la unidad de ámbito vigente y el conjunto de géneros del tipo de datos que se aplica en la <scope unit kind> circundante. El conjunto de operadores de un tipo de datos es la unión del conjunto de operadores introducidos en la unidad de ámbito vigente y el conjunto de operadores del tipo de datos que se aplica en la <scope unit kind> circundante. El conjunto de ecuaciones de un tipo de datos es la unión del conjunto de ecuaciones introducido en la unidad de ámbito vigente y el conjunto de ecuaciones del tipo de datos que se aplica en la <scope unit kind> circundante.

Cada género introducido en una definición de tipo de datos tiene un identificador que es el nombre introducido por una definición parcial de tipo en la unidad de ámbito calificada por el identificador de la unidad de ámbito.

5.2.2 Literales y operadores parametrizados

Gramática abstracta

Signature	=	*Literal-signature* | *Operator-signature*
Literal-signature	::	*Literal-operator-name* *Result*
Operator-signature	::	*Operator-name* *Argument-list* *Result*
Argument-list	=	*Sort-reference-identifier* $^{+}$
Result	=	*Sort-reference-identifier*
Sort-reference-identifier	=	*Sort-identifier* | *Syntype-identifier*
Literal-operator-name	=	*Name*
Operator-name	=	*Name*
Sort-identifier	=	*Identifier*

Los sintipos y *syntype identifier*s no forman parte del núcleo (véase 5.3.1.9).

Gramática textual concreta

<operators> ::=
[<literal list>] [<operator list>]

<literal list> ::=
literals <literal signature> {, <literal signature> }* [<end>]

<literal signature> ::=
<literal operator name>
| <extended literal name>

<operator list> ::=
operators
<operator signature> { <end> <operator signature> }* [<end>]

<operator signature> ::=
<operator name> : <argument list> -> <result>
| <ordering>
| <noequality>

<operator name> ::=
<operator name>
| <extended operator name>

<argument list> ::=
<argument sort> { , <argument sort> }*

<argument sort> ::=
<extended sort>

<result> ::=
<extended sort>

<extended sort> ::=
<sort>
| <generator sort>

<sort> ::=
<sort identifier>
| <syntype>

Las alternativas <extended operator name>, <extended literal name>, <ordering>, <generator sort>, <syntype> y <noequality> no forman parte del núcleo de datos.

Los literales se introducen mediante <literal signature>s enumeradas después de la palabra clave **literals**. El *result* de una *literal signature* es el género introducido por la <partial type definition> que define el literal.

Cada <operator signature> en la lista de <operator signature>s enumeradas después de la palabra clave **operators** representa una *operator signature* con un *operator name*, una *argument list* y un *result*.

El <operator name> corresponde a un *operator name* en la sintaxis abstracta que es único dentro de la unidad de ámbito definidora, aunque el nombre puede no ser único en la sintaxis concreta.

El *Operator-name* o *Literal-operator-name* único en la sintaxis abstracta se deriva de:

a) el <operator name> (o <literal operator name>); más

b) la lista de identificadores de género de argumento; más

c) el identificador de género de resultado; más

d) el identificador de género de la definición parcial de tipo en que está definido el <operator name> (o <literal operator name>).

Siempre que se especifica un <operator identifier>, el *Operator-name* único en el *Operator-identifier* se deriva de la misma manera con la lista de géneros de argumento y el género de resultado derivado del contexto. Dos operadores que tienen el mismo <name> y que difieren en uno o más de los géneros de argumento o resultado, tienen *names* diferentes.

Cada <argument sort> en una <argument list> representa un *Sort-reference-identifier* en una *Argument-list*. Un <result> representa el *Sort-reference-identifier* de un *Result*.

Dondequiera que un <qualifier> de un <operator identifier> (o <literal operator identifier>) contiene un <path item> con la palabra clave **type**, el <sort name> después de la palabra clave no forma parte del *Qualifier* del *Operator-identifier* (o *Literal-operator-identifier*) pero se utiliza para derivar el *Name* único del *Identifier*. En este caso el *Qualifier* se forma a partir de la lista de <path item>s que precede a la palabra clave **type**.

Semántica

Un operador es «total», lo que significa que la aplicación del operador a cualquier lista de valores de los géneros de argumento denota un valor del género de resultado.

Una signatura de operador define cómo puede utilizarse el operador en expresiones. La signatura de operador es la identidad de operador más la lista de géneros de los argumentos y el género del resultado. La signatura de operador la que determina si una expresión es una expresión válida en el lenguaje de acuerdo con las reglas requeridas para la comparación de los géneros de expresiones de argumento.

Un operador sin argumento se denomina un literal.

Un literal representa un valor fijo, perteneciente al género resultado del operador.

Un operador tiene un género resultado que es el género identificado por el resultado.

NOTA – Directriz: una <operator signature> debe mencionar el género introducido por la <partial type definition> circundante bien como un <argument sort>, o como un <result>.

Ejemplo 1

```
literals   free, busy ;
```

Ejemplo 2

```
operators
     findstate:   Telephone -> Availability;
```

Ejemplo 3

```
literals
     empty_list
operators
     add_to_list: list_of_telephones, Telephone -> list_of_telephones;
     sub_list:    list_of_telephones, Telephone -> list_of_telephones
```

5.2.3 Axiomas

Los axiomas determinan qué términos representan el mismo valor. A partir de los axiomas de una definición de tipo de datos se determina la relación entre valores de argumento y valores de resultado de operadores, dando así significado a los operadores. Los axiomas se dan como axiomas booleanos o en forma de ecuaciones de equivalencia algebraica.

Gramática abstracta

Equation	=	*Unquantified-equation* | *Quantified-equations* | *Conditional-equation* | *Informal-text*
Unquantified-equation	::	*Term Term*
Quantified-equations	::	*Value-name*-**set** *Sort-identifier* *Equations*
Value-name	=	*Name*
Term	=	*Ground-term* | *Composite-term* | *Error-term*
Composite-term	::	*Value-identifier* | *Operator-identifier Term*$^{+}$ | *Conditional-composite-term*
Value-identifier	=	*Identifier*
Operator-identifier	=	*Identifier*
Ground-term	::	*Literal-operator-identifier* | *Operator-identifier Ground-term*$^{+}$ | *Conditional-ground-term*
Literal-operator-identifier	=	*Identifier*

Las alternativas *Conditional-composite-term* y *Conditional-ground-term* en las reglas *Composite-term* y *Ground-term* respectivamente no forman parte del núcleo de datos, aunque las ecuaciones que contienen estos términos pueden

reemplazarse por ecuaciones semánticamente equivalentes escritas en el lenguaje núcleo (véase 5.3.1.6). La alternativa *Error-term* en la regla *Term* no forma parte del núcleo de datos.

Las definiciones de *informal text* y *conditional equations* se dan en 2.2.3 y 5.2.4 respectivamente.

Cada *term* (o *ground term*) en la lista de términos después de un *operator identifier* debe tener el mismo género que el género correspondiente (por posición) en la *argument list* de la *operator signature*.

Los dos *terms* de una *unquantified equation* tienen que ser del mismo género.

Gramática textual concreta

```
<axioms> ::=
                    <equation> { <end> <equation>}* [ <end> ]

<equation> ::=
                    <unquantified equation>
                |   <quantified equations>
                |   <conditional equation>
                |   <informal text>

<quantified equations> ::=
                    <quantification> ( <axioms> )

<quantification> ::=
                    for all <value name> { , <value name> }*
                    in <extended sort>

<unquantified equation> ::=
                    <term> == <term>
                |   <Boolean axiom>

<term> ::=
                    <ground term>
                |   <composite term>
                |   <error term>
                |   <spelling term>

<composite term> ::=
                    <value identifier>
                |   <operator identifier> ( <composite term list> )
                |   ( <composite term> )
                |   <extended composite term>

<composite term list> ::=
                    <composite term> { , <term> }*
                |   <term> , <composite term list>

<ground term> ::=
                    <literal identifier>
                |   <operator identifier> ( <ground term> { , <ground term> }* )
                |   ( <ground term> )
                |   <extended ground term>

<literal identifier> ::=
                    <literal operator identifier>
                |   <extended literal identifier>
```

Las alternativas <Boolean axiom> de la regla <unquantified equation>, <error term> y <spelling term> de la regla <term>, <extended composite term> de la regla <composite term>, <extended ground term> de la regla <ground term>, y <extended literal identifier> de la regla <literal identifier> no forman parte del núcleo de datos.

El <sort> en una <quantification> representa el *sort identifier* en *quantified equations* . Los <value name>s en una <quantification> representan el conjunto *value name* en *quantified equations*.

Una <composite term list> representa una lista de *term*. Un *operator identifier* seguido por una lista de *term* es solamente un *composite term* si la lista de *term* contiene al menos un *value identifier*.

Un <identifier> que es un nombre incalificado que aparece en un <term> representa:

a) un *operator identifier* si precede a un paréntesis redondo de apertura (o es un <operator name> que es un <extended operator name> (véase 5.3.1); de no ser así,

b) un *value identifier* si hay una definición de ese nombre en una <quantification> de <quantified equations> que encierra el <term> de un género adecuado para el contexto; de no ser así,

c) un *literal operator identifier* si hay un literal visible con ese nombre, que debe tener un género adecuado para el contexto; de no ser así,

d) un *value identifier* que tiene una *quantified equation* implícita en la sintaxis abstracta para la <unquantified equation>.

Dos o más ocurrencias del mismo identificador de valor no vinculado dentro de una <unquantified equation>, o si la <unquantified equation> está contenida en una <conditional equation> que se encuentra entonces dentro de la <conditional equation>, entrañan una *quantification*.

Un *operator identifier* se deriva del contexto, de modo que si el <operator name> está sobrecargado (es decir, si se utiliza el mismo <name> para más de un operador), es el *operator name* que identifica un operador visible con el mismo nombre y con géneros de argumento y género de resultado consistentes con la aplicación del operador. Si el <operator name> está sobrecargado, puede que sea necesario derivar los géneros de argumento a partir de los argumentos y el género de resultado a partir del contexto para determinar el *operator name*.

Dentro de una <unquantified equation> tiene que haber exactamente un género para cada identificador de valor implícitamente cuantificado, que sea consistente con todos los usos.

Tiene que ser posible vincular cada <operator identifier> o <literal operator identifier> incalificado a exactamente un *operator identifier* o *literal operator identifier* definido que satisfaga las condiciones en la construcción en la cual se utiliza el <identifier>. Esto es: la vinculación tiene que ser única.

NOTA – Directriz: un axioma debe ser pertinente para el género de la definición parcial de tipo circundante al mencionar un operador o literal con un resultado de este género o un operador que tenga un argumento de este género; un axioma debe definirse sólo una vez.

Semántica

Cada ecuación es un enunciado sobre la equivalencia algebraica de términos. El término del lado izquierdo y el término del lado derecho se enuncian como equivalentes, de modo que cuando aparece un término puede ser sustituido por el otro. Cuando un identificador de valor aparece en una ecuación, puede ser sustituido simultáneamente en esa ecuación por el mismo término para cada ocurrencia del identificador de valor. Para esta sustitución, el término puede ser cualquier término fundamental del mismo género que el identificador de valor.

Se introducen identificadores de valor por los nombres de valor en ecuaciones cuantificadas. Un identificador de valor se utiliza para representar cualquier valor de datos que pertenezca al género de la cuantificación. Una ecuación queda satisfecha si el mismo valor reemplaza simultáneamente cada ocurrencia del identificador de valor en la ecuación, independientemente del valor elegido para la sustitución.

Un término fundamental es un término que no contiene ningún identificador de valor. Un término fundamental representa un valor particular, conocido. Para cada valor en un género existe al menos un término fundamental que representa ese valor.

Si uno o más axiomas cualesquiera contienen texto informal, la interpretación de expresiones no está formalmente definida por SDL pero puede ser determinada a partir del texto informal por el interpretador. Se parte del supuesto de que si se especifica texto informal, se sabe que el conjunto de ecuaciones está incompleto; por lo tanto, no se ha dado una especificación formal completa en SDL.

Un nombre de valor es siempre introducido por ecuaciones cuantificadas en la sintaxis abstracta, y el valor correspondiente tiene un identificador de valor que es el nombre de valor calificado por el identificador de género de las ecuaciones cuantificadas circundantes. Por ejemplo:

```
for all z,z in X (for all z in X _ )
```

introduce sólo un identificador denominado z de género X.

En la sintaxis concreta no está permitido especificar un calificador para identificadores de valor.

Cada identificador de valor introducido por ecuaciones cuantificadas tiene un género que es el identificado en las ecuaciones cuantificadas por el *sort reference identifier*. El género de las cuantificaciones implícitas es el género requerido por el contexto (o los contextos) de la ocurrencia del identificador no vinculado. Si los contextos de un identificador de valor que tiene una cuantificación implícita permiten géneros diferentes, el identificador está vinculado a un género que es consistente con todos sus usos en la ecuación.

Un término tiene un género que es el género del identificador de valor o el género resultado del operador (literal).

A menos que pueda deducirse de las ecuaciones que dos términos denotan el mismo valor, cada término denotará un valor diferente.

Ejemplo 1

for all b **in** logical (eq(b,b)==T)

Ejemplo 2

neq(T, F)== T; neq(T, T) == F;
neq(F, T)== T; neq(F, F) == F;

Ejemplo 3

eq(b, b) == T;
eq(F, eq(T, F)) == T;
eq(eq(b, a),eq(a, b)) == T;

5.2.4 Ecuaciones condicionales

Una ecuación condicional permite la especificación de ecuaciones que sólo son aplicables cuando se cumplen ciertas restricciones. Las restricciones se escriben en forma de ecuaciones simples.

Gramática abstracta

Conditional-equation	::	*Restriction*-**set** *Restricted-equation*
Restriction	=	*Unquantified-equation*
Restricted-equation	=	*Unquantified-equation*

Gramática textual concreta

<conditional equation> ::=
<restriction> { , <restriction> }* ==> <restricted equation>

<restricted equation> ::=
<unquantified equation>

<restriction> ::=
<unquantified equation>

Semántica

Una ecuación condicional establece que unos términos denotan el mismo valor solamente cuando cualquier identificador de valor en las ecuaciones restringidas denota un valor con relación al cual se puede demostrar, mediante otras ecuaciones, que satisface la restricción.

La semántica de un conjunto de ecuaciones para un tipo de datos que incluye ecuaciones condicionales se deriva de la manera siguiente:

a) La cuantificación se suprime generando todas las ecuaciones de términos fundamentales que puedan derivarse de las ecuaciones condicionales. Como esto se aplica a la cuantificación explícita e implícita, se genera un conjunto de ecuaciones no cuantificadas en términos fundamentales.

b) Una ecuación condicional se denominará una ecuación condicional comprobable si puede demostrarse que todas las restricciones (únicamente en términos fundamentales) se cumplen a partir de ecuaciones no cuantificadas que no son ecuaciones restringidas. Si existe una ecuación condicional comprobable, se sustituye por la ecuación restringida de la ecuación condicional comprobable.

c) Si quedan ecuaciones condicionales en el conjunto de ecuaciones, y ninguna de esas ecuaciones condicionales es una ecuación condicional comprobable, se suprimen esas ecuaciones condicionales, y si no, se vuelve al paso b).

d) El conjunto de ecuaciones no cuantificadas restantes define la semántica del tipo de datos.

Ejemplo

```
z /= 0 == True ==> (x/z)*z==x
```

5.3 Uso pasivo de datos SDL

En 5.3.1 se definen ampliaciones de los constructivos de definición de datos indicadas en 5.2. La manera de interpretar el uso de tipos abstractos de datos en expresiones se define en 5.3.3 para la expresión «pasiva» (es decir, cuando no depende de variables ni del estado del sistema). La manera de interpretar expresiones que no son pasivas (es decir, expresiones «activas») se define en 5.4.

5.3.1 Constructivos ampliados de definición de datos

Los constructivos definidos en 5.2 son la base de formas más concisas explicadas más abajo.

Gramática abstracta

No hay una sintaxis abstracta adicional para la mayor parte de estos constructivos. En 5.3.1 y en todos sus apartados la sintaxis abstracta pertinente es generalmente la descrita en 5.2.

Gramática textual concreta

```
<extended properties> ::=
                        <inheritance rule>
                |       <generator transformations>
                |       <structure definition>

<extended composite term> ::=
                        <extended operator identifier> ( <composite term list> )
                |       <composite term> <infix operator> <term>
                |       <term> <infix operator> <composite term>
                |       <monadic operator> <composite term>
                |       <conditional composite term>
```

<extended ground term> ::=
<extended operator identifier>
(<ground term> {, <ground term> }*)
| <ground term> <infix operator> <ground term>
| <monadic operator> <ground term>
| <conditional ground term>

<extended operator identifier> ::=
<operator identifier> <exclamation>
| <generator formal name>
| [<qualifier>] <quoted operator>

<extended operator name> ::=
<operator name> <exclamation>
| <generator formal name>
| <quoted operator>

<exclamation> ::=
!

<extended literal name> ::=
<character string literal>
| <generator formal name>
| <name class literal>

<extended literal identifier> ::=
<character string literal identifier>
| <generator formal name>

Alternativas con <generator formal name>s sólo son válidas en una <properties expression> en un <generator text> (véase 5.3.1.12) que tenga ese nombre definido como un parámetro formal.

Las alternativas con <exclamation> no son válidas en <operator definitions>.

Las alternativas de <extended composite term> y <extended ground term> con un <generator formal name> que precede un «(» sólo son válidas si el <generator formal name> se define de modo que sea la clase **operator** (véase 5.3.1.12).

La alternativa de <extended literal name> con un <generator formal name> sólo es válida si el <generator formal name> está definido de modo que sea de la clase **literal** (véase 5.3.1.12).

La alternativa de <extended literal identifier> con un <generator formal name> sólo es válida si el <generator formal name> está definido de modo que sea de la clase **literal** o de la clase **constant** (véase 5.3.1.12).

Si un nombre de operador está definido con una <exclamation>, la <exclamation> forma semánticamente parte del *name*.

Las formas <operator name> <exclamation> o <operator identifier> <exclamation> representan *operator name* (5.2.2) y *operator identifier* (5.2.3) respectivamente.

Semántica

Un nombre de operador definido con una <exclamation> tiene la semántica normal de un operador, pero el nombre de operador sólo es visible en axiomas y en <inheritance list>s.

5.3.1.1 Operadores especiales

Estos son nombres de operador que tienen formas sintácticas especiales. La sintaxis especial se introduce de modo que los operadores aritméticos y operadores booleanos puedan tener su forma sintáctica usual. Es decir, el usuario puede escribir «(1 + 1) = 2» y no está forzado a utilizar por ejemplo equal(add(1,1),2). Los géneros que serán válidos para cada operador dependerán de la definición de tipo de datos.

Gramática textual concreta

<quoted operator> ::=
<quote> <infix operator> <quote>
| <quote> <monadic operator> <quote>

<quote> ::=
"

<infix operator> ::=
=> | **or** | **xor** | **and** | **in** | /= | = | > | < | <=
| >= | + | / | * | // | **mod** | **rem** | -

<monadic operator> ::=
- | **not**

Semántica

Un operador infijo es un término que tiene la semántica normal de un operador pero con sintaxis de infijo o de prefijo entre comillas, como se ha indicado anteriormente.

Un operador monádico es un término que tiene la semántica normal de un operador pero con la sintaxis de prefijo o prefijo entre comillas, como se ha indicado anteriormente.

Las formas entre comillas de operadores infijo o monádico son nombres válidos para operadores.

Los operadores infijos tienen un orden de precedencia que determina la vinculación de operadores. La vinculación es la misma que la vinculación en <expression>s, como se especifica en 5.3.3.1.

Cuando la vinculación es ambigua, como lo es en

a **or** b **xor** c ;

entonces va de izquierda a derecha, de modo que este término es equivalente a

(a **or** b) **xor** c ;

Modelo

Un término de la forma <term1> <infix operator> <term2> es sintaxis derivada para «<infix operator>» (<term1>, <term2>) con «<infix operator>» como un nombre lícito. «<infix operator>» representa un *operator name*.

De manera similar <monadic operator> <term> es sintaxis derivada para «<monadic operator>» (<term>) con «<monadic operator>» como un nombre lícito que representa un *operator name*.

NOTA – El operador de igualdad (=) no debe confundirse con el símbolo de equivalencia de términos (==).

5.3.1.2 Literales cadena de caracteres

Gramática textual concreta

<character string literal identifier> ::=
[<qualifier>] <character string literal>

<character string literal> ::=
<character string>

Una <character string> es una unidad léxica.

Un <character string literal identifier> representa un *Literal-operator-identifier* en la sintaxis abstracta.

Un <character string literal> representa un *Literal-operator-name* (5.2.2) en la sintaxis abstracta derivada de la <character string>.

Semántica

Identificadores de literal cadena de caracteres son los identificadores formados a partir de literales cadena de caracteres en términos y expresiones.

Los literales cadena de caracteres se utilizan para los géneros de datos predefinidos cadena-de-caracteres (Charstring) y carácter (Character) (véase el Anexo D). Tienen también una relación especial con los literales clase de nombre (véase 5.3.1.14) y correspondencias de literales (véase 5.3.1.15). Estos literales pueden también definirse de modo que tengan otros usos.

Un <character string literal> tiene una longitud que es el número de <alphanumeric>s más <other character>s más <special>s más <full stop>s más <underline>s más <space>s más pares <apostrophe> <apostrophe> en la <character string> (véase 2.2.1).

Si un <character string literal>

a) tiene una longitud superior a 1; y

b) tiene una subcadena formada por la supresión del último carácter (<alphanumeric> u <other character> o <special> o <full stop> o <underline> o <space> o pares <apostrophe> <apostrophe>) de la <character string>; y

c) tiene una subcadena definida como un literal tal que

subcadena // carácter_suprimido_entre_apóstrofos

es un término válido con el mismo género que el <character string literal>, donde el operador de concatenación y sus argumentos están calificados en el género que los contiene,

tiene una ecuación implícita dada por la sintaxis concreta según la cual el <character string literal> es equivalente a la subcadena seguida por el operador infijo «//» seguido por el carácter suprimido con apóstrofos para formar una <character string>.

Por ejemplo los literales 'ABC', 'AB'', y 'AB' en

```
newtype s
literals 'ABC', 'AB''', 'AB', 'A', 'B', 'C', '''';
operators "//": s, s -> s;
```

tienen las ecuaciones implícitas

```
'ABC'   == 'AB' // 'C';
'AB''   == 'AB' // '';
'AB'    == 'A' // 'B';
```

5.3.1.3 Datos predefinidos

Los datos predefinidos, incluyendo el género booleano que define propiedades para dos literales, True (Verdadero) y False (Falso), se definen en el Anexo D. La semántica de igualdad (5.3.1.4), los axiomas booleanos (5.3.1.5), los términos condicionales (5.3.1.6), la ordenación (5.3.1.8), y los sintipos (5.3.1.9) se basan en la definición del género booleano (D.1). La semántica de los literales clase de nombre (si se utilizan <regular interval>s – 5.3.1.14) y la correspondencia de literales (5.3.1.15) se basan también en la definición de carácter (Character) (D.2) y cadena-de-caracteres (Charstring) (D.4) respectivamente.

Además, los dos *términos* booleanos Verdadero y Falso no deben definirse (directa ni indirectamente) como equivalentes. Cada expresión fundamental booleana que se utiliza fuera de definiciones de tipo de datos tiene que interpretarse bien como Verdadero o como Falso. Si no es posible reducir tal expresión a Verdadero o Falso, la especificación es incompleta y permite más de una interpretación del tipo de datos.

Tampoco puede reducirse el número de valores para el género predefinido PId.

Los datos predefinidos se definen en un paquete Predefined utilizado implícitamente (véase 2.4.1.2). Este paquete se define en el Anexo D.

5.3.1.4 Igualdad y no igualdad

Gramática textual concreta

<noequality> ::=

noequality

Semántica

Los símbolos = y /= en la sintaxis concreta representan los nombres de los operadores que son denominados los operadores igual y no igual.

Modelo

Cualquier <partial type definition> que introduce un género denominado S sin <inheritance rule> tiene implícito un par *operator signature* equivalente a:

"=" : S, S -> << **package** Predefined >> Boolean;

"/=": S, S -> << **package** Predefined >> Boolean;

donde Boolean es el género booleano predefinido.

Cualquier <partial type definition> que introduzca un género denominado S, a menos que:

a) tenga la palabra clave **noequality** en su <operator list>; o

b) esté basado en herencia sin la palabra clave **noequality** en su <inheritance list>

tiene un conjunto de ecuaciones implícito:

```
for all a,b,c in S (
   a = a == True;
   a = b == b = a;
   ((a = b) and (b = c)) ==> a = c == True;
   a /= b == not(a = b);
   a = b  == True ==> a == b;)
```

y una <literal equation> implícita:

```
for all L1,L2 in S literals (
   Spelling(L1) /= Spelling(L2) ==> L1 = L2 == False;)
```

5.3.1.5 Axiomas booleanos

Gramática textual concreta

<Boolean axiom> ::=

<Boolean term>

Semántica

Un axioma booleano es una sentencia de verdad que se cumple para todas las condiciones del tipo de datos que se está definiendo, y por tanto puede utilizarse para especificar el comportamiento del tipo de datos.

Modelo

Un axioma de la forma

terme <Boolean term>;

es sintaxis derivada para la ecuación de sintaxis concreta

<Boolean term> == << **package** Predefined/**type** Boolean >> True;

que tiene la relación normal de una ecuación con la sintaxis abstracta.

5.3.1.6 Términos condicionales

En lo sucesivo, la ecuación que contiene el término condicional se denomina ecuación de término condicional.

Gramática abstracta

Conditional-composite-term	=	*Conditional-term*
Conditional-ground-term	=	*Conditional-term*
Conditional-term	::	*Condition* *Consequence* *Alternative*
Condition	=	*Term*
Consequence	=	*Term*
Alternative	=	*Term*

El género de *Condition* tiene que ser el género booleano predefinido, y la *Condition* no debe ser el *Error-term*. La *Consequence* y la *Alternative* tienen que ser del mismo género.

Un *conditional term* es un *conditional composite term* si, y sólo si, uno o más de los *terms* en la *condition*, la *Consequence* o la *Alternative* son *Composite-term*.

Un *conditional term* es un *conditional ground term* si, y sólo si, todos los *terms* en la *condition*, la *consequence* o la *alternative* son *ground terms*.

Gramática textual concreta

<conditional composite term> ::=
<conditional term>

<conditional ground term> ::=
<conditional term>

<conditional term> ::=
if <condition> **then** <consequence> **else** <alternative> **fi**

<condition> ::=
<Boolean term>

<consequence> ::=
<term>

<alternative> ::=
<term>

Semántica

Una ecuación que contiene un término condicional es semánticamente equivalente a un conjunto de ecuaciones en los cuales se han eliminado todos los identificadores de valor cuantificado en el término booleano. Este conjunto de ecuaciones puede formarse sustituyendo simultáneamente en toda la ecuación de término condicional cada *value identifier* en la *condition* por cada *ground term* del género apropiado. En este conjunto de ecuaciones, la *condition* siempre habrá sido reemplazada por un *ground term* booleano. En lo sucesivo, este conjunto de ecuaciones será el conjunto fundamental ampliado.

Una ecuación de término condicional es equivalente al conjunto de *equations* que contiene:

a) para cada *equation* del conjunto fundamental ampliado en la cual la *condition* es equivalente a Verdadero, esa *equation* del conjunto fundamental ampliado con el *conditional term* reemplazado por la *consequence* (fundamental); y

b) para cada *equation* del conjunto fundamental ampliado en la cual la *condition* es equivalente a Falso, esa *equation* del conjunto fundamental ampliado con el *conditional term* reemplazado por la *alternative* (fundamental).

Obsérvese que el caso especial de una ecuación de la forma

ex1 == **if** a **then** b **else** c **fi**;

es equivalente al par de ecuaciones condicionales

a == True ==> ex1 == b;

a == False ==> ex1 == c.

Ejemplo

```
if i = j * j then posroot(i) else abs(j) fi ==
if positive(j) then j else -j fi;
```

NOTA – Hay formas mejores de especificar estas propiedades; éste es sólo un ejemplo.

5.3.1.7 Errores

Se utilizan errores para poder definir completamente las propiedades de un tipo de datos incluso en casos en que no se puede dar un significado específico al resultado de un operador.

Gramática abstracta

Error-term :: ( )

Un *error term* no debe utilizarse como un término de argumento para un *operator identifier* en un *composite term*.

Un *error term* no podrá utilizarse como parte de una *restriction*.

No será posible derivar de *Equations* que un *literal operator identifier* es igual a un *error term*.

Gramática textual concreta

<error term> ::=

error <exclamation>

Semántica

Un término puede ser un error, por lo que es posible especificar las circunstancias en las cuales un operador produce un error. Si estas circunstancias surgen durante la interpretación, el comportamiento ulterior del sistema está indefinido.

5.3.1.8 Ordenación

Gramática textual concreta

<ordering> ::=
ordering

Semántica

La palabra clave para ordenación es una notación taquigráfica para especificar explícitamente operadores de ordenación y un conjunto de ecuaciones de ordenación para una definición parcial de tipo.

Modelo

Una <partial type definition> que introduce un género denominado S con la palabra clave **ordering** implica un *operator signature* equivalente a las definiciones explícitas:

```
"<" : S,S -> package Predefined Boolean;
">" : S,S -> package Predefined Boolean;
"<=": S,S -> package Predefined Boolean;
">=": S,S -> package Predefined Boolean;
```

donde Boolean es el género booleano predefinido, e implica también los *Axioms* booleanos:

```
for all a, b, c, d in S
(  a = b        ==> a < b  == False;
a < b             == b > a;
a <= b            == a < b or a = b;
a >= b            == a > b or a = b;
a < b == True  ==> b < a  == False;
a < b and b = c and c < d == True ==> a < d == True; );
```

Cuando una <partial type definition> incluye una <literal list> y la palabra clave **ordering**, las <literal signature>s se designan en orden ascendente, es decir:

```
literals A,B,C;
operators ordering;
```

implica A<B, B<C.

5.3.1.9 Sintipos

Un sintipo especifica un conjunto de valores de un género. Un sintipo utilizado como un género tiene la misma semántica que el género referenciado por el sintipo, salvo verificaciones de que unos valores pertenecen al conjunto de valores del género.

Gramática abstracta

Syntype-identifier	=	*Identifier*
Syntype-definition	::	*Syntype-name* *Parent-sort-identifier* *Range-condition*
Syntype-name	=	*Name*
Parent-sort-identifier	=	*Sort-identifier*

Gramática textual concreta

<syntype> ::=
<syntype identifier>

<syntype definition> ::=

syntype

<syntype name> = <parent sort identifier>

[<default initialization>] [**constants** <range condition>]

endsyntype [<syntype name>]

| **newtype** <syntype name> [<extended properties>]

<properties expression> **constants** <range condition>

endnewtype [<syntype name>]

<parent sort identifier> ::=

<sort>

Un <syntype> es una alternativa para un <sort>.

Una <syntype definition> con la palabra clave **syntype** y «= <syntype identifier>» es una sintaxis derivada definida más adelante.

Una <syntype definition> con la palabra clave **syntype** en la sintaxis concreta corresponde a *Syntype-definition* en la sintaxis abstracta.

Cuando un <syntype identifier> se utiliza como un <argument sort> en una <argument list> que define un operador, el género del argumento en una *argument list* es el *parent sort identifier* del sintipo.

Cuando un <syntype identifier> se utiliza como un resultado de un operador, el género del *result* es el *parent sort identifier* del sintipo.

Cuando un <syntype identifier> se utiliza como un calificador para un nombre, el *qualifier* es el *parent sort identifier* del sintipo.

Si se usa la palabra clave **syntype** y se omite la <range condition>, todos los valores del género están en la condición de intervalo.

Semántica

Una definición de sintipo define un sintipo que hace referencia a un identificador de género y una condición de intervalo. Especificar un especificador de sintipo es lo mismo que especificar el identificador de género de progenitor del sintipo, con excepción de los casos siguientes:

a) asignación a una variable declarada con un sintipo (véase 5.4.3);

b) salida de una señal si uno de los géneros especificados para la señal es un sintipo (véanse 2.5.4 y 2.7.4);

c) llamada a un procedimiento cuando uno de los géneros especificados para las variables de parámetro **in** del procedimiento es un sintipo (véanse 2.7.3 y 2.4.6);

d) creación de un proceso cuando uno de los géneros especificados para los parámetros del proceso es un sintipo (véanse 2.7.2 y 2.4.4);

e) entrada de una señal y una de las variables que está asociada a la entrada tiene un género que es un sintipo (véase 2.6.4);

f) utilización, en una expresión, de un operador que tiene un sintipo definido bien como un género de argumento o como un género de resultado (véanse 5.3.3.2 y 5.4.2.4);

g) sentencia o expresión activa de inicializar o reinicializar un temporizador y uno de los géneros en la definición de temporizador es un sintipo (véase 2.8);

h) definición de variable remota o de procedimiento remoto si uno de los géneros para la obtención de señales implícitas es un sintipo (véanse 4.13 y 4.14);

i) un parámetro de contexto formal de procedimiento con un parámetro **in/out** en <procedure signature> adaptado a un parámetro de contexto real en el cual el parámetro formal correspondiente o el parámetro **in/out** en <procedure signature> es un sintipo;

j) expresión con cualquier valor, en la cual el valor devuelto está dentro del intervalo (véase 5.4.4.6).

Por ejemplo, una <syntype definition> con la palabra clave **syntype** y «= <syntype identifier>» es equivalente a sustituir su <parent sort identifier> por el <parent sort identifier> de la <syntype definition> del <syntype identifier>. Es decir:

```
syntype s2 = n1 constants a1:a3 endsyntype s2;
syntype s3 = s2 constants a1:a2 endsyntype s3;
```

es equivalente a

```
syntype s2 = n1 constants a1:a3 endsyntype s2;
syntype s3 = n1 constants a1:a2 endsyntype s3;
```

Cuando un sintipo se especifica en términos de <syntype identifier>, los dos sintipos no pueden estar mutuamente definidos.

Un sintipo tiene un género que es el género identificado por el identificador de género de progenitor dado en la definición de sintipo.

Un sintipo tiene un intervalo que es el conjunto de valores especificados por las constantes de la definición de sintipo.

Modelo

Una <syntype definition> con la palabra clave **newtype** puede distinguirse de una <partial type definition> por la inclusión de **constants** <range condition>. Tal <syntype definition> es una notación taquigráfica para introducir una <partial type definition> con un nombre anónimo seguido de una <syntype definition> con la palabra clave **syntype** basada en este género anónimamente denominado. Esto es:

```
newtype X /* details */
 constants /* constant list */
endnewtype X;
```

es equivalente a

```
newtype anon /* details */
endnewtype anon;
```

seguido por

```
syntype X = anon
 constants /* constant list */
endsyntype X;
```

5.3.1.9.1 Condición de intervalo

Gramática abstracta

Range-condition	::	*Or-operator-identifier* *Condition-item*-**set**
Condition-item	=	*Open-range* | *Closed-range*
Open-range	::	*Operator-identifier* *Ground-expression*
Closed-range	::	*And-operator-identifier* *Open-range* *Open-range*
Or-operator-identifier	=	*Identifier*
And-operator-identifier	=	*Identifier*

Gramática textual concreta

<range condition> ::=
{ <closed range> | <open range> }
{ , { <closed range> | <open range> } }*

<closed range> ::=
<constant> : <constant>

<open range> ::=
<constant>
| { = | /= | < | > | <= | >= } <constant>

<constant> ::=
<ground expression>

El símbolo «<» («<=», «>», «>=» respectivamente) sólo debe utilizarse en la sintaxis concreta de la <range condition> si dicho símbolo ha sido definido con una <operator signature>

```
P, P -> package Predefined Boolean;
```

donde P es el género del sintipo. Estos símbolos representan *operator identifier.*

Un <closed range> sólo debe utilizarse si el símbolo «<=» está definido con una <operator signature>

```
P, P -> package Predefined Boolean;
```

donde P es el género del sintipo.

Una <constant> en una <range condition> tiene que tener el mismo género que el del sintipo.

Semántica

Una condición de intervalo define una verificación de intervalo. Una verificación de intervalo se utiliza cuando un sintipo tiene semántica adicional al género del sintipo (véanse 2.6.1.1 y 5.3.1.9 y los casos de sintipos con semántica diferente – véanse las secciones indicadas en los apartados a) a j) de 5.3.1.9, *Semántica*). Una verificación de intervalo se utiliza también para determinar la interpretación de una decisión (véase 2.7.5).

La verificación de intervalo es la aplicación del operador formado a partir de la condición de intervalo. Para las verificaciones de intervalo de sintipo, la aplicación de este operador tiene que ser equivalente a Verdadero; de no ser así, el comportamiento ulterior del sistema está indefinido. La verificación de intervalo se deriva como sigue:

a) Cada elemento (<open range> o <closed range>) en la <range condition> tiene un *open range* o *closed range* correspondiente en el *condition item*.

b) Un <open range> de la forma <constant> es equivalente a un <open range> de la forma = <constant>.

c) Para un término dado, A, entonces

   i) un <open range> de la forma = <constant>, /= <constant>,< <constant>, <= <constant>, > <constant>, y >= <constant>, tiene subtérminos en la verificación de intervalo de la forma A = <constant>, A /= <constant>, A < <constant>, A <= <constant>, A > <constant>, y A >= <constant> respectivamente.

   ii) un <closed range> de la forma <first constant> : <second constant> tiene un subtérmino en la verificación de intervalo de la forma <first constant> <= A **and** A <= <second constant> donde **and** corresponde al operador **and** booleano predefinido y corresponde al *And-operator-identifier* en la sintaxis abstracta.

d) Hay un *Or-operator-identifier* para el operador distribuido entre todos los elementos del *condition-item-***set** que es un booleano predefinido **or** de todos los elementos. La verificación de intervalo es el término formado a partir del booleano predefinido **or** de todos los subtérminos derivados de la <range condition>.

Si un sintipo se especifica sin una <range condition>, la verificación de intervalo es Verdadera.

5.3.1.10 Géneros estructura

Gramática textual concreta

<structure definition> ::=
struct <field list> [<end>] [**adding**]

<field list> ::=
<fields> { <end> <fields> }*

<fields> ::=
<field name> { , <field name> }* <field sort>

<field sort> ::=
<sort>

Cada <field name> de un género estructura tiene que ser diferente de cualquier otro <field name> de la misma <structure definition>.

Semántica

Una definición de estructura define un género estructura cuyos valores están compuestos a partir de una lista de valores de campo de géneros.

La longitud de la lista de valores está determinada por la definición de estructura, y el género de un valor está determinado por su posición en la lista de valores.

Modelo

Una definición de estructura es sintaxis derivada para la definición de:

a) un operador, Make! (¡hacer!), para crear valores de estructura; y

b) operadores para modificar valores de estructura y para extraer valores de campo a partir de valores de estructura.

El nombre del operador implicado para modificar un campo es el nombre de campo concatenado con «Modify!» (¡modificar!).

El nombre del operador implicado para extraer un campo es el nombre de campo concatenado con «Extract!» (¡extraer!).

La <argument list> para el operador Make! es la lista de <field sort>s que aparecen en la lista de campos, y en el orden de aparición de los mismos.

El <result> para el operador Make! es el identificador de género de la estructura.

La <argument list> para el operador de modificación de campo es el identificador de género de la estructura seguido por el <field sort> de ese campo. El <result> para un operador de modificación de campo es el identificador de género de la estructura.

La <argument list> para un operador de extracción de campo es el identificador de género de la estructura. El <result> para un operador de extracción de campo es el <field sort> de ese campo.

Hay una ecuación implícita para cada campo que define que modificar un campo de una estructura para darle un valor es lo mismo que construir un valor de estructura con ese valor para el campo.

Hay una ecuación implícita para cada campo que define que extraer un campo de un valor de estructura da el valor asociado con ese campo cuando la estructura fue construida.

A menos que se especifique **noequality** en <operator list>, hay una ecuación implícita que define la igualdad de valores de estructura mediante una igualdad por elementos.

Por ejemplo:

```
newtype s struct
   b  Boolean;
   i  Integer;
   c  Character;
endnewtype   s;
```

implica

```
newtype s
 operators
  Make! : Boolean, Integer, Character  -> s;
  bModify!   : s, Boolean              -> s;
  iModify!   : s, Integer              -> s;
  cModify!   : s, Character            -> s;
  bExtract! : s                        -> Boolean;
  iExtract! : s                        -> Integer;
  cExtract! : s                        -> Character;
 axioms
  bModify!   (Make!(x,y,z),b)     == Make!(b,y,z);
  iModify!   (Make!(x,y,z),i)     == Make!(x,i,z);
  cModify!   (Make!(x,y,z),c)     == Make!(x,y,c);
  bExtract! (Make!(x,y,z))        == x;
  iExtract! (Make!(x,y,z))        == y;
  cExtract! (Make!(x,y,z))        == z;
  Make!(x1,y1,z1) = Make!(x2,y2,z2) == (x1=x2) and (y1=y2) and (z1=z2);
endnewtype s;
```

5.3.1.11 Herencia

Gramática textual concreta

<inheritance rule> ::=
inherits <sort type expression>
[<literal renaming>]
[[**operators**] { **all** | (<inheritance list>) }
[<end>]] [**adding**]

<inheritance list> ::=
<inherited operator> { , <inherited operator> }*
[, **noequality**]

<inherited operator> ::=
[<operator name> =] <inherited operator name>

<inherited operator name> ::=
<base type operator name>

<literal renaming> ::=
literals <literal rename list> <end>

<literal rename list> ::=
<literal rename pair> { , <literal rename pair> }*

<literal rename pair> ::=
<literal rename signature> = <base type literal rename signature>

<literal rename signature> ::=
<literal operator name>
| <character string literal>

Si <sort type expression> contiene <actual context parameters>, la <inheritance rule> no puede contener <literal renaming> ni <inheritance list> y la palabra clave **all** está implícita si se omite. En ese caso, todos los literales,

operadores y axiomas del <base type> son heredados, incluidos los operadores iguales y no iguales y sus axiomas asociados (si los hay).

Todas las <literal rename signature>s en una <literal rename list> deben ser distintas. Todas las <base type literal rename signature>s en una <literal rename list> deben ser distintas.

Todos los <inherited operator name>s en una <inheritance list> tienen que ser distintos. Todos los <operator name>s en una <inheritance list> tienen que ser distintos.

Un <inherited operator name> especificado en una <inheritance list> tiene que ser un operador del <base type> definido en la <partial type definition> que define el <base type>.

Los nombres de operador «=» y «/=» están contenidos implícitamente en la <inheritance list> sin ninguna redenominación.

Cuando varios operadores del <base type> tienen el mismo nombre que el <inherited operator name>, todos esos operadores son heredados.

Modelo

Si la <sort type expression> contiene <actual context parameters> se utiliza el modelo de especialización de 6.3, y en los demás casos se utiliza el modelo siguiente. Cuando se aplica este último, no se heredan <operator definition>s, <operator diagram>s ni <default initialization>.

Un género (S) puede basarse en otro género (básico – BS) utilizando **newtype** en combinación con una regla de herencia. El género definido utilizando la regla de herencia es disjunto del tipo básico.

Si el tipo básico tiene literales definidos, los nombres de literal se heredan como nombres para literales del género que se define, a menos que se haya aplicado una redenominación a ese literal. Una redenominación de literal ha tenido lugar para un literal si el nombre de literal de tipo básico aparece como segundo nombre en un par de redenominación de literal, en cuyo caso el literal es redenominado con el primer nombre de ese par.

Hay un operador implícito de conversión de tipo que toma un argumento del género de tipo básico y tiene un resultado del nuevo género. El nombre de este operador es el nombre del género que se define concatenado con «!».

Hay nombres de operadores heredados como especificados por **all** o la lista de herencias. El nombre de un operador heredado es:

a) igual que el nombre del operador de tipo básico si se especifica **all** y el nombre está explícita o implícitamente definido como un nombre de operador en la definición parcial de tipo o definición de sintipo que define el tipo básico; de no ser así,

b) si el nombre de operador de tipo básico está dado en la lista de herencias y un nombre de operador seguido por «=» está dado para el operador heredado, entonces es redenominado con ese nombre; de no ser así,

c) si el nombre de operador de tipo básico está dado en la lista de herencias y un nombre de operador seguido por «=» no está dado para el operador heredado, entonces el nombre es el mismo que el del operador de tipo básico.

Si no se especifica **all** ni la lista de herencia, los operadores heredados son los operadores iguales y no iguales únicamente (véase 5.3.1.4).

Los géneros de argumento y el resultado de un operador heredado son los mismos que los del operador correspondiente del género de tipo básico, salvo que cada ocurrencia del tipo básico en los operadores heredados se cambia al nuevo género.

Para cada operador heredado (salvo para los operadores «=» y «/=», si **noequality** está especificado en <inheritance list>), hay una ecuación implícita:

$IO(t_1,...,t_n) == S!(BTO(v_1,..., v_n));$ si el resultado es el nuevo género

$IO(t_1,...,t_n) == BTO(v_1,...,v_n);$ en los demás casos

donde IO es el Operador Heredado (*inherited operator*), BTO es el Operador de Tipo Básico (*base type operator*) correspondiente, S! es el operador de conversión de tipo implícito. Los v_i son los identificadores de valor disjuntos. Si el género de v_i es el tipo básico, $t_i = S!(v_i)$, y en los demás casos $t_i = v_i$.

Para cada par de redenominación de literal $il_i = bl_i$ hay una ecuación implícita

il_i == S!(bl_i);

Hay una ecuación de literal implícita

```
for all ilv in S literals(
for all blv in BS literals(
Spelling(ilv) /= Spelling(il1),.., Spelling(ilv) /= Spelling(iln),
Spelling(ilv) == Spelling(blv) ==>      ilv == S!(blv);))
```

donde los il_i son los literales mencionados en los pares de redenominación de literal $il_i = bl_i$.

NOTA – Desde el punto de vista del usuario no hay diferencia entre las semánticas de los modelos ya que todas las propiedades de géneros con parámetros de contexto se describen localmente. No obstante, existen las diferencias siguientes entre géneros básicos con y sin parámetros de contexto:

	con parámetro de contexto	sin parámetro de contexto
Redenominación de operador/literal autorizada	no	sí
Definición de operador heredada	sí	no
Concordancia **atleast** (véase 6.2.9)	no	sí, pero solamente si no hay redenominación de operador/literal

Ejemplo

```
newtype bit
   inherits  Boolean
      literals 1 = True, 0 = False;
      operators  ("not", "and", "or")
   adding
   operators
      Exor: bit,bit -> bit;
   axioms
      Exor(a,b) == (a and (not b)) or ((not a) and b));
endnewtype bit;
```

5.3.1.12 Generadores

Un generador permite definir una plantilla de texto parametrizada que se expande por transformación antes de considerar la semántica de los tipos de datos.

5.3.1.12.1 Definición de generador

Gramática textual concreta

<generator definition> ::=
generator <generator name> (<generator parameter list>)
<generator text>
endgenerator [<generator name>]

<generator text> ::=
[<generator transformations>] <properties expression>

<generator parameter list> ::=
<generator parameter> { , <generator parameter> }*

<generator parameter> ::=
{ **type** | **literal** | **operator** | **constant** }
<generator formal name> {, <generator formal name> }*

<generator formal name> ::=
<generator formal name>

<generator sort> ::=

<generator formal name>

| <generator name>

Un <generator name> o <generator formal name> sólo se utilizará en una <properties expression> si la <properties expression> está en un <generator text>.

En una <generator definition> todos los <generator formal name>s de la misma clase (**type, literal, operator** o **constant**) tienen que ser distintos. Un nombre de la clase de **literal** tiene que ser distinto de cada uno de los nombres de la clase **constant** en la misma <generator definition>.

El <generator name> después de la palabra clave **generator** tiene que ser distinto de todos los nombres de género en la <generator definition> y también distinto de todos los <generator parameter>s **type** de esa <generator definition>.

Un <generator sort> sólo es válido si aparece como un <extended sort> en un <generator text> y el nombre es, bien el <generator name> de esa <generator definition>, bien un <generator formal name> definido por esa definición.

Si un <generator sort> es un <generator formal name>, tiene que estar definido de modo que sea de la clase **type**.

El <generator name> opcional después de **endgenerator** tiene que ser el mismo que el <generator name> dado después de **generator**.

Un <generator formal name> no debe utilizarse en un <qualifier>. Un <generator name> o <generator formal name> no debe:

a) ser calificado; ni

b) ir seguido de una <exclamation>; ni

c) ser utilizado en una <default initialization>.

Semántica

Un generador denomina una pieza de texto que puede utilizarse en transformaciones de generador.

Se considera que los textos de transformaciones de generador dentro de un texto de generador se expanden en el punto de definición del texto de generador.

Cada parámetro de generador tiene una clase (**type, literal, operator**, o **constant**) especificada por las palabras clave **type, literal, operator** o **constant** respectivamente.

Modelo

El texto definido por una definición de generador sólo se relaciona con la sintaxis abstracta si el generador es transformado. No hay sintaxis abstracta correspondiente para la definición de generador en el punto de definición.

Ejemplo

```
generator bag(type item)
literals empty;
operators
   put   : item, bag -> bag;
   count : item, bag -> Integer;
   take  : item, bag -> bag;
axioms
   take (i,put(i,b)) == b;
   take (i,empty)    == error!;
   count(i,empty)    == 0;
   count(i,put(j,b)) == count(i,b) + if i=j then 1 else 0 fi;
   put(i,put(j,b))   == put(j,put(i,b));
endgenerator bag;
```

5.3.1.12.2 Transformación de generador

Gramática textual concreta

<generator transformations> ::=
{ <generator transformation> [<end>] [**adding**] }+

<generator transformation> ::=
<generator identifier> (<generator actual list>)

<generator actual list> ::=
<generator actual> { , <generator actual> }*

<generator actual> ::=
<extended sort>
| <literal signature>
| <operator name>
| <ground term>

Si la clase de un <generator parameter> es **type**, el <generator actual> correspondiente tiene que ser un <extended sort>.

Si la clase de un <generator parameter> es **literal**, el <generator actual> correspondiente tiene que ser una <literal signature>.

Una <literal signature> que es un <name class literal> puede utilizarse como un <generator actual> únicamente si el correspondiente <generator formal name> no aparece en los <axioms>, o <literal mapping> de la <properties expression> en el <generator text>.

Si la clase de un <generator parameter> es **operator**, el correspondiente <generator actual> tiene que ser un <operator name>.

Si la clase de un <generator parameter> es **constant**, el correspondiente <generator actual> tiene que ser un <ground term>.

Si el <generator actual> es un <generator formal name>, la clase del <generator formal name> tiene que ser la misma que la clase del <generator actual>.

Semántica

El uso de una transformación de generador en propiedades ampliadas o en un texto de generador denota transformación del texto identificado por el identificador de generador. Un texto transformado para literales, operadores y axiomas se forma a partir del texto de generador con

a) los parámetros reales del generador sustituidos por los parámetros de generador efectivo; y

b) el nombre del generador sustituido

   i) si la transformación de generador está en una definición parcial de tipo o definición de sintipo, por la identidad del género que está siendo definida por la definición parcial de tipo o definición de sintipo; de no ser así,

   ii) en el caso de transformación de generador dentro de un generador, por el nombre de ese generador.

El texto transformado para literales es el texto transformado a partir de los literales en la expresión de propiedades del texto de generador, omitiendo la palabra clave **literals**.

El texto transformado para operadores es el texto transformado a partir de la lista de operadores en la expresión de propiedades del texto de generador, omitiendo la palabra clave **operators**.

El texto transformado para axiomas es el texto transformado a partir de los axiomas en la expresión de propiedades del texto de generador, omitiendo la palabra clave **axioms**.

Cuando hay más de una transformación de generador en la lista de transformaciones de generador, los textos transformados para literales (operadores y axiomas) se forman concatenando el texto transformado para los literales (operadores, axiomas, respectivamente) de todos los generadores en el orden en que aparecen en la lista.

El texto transformado para literales es una lista de literales para la expresión de propiedades de la definición parcial de tipo, definición de sintipo o definición de generador circundante que ocurre antes de cualquier lista de literales explícitamente mencionada en la expresión de propiedades. Es decir, si se ha especificado una ordenación, los literales definidos por transformaciones de generador aparecerán en el orden en que son instanciados y antes de cualesquiera otros literales.

Los textos transformados para operadores y axiomas se agregan a la lista de operadores y axiomas respectivamente de la definición parcial de tipo, definición de sintipo o definición de generador circundante.

Cuando se añade texto transformado a una expresión de propiedades, se considera que las palabras clave **literals, operators** y **axioms** se añaden, si es necesario, para crear una sintaxis concreta correcta.

Modelo

La sintaxis abstracta correspondiente a una transformación de generador se determina después de la transformación. La relación se determina a partir del texto transformado en el contexto en que aparece <generator transformation>.

Ejemplo

```
newtype boolbag bag(Boolean)
   adding
   operators
      yesvote : boolbag -> Boolean;
   axioms
      yesvote(b) == count(True,b) > count(False,b);
endnewtype boolbag;
```

5.3.1.13 Sinónimos

Un sinónimo da un nombre a una expresión fundamental que representa uno de los valores de un género.

Gramática textual concreta

<synonym definition> ::=
<internal synonym definition>
| <external synonym definition>

<internal synonym definition> ::=
synonym <synonym definition item>
{, <synonym definition item> }*

<synonym definition item> ::=
<synonym name> [<sort>] = <ground expression>

La <ground expression> en la sintaxis concreta denota un *ground term* en la sintaxis abstracta, como se define en 5.3.3.2.

Si se especifica un <sort> el resultado de la <ground expression> está vinculada a ese *sort*. La <ground expression> denota el *ground term* correspondiente en la sintaxis abstracta.

Si el género de la <ground expression> no puede determinarse de manera inequívoca, entonces el género debe especificarse en la <synonym definition>.

El género identificado por el <sort> tiene que ser uno de los géneros a los cuales puede estar vinculada la <ground expression>.

La <ground expression> no referirá al sinónimo definido por la <synonym definition>, directa ni indirectamente (por otro sinónimo).

Semántica

El valor que el sinónimo representa está determinado por el contexto en el cual aparece la definición de sinónimo.

Si el género de la expresión fundamental no puede determinarse de manera inequívoca en el contexto del sinónimo, entonces el género viene dado por el <sort>.

Un sinónimo tiene un valor que es el del término fundamental en la definición de sinónimo.

Un sinónimo tiene un género que es el del término fundamental en la definición de sinónimo.

5.3.1.14 Literales clase de nombre

Un literal clase de nombre es una notación taquigráfica para escribir un conjunto (que podría ser infinito) de nombres de literal definidos por una expresión regular.

Gramática textual concreta

<name class literal> ::=
nameclass <regular expression>

<regular expression> ::=
<partial regular expression>
{ [**or**] <partial regular expression> }*

<partial regular expression>::=
<regular element> [<Natural literal name> | + | *]

<regular element> ::=
(<regular expression>)
| <character string literal>
| <regular interval>

<regular interval> ::=
<character string literal> : <character string literal>

Los nombres formados por el <name class literal> tienen que satisfacer las condiciones estáticas normales para literales (véase 5.2.2) y, bien las reglas léxicas para nombres (véase 2.2.1), bien la sintaxis concreta para <character string literal>.

Los <character string literal>s en un <regular interval> tienen que ser ambos, al mismo tiempo, literales de longitud 1, y literales definidos por el género carácter (véase D.2).

Semántica

Un literal clase de nombre es una forma alternativa de especificar signaturas de literal.

Modelo

El conjunto de nombres equivalente de un literal clase de nombre se define como el conjunto de nombres que satisface la sintaxis especificada por la <regular expression>. El literal clase de nombre es equivalente a este conjunto de nombres en la sintaxis abstracta.

Una <regular expression> que es una lista de <partial regular expression>s sin un **or** especifica que los nombres pueden formarse a partir de los caracteres definidos por la primera <partial regular expression> seguidos de los caracteres definidos por la segunda <partial regular expression>.

Cuando se especifica un **or** entre dos <partial regular expression>s, los nombres se forman a partir de la primera o la segunda de estas <partial regular expression>s. Obsérvese que **or** vincula más estrechamente que una simple secuenciación, de modo que:

```
nameclass 'A' '0' or '1' '2';
```

es equivalente a

```
nameclass 'A' ('0' or '1') '2';
```

y define los literales A02, A12.

Si un <regular element> va seguido de <Natural literal name> la <partial regular expression> es equivalente al <regular element> repetido el número de veces especificado por el <Natural literal name>.

Por ejemplo

```
nameclass 'A' ('A' or 'B') 2
```

define nombres AAA, AAB, ABA y ABB.

Si un <regular element> va seguido de un '*' la <partial regular expression> es equivalente al <regular element> repetido cero o más veces.

Por ejemplo:

```
nameclass 'A' ('A' or 'B')*
```

define nombres A, AA, AB, AAA, AAB, ABA, ABB, AAAA, ... etc.

Si un <regular element> va seguido de '+' la <partial regular expression> es equivalente al <regular element> repetido una o más veces.

Por ejemplo:

```
nameclass 'A' ('A' or 'B') +
```

define nombres AA, AB, AAA, AAB, ABA, ABB, AAAA, ..., etc.

Un <regular element> que es una <regular expression> encerrada entre paréntesis define las secuencias de caracteres definidas por la <regular expression>.

Un <regular element> que es un <character string literal> define la secuencia de caracteres dada en el literal cadena de caracteres (omitiendo las comillas).

Un <regular element> que es un <regular interval> define todos los caracteres especificados por el <regular interval> como secuencias de caracteres alternativas. Los caracteres definidos por el <regular internal> son, todos ellos, mayores que o iguales al primer carácter y menores que o iguales al segundo carácter de acuerdo con la definición del género carácter (véase D.2). Por ejemplo:

```
'a':'f'
```

define las alternativas 'a' o 'b' o 'c' o 'd' o 'e' o 'f'.

Si la secuencia de definición de los nombres es importante (por ejemplo si se especifica **ordering**), se considera que los nombres están definidos en orden, de modo que están clasificados alfabéticamente de acuerdo con la ordenación del género carácter. Los caracteres se consideran como mayúsculas, y un prefijo verdadero de una palabra se considera menos que toda la palabra.

5.3.1.15 Correspondencia de literales

Las correspondencias de literales son notaciones taquigráficas utilizadas para definir la correspondencia de literales con valores.

Gramática textual concreta

<literal mapping> ::=
map <literal equation> { <end> <literal equation> }* [<end>]

<literal equation> ::=
<literal quantification>
(<literal axioms> { <end> <literal axioms> }* [<end>])

<literal axioms> ::=
<equation>
| <literal equation>

<literal quantification> ::=
for all <value name> { , <value name> }* **in** <extended sort> **literals**

<spelling term> ::=
spelling (<value identifier>)

Las reglas <literal mapping> y <spelling term> no forman parte del núcleo de datos pero ocurren en las reglas <properties expression> y <ground term>, respectivamente.

Semántica

La correspondencia de literales es una notación taquigráfica para definir un número grande (posiblemente infinito) de axiomas que pueden comprender todos los literales de un género. La correspondencia de literales permite establecer una relación de correspondencia entre los literales de un género y los valores del género.

El mecanismo de término de ortografía se utiliza en correspondencia de literales para referirse a la cadena de caracteres que contiene la ortografía del literal. Este mecanismo permite utilizar operadores cadena-de-caracteres (Charstring) para definir correspondencias de literales.

Modelo

Una <literal mapping> es una notación taquigráfica para un conjunto de <axioms>. Este conjunto de <axioms> se deriva de las <literal equation>s en la <literal mapping>. Las <equation>s que se utilizan para esta derivación son todas las <equation>s contenidas en <axioms> de las reglas <literal axioms>. En cada una de estas <equation>s, los <value identifier>s definidos por el <value name> en la <literal quantification> se reemplazan. En cada <equation> derivada, cada ocurrencia del mismo <value identifier> se reemplaza por el mismo <literal operator identifier> del <sort> de la <literal quantification>. El conjunto derivado de <axioms> contiene todas las posibles <equation>s que pueden derivarse de esta forma.

Los <axioms> derivados para <literal equation>s se agregan a los <axioms> (si existen) definidos después de la palabra clave **axioms** y antes de la palabra clave **map** en la misma <partial type definition>.

Por ejemplo:

```
newtype abc literals 'A',b,'c';
 operators
  "<" : abc,abc -> Boolean;
  "+" : abc,abc -> Boolean;
 map for all x,y in abc literals
  (x < y => y + x);
endnewtype abc;
```

es sintaxis concreta derivada para:

```
newtype abc literals 'A',b,'c';
 operators
  "<" : abc,abc -> Boolean;
  "+" : abc,abc -> Boolean;
 axioms
  'A'  < 'A' => 'A' + 'A';
  'A'  <  b  =>  b  + 'A';
  'A'  < 'c' => 'c' + 'A';
   b   < 'A' => 'A' +  b ;
   b   <  b  =>  b  +  b ;
   b   < 'c' => 'c' +  b ;
  'c'  < 'A' => 'A' + 'c';
  'c'  <  b  =>  b  + 'c';
  'c'  < 'c' => 'c' + 'c';
endnewtype abc;
```

Si una <literal quantification> contiene uno o más <spelling term>s, existe una sustitución de los <spelling term>s por literales cadena-de-caracteres (véase D.4).

Si la <literal signature> del <literal operator identifier> de un <spelling term> es un <literal operator name>, el <spelling term> es una notación taquigráfica de una cadena-de-caracteres en mayúsculas derivada del <literal operator identifier>. La cadena-de-caracteres contiene la ortografía en mayúsculas del <literal operator name> del <literal operator identifier>.

Si la <literal signature> del <literal operator identifier> de un <spelling term> es un <character string literal>, el spelling term> es una notación taquigráfica de una cadena-de-caracteres derivada del <character string literal>. La cadena-de-caracteres contiene la ortografía del <character string literal>.

La cadena de caracteres (Charstring) se utiliza para reemplazar el <value identifier> después de que la <literal equation> que contiene el <spelling term> es expandida como se ha indicado anteriormente.

Por ejemplo:

```
newtype abc literals 'A',Bb,'c';
 operators
  "<" : abc,abc -> Boolean;
 map for all x,y in abc literals
  spelling(x) < spelling(y) => x < y;
endnewtype abc;
```

es sintaxis concreta derivada para:

```
newtype abc literals 'A',Bb,'c';
 operators
  "<" : abc,abc -> Boolean;
 axioms
  /* obsérvese que 'A', Bb, 'c' están vinculados al género abc */
  /* '''A''', 'BB' and '''c''' deben ser calificados por el identificador de
  cadena de caracteres si estos literales son ambiguos -por razones de brevedad,
  esto se ha omitido en lo que sigue*/
'''A'''      < '''A'''    => 'A' < 'A';
'''A'''      < 'BB'       => 'A' <  Bb;
'''A'''      < '''c'''    => 'A' < 'c';
'BB'         < '''A'''    =>  Bb < 'A';
'BB'         < 'BB'       =>  Bb <  Bb;
'BB'         < '''c'''    =>  Bb < 'c';
'''c'''      < '''A'''    => 'c' < 'A';
'''c'''      < 'BB'       => 'c' <  Bb;
'''c'''      < '''c'''    => 'c' < 'c';
endnewtype abc;
```

Un <spelling term> debe estar en una <literal mapping>.

El <value identifier> en un <spelling term> tiene que ser un <value identifier> definido por una <literal quantification>.

5.3.2 Definiciones de operador

Las definiciones de operador permiten definir los operadores de una manera parecida a los procedimientos de retorno de valor. No obstante, los operadores no deben acceder al estado global, ni cambiarlo. Por lo tanto, sólo contienen una transición. La semántica de las definiciones de operador se expresa transformando la transición en una transición de arranque de un procedimiento.

Gramática textual concreta

<operator definitions> ::=
{ <operator definition> | <textual operator reference>}+

<operator definition> ::=

operator
{ <operator identifier> | <operator name>} <end>
<formal parameters> <end> <operator result> <end>
{ <data definition>
| <variable definition>
| <macro definition>
| <select definition> }*
<start> { <free action> }*
endoperator
[{<operator identifier> | <operator name> }] <end>

<operator result> ::=

returns [<variable name>] <extended sort>

<textual operator reference> ::=

operator <operator name>
[<formal parameters> <operator result>]
referenced <end>

Una <operator definition> o un <operator diagram> no debe utilizarse para definir los operadores de igualdad implícitos «=» y «/=».

El <start> de un <operator definition> no debe contener <virtuality>.

Para cada <operator definition> u <operator diagram>, debe existir una <operator signature> en la misma unidad de ámbito que tiene el mismo <operator name>, y que tienen posicionalmente los mismos <argument sort>s especificados en <formal parameters> y el mismo <result> que el especificado en <operator result>.

Para cada <operator signature> puede darse a lo sumo una <operator definition> o un <operator diagram> correspondiente.

Los <formal parameters> y <operator result> en <textual operator reference> pueden omitirse si dentro del mismo género no hay otra <textual operator reference> que tenga el mismo nombre. En este caso, los <formal parameters> y el <operator result> se derivan de la <operator signature>.

La <transition> o <transition area> no puede referirse a ningún <imperative operator>s ni a ningún <identifier>s definidos fuera de <operator definition> o el <operator diagram> circundantes, respectivamente, salvo para <synonym identifier>s, <operator identifier>s, <literal identifier>s y <sort>s.

Un operador definido por una <operator definition> o un <operator diagram> no debe aparecer en un axioma, generador ni <ground expression>. Una <operator definition> no debe aparecer en un generador.

Gramática gráfica concreta

<operator diagram> ::=

<frame symbol> ***contains***
{ <operator heading>
{ { <operator text area>
| <macro diagram> }*
<procedure start symbol> ***is followed by*** <transition area>
{<in-connector area> } * } ***set*** }

<operator heading> ::=

operator
{<operator identifier> | <operator name> }
<formal parameters>
<operator result>

<operator text area> ::=

<text symbol> ***contains***

{ <data definition>

| <variable definition>

| <select definition> }*

Como no hay gramática gráfica para definiciones de género, un <operator diagram> sólo puede utilizarse a través de una <textual operator reference>.

Semántica

Una definición de operador es una unidad de ámbito que define sus propios datos y variables que pueden ser manipulados dentro de la transición.

Las variables introducidas en <formal parameters> también son modificables.

Modelo

Una <operator definition> o un <operator diagram> se transforma en <procedure definition> o <procedure diagram>, respectivamente, como se indica en 7.

La aplicación en una expresión de un operador definido por una <operator definition> se transforma en una <value returning procedure call>, como se indica en 7.

5.3.3 Uso de datos

A continuación se define la manera de interpretar tipos de datos, géneros, literales, operadores y sinónimos en expresiones.

5.3.3.1 Expresiones

Expresiones son literales, operadores, accesos variables, expresiones condicionales y operadores imperativos.

Gramática abstracta

Expression = *Ground-expression* |
Active-expression

Una *expression* es una *active expression* si contiene un *active primary* (véase 5.4).

Una *expression* que no contiene un *active primary* es una *ground expression*.

Gramática textual concreta

Para simplificar la descripción no se distingue entre la sintaxis concreta de *ground expression* y *active expression*. La sintaxis concreta para <expression> se da en 5.3.3.2.

Semántica

Una expresión se interpreta como el valor de la expresión fundamental o expresión activa. Si el valor es **error**, el comportamiento ulterior del sistema es indefinido.

La expresión tiene el género de la expresión fundamental o expresión activa.

5.3.3.2 Expresiones fundamentales

Gramática abstracta

Ground-expression :: *Ground-term*

Las condiciones estáticas para el *ground term* son también aplicables a la *ground expression*.

Gramática textual concreta

```
<ground expression> ::=
                        <ground expression>

<expression> ::=
                        <sub expression>
                |       <value returning procedure call>

<sub expression> ::=
                        <operand0>
                |       <sub expression> => <operand0>

<operand0> ::=
                        <operand1>
                |       <operand0> { or | xor } <operand1>

<operand1> ::=
                        <operand2>
                |       <operand1> and <operand2>

<operand2> ::=
                        <operand3>
                |       <operand2> { = | /= | > | >= | < | <= | in } <operand3>

<operand3> ::=
                        <operand4>
                |       <operand3> { + | - | // } <operand4>

<operand4> ::=
                        <operand5>
                |       <operand4> { * | / | mod | rem } <operand5>

<operand5> ::=
                        [ - | not ] <primary>

<primary> ::=
                        <ground primary>
                |       <active primary>
                |       <extended primary>

<ground primary> ::=
                        <literal identifier>
                |       <operator identifier> ( <ground expression list> )
                |       ( <ground expression> )
                |       <conditional ground expression>

<extended primary> ::=
                        <synonym>
                |       <indexed primary>
                |       <field primary>
                |       <structure primary>
```

<ground expression list> ::=
<ground expression> { , <ground expression> }*

<operator identifier> ::=
<operator identifier>
| [<qualifier>] <quoted operator>

Una <expression> que no contiene ningún <active primary> representa una *ground expression* en la sintaxis abstracta. Una <ground expression> no debe contener una <active primary>.

Si una <expression> es un <ground primary> con un <operator identifier> y un <argument sort> de la <operator signature> es un <syntype>, la verificación de intervalo para ese sintipo, definida en 5.3.1.9.1, se aplica al correspondiente valor de argumento. El valor de la verificación de intervalo tiene que ser Verdadero.

Si una <expresión> es un <ground primary> con un <operator identifier> y el <result> de la <operator signature> es un <syntype>, la verificación de intervalo para ese sintipo, definida en 5.3.1.9.1, se aplica al valor de resultado. El valor de la verificación de intervalo tiene que ser Verdadero.

Si una <expression> contiene un <extended primary> (es decir, que es un <synonym>, <indexed primary>, <field primary> o <structure primary>), se reemplaza en el nivel de sintaxis concreta como se define en 5.3.3.3, 5.3.3.4, 5.3.3.5 y 5.3.3.6 respectivamente antes de considerar la relación con la sintaxis abstracta.

El <qualifier> opcional antes de un <quoted operator> tiene la misma relación con la sintaxis abstracta que un <qualifier> de un <operator identifier> (véase 5.2.2).

Semántica

Una expresión fundamental se interpreta como el valor denotado por el término fundamental sintácticamente equivalente a la expresión fundamental.

En general, no hay necesidad ni motivo para distinguir entre el término fundamental y el valor del término fundamental. Por ejemplo, el término fundamental para el valor de entero unidad puede escribirse «1». De ordinario hay varios términos fundamentales que denotan el mismo valor, por ejemplo los términos fundamentales enteros «0+1», «3–2» y «(7+5)/12», y es usual considerar que el valor es denotado por una forma simple del término fundamental (en este caso «1»).

El género de una expresión fundamental es el género del término fundamental equivalente.

El valor de una expresión fundamental es el valor del término fundamental equivalente.

5.3.3.3 Sinónimo

Gramática textual concreta

<synonym> ::=
<synonym identifier>
| <external synonym>

La alternativa <external synonym> se describe en 4.3.1.

Semántica

Un sinónimo es una notación taquigráfica para denotar una expresión definida en otro lugar.

Modelo

Un <synonym> representa la <ground expression> definida en la <synonym definition> identificada por el <synonym identifier>. Un <identifier> utilizado por la <ground expression> representa un *identifier* en la sintaxis abstracta de acuerdo con el contexto de la <synonym definition>.

5.3.3.4 Primario indizado

Un primario indizado es una notación sintáctica taquigráfica que puede utilizarse para denotar «indización» de un valor de «matriz». Sin embargo, aparte de la forma sintáctica especial, un primario indizado no tiene propiedades especiales y denota un operador que tiene como parámetro el primario.

Gramática textual concreta

<indexed primary> ::=
<primary> (<expression list>)

Semántica

Una expresión indizada representa la aplicación de un operador Extract! (¡extraer!).

Modelo

Un <primary> seguido de una <expression list> entre paréntesis es sintaxis concreta derivada para la sintaxis concreta

Extract!(<primary>, <expression list>)

y se considera entonces una expresión lícita aunque Extract! no esté autorizado como un nombre de operador en la sintaxis concreta para expresiones. La sintaxis abstracta se determina a partir de esta expresión concreta de acuerdo con 5.3.3.2.

5.3.3.5 Primario de campo

Un primario de campo es una notación sintáctica taquigráfica que puede utilizarse para denotar «selección de campo» de «estructuras». Sin embargo, aparte de la forma sintáctica especial, un primario de campo no tiene propiedades especiales y denota un operador que tiene como parámetro el primario.

Gramática textual concreta

<field primary> ::=
<primary> <field selection>

<field selection> ::=
! <field name>
| (<field name> { , <field name> }*)

El nombre de campo tiene que ser un nombre de campo definido para el género del primario.

Semántica

Un primario de campo representa la aplicación de uno de los operadores extraer campo de un género estructurado.

Modelo

La forma

<primary> (<field name>)

es sintaxis derivada para

<primary> ! (<field name>)

La forma

<primary> (<first field name> { , <field name> }*)

es sintaxis derivada para

<primary> ! <first field name> { ! <field name> }*

donde se preserva el orden de los nombres de campo.

La forma

<primary> ! <field name>

es sintaxis derivada para

<field extract operator name> (<primary>)

donde el nombre del operador extraer campo se ha formado por la concatenación del nombre de campo y «Extract!» en ese orden. Por ejemplo

s ! f1

es sintaxis derivada para

f1Extract!(s)

y se considera entonces que ésta es una expresión lícita aunque f1Extract! no esté autorizado como un nombre de operador en la sintaxis concreta para expresiones. La sintaxis abstracta se determina a partir de esta expresión concreta de acuerdo con 5.3.3.2.

En el caso en que hay un operador definido para un género de modo que

Extract!(s,name)

es un término válido cuando «name» es el mismo que un nombre de campo válido del género de s, entonces un primario

s(name)

es sintaxis concreta derivada para

Extract!(s,name)

y la selección de campo debe escribirse

s ! name

5.3.3.6 Primario de estructura

Gramática textual concreta

<structure primary> ::=
[<qualifier>] (. <expression list> .)

Semántica

Un primario de estructura representa un valor de un género estructurado que está construido a partir de expresiones para cada campo de la estructura.

La forma

(. <expression list> .)

es sintaxis concreta derivada para

Make!(<expression list>)

y se considera que ésta es una expresión fundamental lícita aunque Make! no esté autorizado como un nombre de operador en la sintaxis concreta para expresiones fundamentales. La sintaxis abstracta se determina a partir de esta expresión fundamental concreta de acuerdo con 5.3.3.1.

5.3.3.7 Expresión fundamental condicional

Gramática textual concreta

<conditional ground expression> ::=
if <Boolean ground expression>
then <consequence ground expression>
else <alternative ground expression>
fi

<consequence ground expression> ::=
<ground expression>

<alternative ground expression> ::=
<ground expression>

La <conditional ground expression> representa una *ground expression* en la sintaxis abstracta. Si la <Boolean ground expression> representa Verdadero, entonces la *ground expression* está representada por una <consequence ground expression>; de no ser así, está representada por la <alternative ground expression>.

El género de la <consequence ground expression> tiene que ser el mismo que el género de la <alternative ground expression>.

Semántica

Una expresión fundamental condicional es un primario fundamental que se interpreta, bien como la expresión fundamental de consecuencia, bien como la expresión fundamental de alternativa.

Si la <Boolean ground expression> tiene el valor Verdadero, la <alternative ground expression> no se interpreta. Si la <Boolean ground expression> tiene el valor Falso, la <consequence ground expression> no se interpreta. Si la <ground expression> que se interpreta tiene el valor de un error, el comportamiento ulterior del sistema está indefinido.

El género de una expresión fundamental condicional es el género de la expresión fundamental de consecuencia (y también el género de la expresión fundamental de alternativa).

5.4 Uso de datos con variables

Esta subcláusula define el uso de datos y variables declaradas en procesos y procedimientos, así como los operadores imperativos que obtienen valores a partir del sistema subyacente.

Una variable tiene un género y un valor asociado de ese género. El valor asociado a una variable puede cambiarse asignando un nuevo valor a la variable. El valor asociado a la variable puede utilizarse en una expresión accediendo a la variable.

Una expresión que contiene una variable se considera «activa» pues el valor obtenido al interpretar la expresión puede variar de conformidad con el último valor asignado a la variable.

5.4.1 Definiciones de variables y de datos

Gramática textual concreta

<data definition> ::=
{ <partial type definition>
| <syntype definition>
| <generator definition>
| <synonym definition> } <end>

Una definición de datos forma parte de una *data type definition* si es una <partial type definition> o <syntype definition>.

La sintaxis para introducir variables de proceso y para variables de parámetros de procedimiento se indica en 2.5.1.1 y 2.3.4 respectivamente. Una variable definida en un procedimiento no será revelada.

Semántica

Una definición de datos se utiliza sea para la definición de una parte de un tipo de datos, sea para la definición de un sinónimo para una expresión, tal como se define más detalladamente en 5.2.1, 5.3.1.9 ó 5.3.1.13.

5.4.2 Acceso a variables

A continuación se define la manera de interpretar una expresión que comprende variables.

5.4.2.1 Expresiones activas

Gramática abstracta

Active-expression = *Variable-access* |
Conditional-expression |
Operator-application |
Imperative-operator |
Error-term

Gramática textual completa

<active expression> ::=
<active expression>

<active primary> ::=
<variable access>
| <operator application>
| <conditional expression>
| <imperative operator>
| (<active expression>)
| <active extended primary>
| **error**

<active extended primary> ::=
<active extended primary>

<expression list> ::=
<expression> { , <expression> }*

Una <expression> es una <active expression> si contiene un <active primary>.

Un <extended primary> es un <active extended primary> si contiene un <active primary>. Para un <extended primary> se produce una sustitución en el nivel de sintaxis concreta como se define en 5.3.3.3, 5.3.3.4, 5.3.3.5 y 5.3.3.6 antes de considerar la relación con la sintaxis abstracta.

Semántica

Una expresión activa es una expresión cuyo valor dependerá del estado en que se encuentre el sistema.

El género de una expresión activa es el género del término fundamental equivalente.

El valor de una expresión activa es el término fundamental equivalente a la expresión activa en el momento de la interpretación.

Si una expresión que contiene **error** es interpretada, el comportamiento ulterior del sistema es indefinido.

En una expresión activa cada operador es interpretado en el orden determinado atravesando la sintaxis concreta indicada en 5.3.3.2 de izquierda a derecha. En una lista de expresiones activa o una lista de expresiones, cada elemento de la lista es interpretado de izquierda a derecha.

Modelo

Cada vez que se interpreta la expresión activa, su valor se determina hallando el término fundamental equivalente a la expresión activa. Este término fundamental se determina a partir de una expresión fundamental formada reemplazando cada primario activo en la expresión activa por el término fundamental equivalente al valor de dicho primario activo. El valor de una expresión activa es el mismo que el de la expresión fundamental.

5.4.2.2 Acceso a variable

Gramática abstracta

Variable-access = *Variable-identifier*

Gramática textual completa

<variable access> ::=
<variable identifier>

Semántica

La interpretación de un acceso a variable da el valor asociado a la variable identificada.

Un acceso a variable tiene el género de la variable que él identifica.

El valor de un acceso a variable es el último valor asociado a la variable, o **error,** si la variable es «indefinida». Si el valor es **error**, el comportamiento ulterior del sistema es indefinido.

5.4.2.3 Expresión condicional

Una expresión condicional es una expresión que se interpreta bien como la consecuencia, bien como la alternativa.

Gramática abstracta

Conditional-expression :: *Boolean-expression*
Consequence-expression
Alternative-expression

Boolean-expression = *Expression*

Consequence-expression = *Expression*

Alternative-expression = *Expression*

El género de la *consequence expression* tiene que ser igual al de la *alternative expression*.

Gramática textual concreta

<conditional expression> ::=
if <Boolean active expression>
then <consequence expression>
else <alternative expression>
fi
| **if** <Boolean expression>
then <active consequence expression>
else <alternative expression>
fi
| **if** <Boolean expression>
then <consequence expression>
else <active alternative expression>
fi

<consequence expression> ::=
<expression>

<alternative expression> ::=
<expression>

Una <conditional expression> se distingue de una <conditional ground expression> por la ocurrencia de una <active expression> en la <conditional expression>.

Semántica

Una expresión condicional se interpreta como la interpretación de la condición seguida, bien por la interpretación de la expresión de consecuencia, bien por la interpretación de la expresión de alternativa. La consecuencia sólo se interpreta si la condición tiene el valor Verdadero, de modo que, si la condición tiene el valor Falso, el comportamiento ulterior del sistema está indefinido solamente si la expresión de alternativa es un error. De manera similar, la alternativa sólo se interpreta si la condición tiene el valor Falso, de modo que si la condición tiene el valor Verdadero el comportamiento del sistema está indefinido solamente si la expresión de consecuencia es un error.

El género de la expresión condicional es igual al género de la consecuencia y de la alternativa. El valor de la expresión condicional es el valor de la consecuencia si la condición es Verdadero, o el valor de la alternativa si la condición es Falso.

5.4.2.4 Aplicación de operador

Una aplicación de operador es la aplicación de un operador, siendo uno o más de los argumentos efectivos una expresión activa.

Gramática abstracta

Operator-application :: *Operator-identifier*
Expression^{+}

Si un *sort* argumento de la *operator signature* es un *syntype* y la *expression* correspondiente en la lista de *expressions* es una *ground expression*, la verificación de intervalo definida en 5.3.1.9.1 aplicada al valor de la *expression* tiene que ser Verdadero.

Gramática textual concreta

<operator application> ::=
<operator identifier> (<active expression list>)

<active expression list> ::=

<active expression> [, <expression list>]

| <ground expression> , <active expression list>

Una <operator application> se distingue de la <ground expression> sintácticamente similar porque una de las <expression>s en la lista de <expression>s entre paréntesis es una <active expression>. Si todas las <expression>s entre paréntesis son <ground expression>s, la construcción representa una *ground expression*, definida en 5.3.3.2.

Semántica

Una aplicación de operador es una expresión activa que tiene el valor del término fundamental equivalente a la aplicación de operador. El término fundamental equivalente se determina como se indica en 5.4.2.1.

La lista de expresiones para la aplicación de operador se interpreta en el orden dado, antes de interpretar el operador.

Si un género argumento de la signatura de operador es un sintipo y la expresión correspondiente en la lista de expresiones activas es una expresión activa, la verificación de intervalo definida en 5.3.1.9.1 se aplica al valor de la expresión. Si la verificación de intervalo es Falso en el momento de la interpretación, el sistema está en error y su comportamiento ulterior está indefinido.

Si el género resultado de la signatura de operador es un sintipo, la verificación de intervalo definida en 5.3.1.9.1 se aplica al valor de la aplicación de operador. Si la verificación de intervalo es Falso en el momento de la interpretación, el sistema está en error y su comportamiento ulterior es indefinido.

5.4.3 Enunciado de asignación

Gramática abstracta

Assignment-statement :: *Variable-identifier*
Expression

El género del *variable identifier* y el género de la *expression* deben ser iguales.

Si la *variable* está declarada con un *syntype* y la *expression* es una *ground expression*, la verificación de intervalo definida en 5.3.1.9.1 aplicada a la expresión tiene que ser Verdadero.

Gramática textual concreta

<assignment statement> ::=

<variable> := <expression>

<variable> ::=

<variable identifier>

| <indexed variable>

| <field variable>

Si la <variable> es un <variable identifier> la <expression> en la sintaxis concreta representa la <expression> en la sintaxis abstracta. Las otras formas de <variable>, <indexed variable> y <field variable> son sintaxis derivadas, y la <expression> en la sintaxis abstracta se halla en la sintaxis concreta equivalente definida en 5.4.3.1 y 5.4.3.2.

Semántica

La interpretación de un enunciado de asignación crea una asociación de la variable identificada en la sentencia de asignación con el valor de la expresión en la sentencia de asignación. La asociación anterior de la variable desaparece.

Si la variable está declarada con un sintipo y la expresión es una expresión activa, la verificación de intervalo definida en 5.3.1.9.1 se aplica a la expresión. Si esta verificación de intervalo es equivalente a Falso, la asignación es errónea y el comportamiento ulterior del sistema está indefinido.

5.4.3.1 Variable indizada

Una variable indizada es una notación sintáctica taquigráfica que puede utilizarse para denotar «indización» de «matrices». Sin embargo, aparte de la forma sintáctica especial, un primario activo indizado no tiene propiedades especiales y denota un operador con el primario activo como parámetro.

Gramática textual concreta

<indexed variable> ::=
<variable> (<expression list>)

Debe haber una definición apropiada de un operador denominado Modify! (¡Modificar!).

Semántica

Una variable indizada representa la asignación de un valor formado por la aplicación del operador Modify! a un acceso de la variable y la expresión dada en la variable indizada.

Modelo

La forma de sintaxis concreta

<variable> (<expression list>) := <expression>

es sintaxis concreta derivada para

<variable> := Modify!(<variable>,<expression list>,<expression>)

donde se repite la misma <variable> y se considera que el texto es una asignación lícita aunque Modify! no esté autorizado como nombre de operador en la sintaxis concreta para expresiones. La sintaxis abstracta se determina para esta <assignment statement> de acuerdo con 5.4.3.

NOTA – Una consecuencia de este modelo es que un valor debe ser asignado a una matriz completa antes de que pueda modificarse ningún elemento.

5.4.3.2 Variable de campo

Una variable de campo es una notación taquigráfica para asignar un valor a una variable de modo que sólo cambie el valor de un campo de esa variable.

Gramática textual concreta

<field variable> ::=
<variable> <field selection>

Debe haber una definición apropiada de un operador denominado Modify!. Normalmente esta definición estará implicada por una definición de género estructurada.

Semántica

Una variable de campo representa la asignación de un valor formado por la aplicación de un operador de modificar campo.

Modelo

Una selección de campo entre paréntesis es sintaxis derivada para ! <field name> en la selección de campo definida en 5.3.3.5.

La forma de sintaxis concreta

<variable> ! <field name> := <expression>

es sintaxis concreta derivada para

<variable> := <field modify operator name> (<variable>, <expression>)

donde

a) se repite la misma <variable>; y

b) el <field modify operator name> se forma a partir de la concatenación del nombre de campo y «Modify!», después de lo cual

c) se considera que el texto es una asignación lícita aunque el <field modify operator name> no esté autorizado como nombre de operador en la sintaxis concreta para expresiones.

Si hay más de un <field name> en la selección de campo, se les modela como se ha indicado antes expandiendo cada ! <field name>, uno por uno, de derecha a izquierda y considerando la parte restante de la <field variable> como una <variable>. Por ejemplo

var ! fielda ! fieldb := expression;

es modelado primeramente por

var ! fielda := fieldbModify!(var ! fielda, expression);

y después por

var := fieldbModify!(var, fieldbModify!(var ! fielda, expression));

La sintaxis abstracta se determina para la <assignment statement> formada mediante el modelado conforme 5.4.3.

NOTA – Una consecuencia de este modelo es que un valor debe ser asignado a una estructura completa antes de que pueda modificarse ningún campo.

5.4.3.3 Inicialización por defecto

Una inicialización por defecto permite inicializar todas las variables de un género especificado con el mismo valor, cuando las variables son creadas.

Gramática textual concreta

<default initialization> ::=
default <ground expression> [<end>]

Una <partial type definition> o <syntype definition> no debe contener más de una <default initialization>.

Semántica

Una inicialización por defecto se añade facultativamente a una expresión de propiedades de un género. Una inicialización por defecto especifica que cualquier variable declarada con el género introducido por la definición parcial de tipo o definición de sintipo recibe inicialmente el valor de la expresión fundamental asociada.

Modelo

La inicialización por defecto es una notación taquigráfica para especificar una inicialización explícita para las variables del <sort>, que se declaran sin una <ground expression>.

Si no se da ninguna <default initialization> en una <syntype definition>, el sintipo tiene la <default initialization> del <parent sort identifier> siempre y cuando su valor esté en el intervalo.

Una variable declarada dentro de un tipo parametrizado cuyo género es un parámetro de contexto formal no obtiene la inicialización por defecto de su género.

5.4.4 Operadores imperativos

Los operadores imperativos obtienen valores del estado del sistema subyacente.

Las transformaciones descritas en los *Modelos* de esta subcláusula se efectúan al mismo tiempo que la expansión para importación. Una etiqueta asociada a una acción en la cual aparece un operador imperativo se desplaza a la primera tarea insertada durante la transformación descrita. Si aparecen varios operadores imperativos en una expresión, las tareas se insertan en el mismo orden en que aparecen los operadores imperativos en la expresión.

Gramática abstracta

Imperative-operator = *Now-expression* |
Pid-expression |
View-expression |
Timer-active-expression |
Anyvalue-expression

Gramática textual concreta

<imperative operator> ::=
<now expression>
| <import expression>
| <PId expression>
| <view expression>
| <timer active expression>
| <any value expression>

Los operadores imperativos son expresiones para acceder al reloj de sistema, al valor de variables importadas, a los valores PId asociados con un proceso, al valor de variables observadas, al estado de temporizadores o para suministrar valores no especificados.

5.4.4.1 Expresión now

Gramática abstracta

Now-expression :: ( )

Gramática textual concreta

<now expression> ::=
now

Semántica

La expresión now (ahora) es una expresión que accede a la variable de reloj de sistema para determinar el tiempo absoluto de sistema.

La expresión now representa una expresión que solicita el valor vigente del reloj de sistema que da la hora. El origen y la unidad de tiempo dependen del sistema. El hecho de que dos ocurrencias de **now** en la misma transición den el mismo valor depende del sistema. No obstante, siempre es cierto que:

```
now <= now;
```

Una expresión now tiene el género Tiempo.

Modelo

La utilización de <now expression> en una expresión es una notación taquigráfica para insertar una tarea exactamente antes de la acción en la cual aparece la expresión que asigna a una variable implícita el valor de <now expression> y

utiliza después esa variable implícita en la expresión. Si <now expression> aparece varias veces en una expresión, se utiliza una variable para cada ocurrencia.

5.4.4.2 Expresión import

Gramática textual concreta

La sintaxis concreta para una expresión import (importación) se define en 4.13.

Semántica

Además de la semántica definida en 4.13, una expresión importación se interpreta como un acceso a variable (véase 5.4.2.2) a la variable implícita para la expresión importación.

Modelo

La expresión importación tiene una sintaxis implícita para la importación del valor, definida en 4.13, y también un *variable access* implícito de la variable implícita para la importación en el contexto en que aparece <import expression>.

La utilización de <import expression> en una expresión es una notación taquigráfica para insertar una tarea exactamente antes de la acción en la cual aparece la expresión que asigna a una variable implícita el valor de <import expression> y utiliza después esa variable implícita en la expresión. Si <import expression> aparece varias veces en una expresión, se utiliza una variable para cada ocurrencia.

5.4.4.3 Expresión PId

Gramática abstracta

Pid-expression	=	*Self-expression* | *Parent-expression* | *Offspring-expression* | *Sender-expression*
Self-expression	::	( )
Parent-expression	::	( )
Offspring-expression	::	( )
Sender-expression	::	( )

Gramática textual concreta

<PId expression> ::=
self
| **parent**
| **offspring**
| **sender**

Semántica

Una expresión PId accede a una de las variables implícitas de proceso definidas en 2.4.4. La expresión de variable de proceso se interpreta como el último valor asociado a la variable implícita correspondiente.

Una expresión PId tiene un género, que es PId.

Una expresión PId tiene un valor, que es el último valor asociado a la variable correspondiente, como se define en 2.4.4.

5.4.4.4 Expresión view

Una expresión view (visión) permite a un proceso obtener el valor de una variable de otro proceso en el mismo bloque, como si la variable estuviese definida localmente. El proceso que observa no puede modificar el valor asociado a la variable.

Gramática abstracta

View-expression :: *View-identifier*
[*Expression*]

La *expression* tiene que ser una expresión PId.

Gramática textual concreta

<view expression> ::=

view (<view identifier> [, <PId expression>])

Semántica

Una expresión visión se interpreta de la misma manera que un acceso a variable (véase 5.4.2.2).

El valor de una expresión visión es el acceso a variable y un género es el género de *View-definition*.

Si se da una *Expression*, la variable a la que se accede es la variable en la instancia de proceso dentro del mismo bloque identificado por *Expression*. Si *Expression* denota una instancia no existente o si el proceso denotado por *Expression* no contiene una variable del mismo nombre y género, no puede accederse a variable.

Si no se da *Expression*, la variable a la que se accede es la variable en una instancia de proceso arbitraria dentro del bloque, que contiene una variable revelada con el mismo nombre y género. Si no existe esa instancia, no puede accederse a variable.

Si no puede accederse a variable sobre la base de <view expression> el comportamiento ulterior del sistema es indefinido.

Modelo

La utilización de <view expression> en una expresión es una notación taquigráfica para insertar una tarea exactamente antes de la acción en la cual aparece la expresión que asigna a una variable implícita el valor de <view expression> y utiliza después esa variable implícita en la expresión. Si <view expression> aparece varias veces en una expresión, se utiliza una variable para cada ocurrencia.

5.4.4.5 Expresión Timer active

Gramática abstracta

Timer-active-expression :: *Timer-identifier*
*Expression**

Los géneros de la *Expression** en *Timer-active-expression* (Expresión temporizador activo) tienen que corresponder en posición con el *Sort-reference-identifier** que sigue inmediatamente al *Timer-name* (2.8) identificado por el *Timer-identifier*.

Gramática textual concreta

<timer active expression> ::=

active (<timer identifier> [(<expression list>)])

Semántica

Una expresión temporizador activo es una expresión del género booleano predefinido que tiene el valor Verdadero si el temporizador identificado por el identificador de temporizador, e inicializado con los mismos valores que los denotados por la lista de expresiones (si existe), está activo (véase 2.8). De no ser así, la expresión temporizador activo tiene el valor Falso. Las expresiones se interpretan en el orden dado.

Si un género especificado en una definición de temporizador es un sintipo, la comprobación de intervalo definida en 5.3.19.1 aplicada a la expresión correspondiente en <expression list> debe ser Verdadera, o en caso contrario el sistema está en error y su comportamiento ulterior es indefinido.

Modelo

La utilización de <timer active expression> en una expresión es una notación taquigráfica para insertar una tarea exactamente antes de la acción en la cual aparece la expresión que asigna a una variable implícita el valor de <timer active expression> y utiliza después esa variable implícita en la expresión. Si <timer active expression> aparece varias veces en una expresión, se utiliza una variable para cada ocurrencia.

5.4.4.6 Expresión anyvalue

Gramática abstracta

Anyvalue-expression :: *Sort-reference-identifier*

Gramática textual concreta

<anyvalue expression> ::=

any(<sort>)

Semántica

Una *Anyvalue-expression* (cualquier valor) es un valor no especificado del género o sintipo designado por *Sort-reference-identifier*. Si no existe valor, el comportamiento ulterior del sistema es indefinido. Si *Sort-reference-identifier* denota un *Syntype-identifier*, el valor resultante estará dentro del intervalo de ese sintipo. *Anyvalue-expression* es útil para modelar el comportamiento en los casos en que indicar un valor específico implicaría una sobreespecificación. A partir de un valor devuelto por una *Anyvalue-expression* no pueden suponerse otros valores devueltos por *Anyvalue-expression*.

Modelo

La utilización de <anyvalue expression> en una expresión es una notación taquigráfica para insertar una tarea exactamente antes de la acción en la cual aparece la expresión que asigna a una variable implícita el valor de <anyvalue expression> y utiliza después esa variable implícita en la expresión. Si <anyvalue expression> aparece varias veces en una expresión, se utiliza una variable para cada ocurrencia.

5.4.5 Llamada a procedimiento de retorno de valor

Gramática textual concreta

<value returning procedure call> ::=

<procedure call>

| <remote procedure call>

Una <value returning procedure call> no debe aparecer en la <Boolean expression> de una <continuous signal area>, <continuous signal>, <enabling condition area> o <enabling condition>.

El <procedure identifier> en una <value returning procedure call> debe identificar un procedimiento (o procedimiento remoto) que tiene por lo menos un parámetro en sus <procedure formal parameters> y el parámetro formal de finalización debe tener el atributo **in**/**out**. Esta regla se aplica después de la transformación de <procedure result>.

NOTA – Normalmente el <procedure identifier> identifica un procedimiento con un <procedure result>.

Modelo

La utilización de <value returning procedure call> en una expresión es una notación taquigráfica para insertar una acción de llamada a procedimiento, exactamente antes de la acción en la cual aparece la expresión, que contiene el valor <value returning procedure call> en el cual se ha añadido una <expression> suplementaria a los <actual parameters>, y utiliza después esa <expression> en lugar de <value returning procedure call> en la acción siguiente.

La <expression> construida consiste en el identificador de una nueva variable implícita distinta, cuyo <sort> es el <sort> del parámetro formal de finalización para el procedimiento.

La transformación se efectúa cuando se suprimen otros operadores imperativos de las expresiones (véase 5.4.4).

NOTA – La transformación de la llamada a procedimiento de retorno de valor después de <procedure result> implica que un procedimiento con <procedure result> puede utilizarse en <procedure call> y que un procedimiento sin <procedure result> puede utilizarse en <value returning procedure call> si cumple las condiciones estipuladas en esta subcláusula.

5.4.6 Datos externos

Gramática textual concreta

<external properties> ::=

alternative

<external formalism name> [, <word>] <end>

<external data description>

[**endalternative**] [<end>]

<external formalism name> ::=

<text>

<external data description> ::=

<text>

<external formalism name> no debe contener el carácter «;» ni «,». Si <word> está presente, denota la secuencia que termina la <external data description>. Si <external data description> no contiene la palabra clave **endalternative**, <word> puede omitirse. En los demás casos, <word> termina la definición de datos alternativos y puede omitirse **endalternative**. La <word> de terminación no forma parte formalmente de la descripción SDL.

Semántica

alternative indica que <external data description> no forma parte formalmente de la descripción SDL. La utilización de <external formalism name> permite relacionar <external data description> con algún formalismo externo. Las relaciones con formalismos externos no forman parte de la presente Recomendación.

Los literales y operadores utilizados fuera de <partial type definition> sólo son los que están definidos explícitamente en <operators>.

Si en otra Recomendación se define la correspondencia de un formalismo denotado por <external formalism name> con valores basados en el paquete Predefined (véase el Anexo D), la correspondencia puede implicar literales y operadores implícitos. En este caso, éstos se consideran además de los indicados en <operators> de <partial type definition>.

Se supone conceptualmente que la semántica de una <partial type definition> con <external properties> se da en un conjunto de axiomas que no pueden figurar en la descripción SDL.

Ejemplo

```
newtype application_data

     literals empty;

     operators

        anyuseExtract!  : application_data        -> Boolean;

        idExtract!      : application_data        -> Integer;

        anyuseModify!   : Boolean, application_data   -> application_data;

        idModify!       : Integer, application_data   -> application_data;

        Make!           : Boolean, Integer        -> application_data;

     alternative ASN.1;

        SET {   anyuse BOOLEAN

                id   INTEGER }

     endalternative;

endnewtype application_data;
```

6 Conceptos de tipificación estructural en SDL

En esta cláusula se presentan varios mecanismos de lenguaje caracterizados por el modelado de fenómenos específicos de aplicaciones por instancias y conceptos específicos de aplicaciones por tipos. Ello implica que el mecanismo de herencia está destinado a representar una generalización/especialización de conceptos.

Los mecanismos de lenguaje presentados dan:

a) definiciones de tipos (puras) que se pueden definir en cualquier lugar de un sistema o un paquete;

b) definiciones de instancias basadas en tipos que definen instancias o conjuntos de instancias de acuerdo con tipos;

c) definiciones de tipos parametrizadas que son independientes del ámbito circundante mediante parámetros de contexto y que puedan estar vinculadas a ámbitos específicos;

d) especialización de definiciones de supertipos en definiciones de subtipos, añadiendo propiedades y redefiniendo tipos y transiciones virtuales.

Los conceptos presentados en esta cláusula son conceptos adicionales. Las propiedades de una notación taquigráfica se derivan de la manera en que se modela en función de los conceptos primitivos (o se transforma en ellos). Para garantizar una utilización fácil e inequívoca de las notaciones taquigráficas, y reducir sus repercusiones cuando se combinan varias notaciones taquigráficas, estos conceptos se transforman en un orden especificado que se define en 7.

6.1 Tipos, instancias y puertas

Existe una distinción entre la definición de instancias (o conjuntos de instancias) y la definición de tipos en las descripciones SDL. En esta subcláusula se presentan (en 6.1.1) definiciones de tipos para sistemas, bloques, procesos y servicios, y (en 6.1.3) las especificaciones de instancias correspondientes, mientras que en 2 y 5 se presentan señales, procedimientos, temporizadores y géneros como tipos. Una definición de tipo no está conectada (mediante canales o rutas de señales) a ninguna instancia; en cambio, las definiciones de tipo introducen puertas (6.1.4). Estos son puntos de conexión en las instancias basadas en tipos para canales y rutas de señales.

6.1.1 Definiciones de tipo

6.1.1.1 Tipo de sistema

Gramática textual concreta

<system type definition> ::=
system type {<system type name> | <system type identifier>}
[<formal context parameters>]
[<specialization>] <end>
{<entity in system>}*
endsystem type [<system type name> | <system type identifier>] <end>

<textual system type reference>::=
system type <system type name> **referenced** <end>

Un <formal context parameter> de <formal context parameters> no debe ser un <process context parameter>, <variable context parameter> ni <timer context parameter>.

Gramática gráfica concreta

<system type diagram> ::=
is associated with
<frame symbol> ***contains***
{<system type heading>
{ {<system text area>}*
{<macro diagram>}*
<block interaction area>
{<type in system area>}*} ***set*** }

<system type heading> ::=

system type {<system type name> | <system type identifier>}
[<formal context parameters>]
[<specialization>]

<type in system area> ::=

<block type diagram>
| <block type reference>
| <process type diagram>
| <process type reference>
| <service type diagram>
| <service type reference>
| <procedure diagram>
| <graphical procedure reference>

<system type reference>::=

<system type symbol> ***contains*** { **system** <system type name> }

<system type symbol>::=

<block type symbol>

Semántica

Una <system type definition> define un tipo de sistema. Todos los sistemas de un tipo de sistema tienen las mismas propiedades definidas para ese tipo de sistema.

6.1.1.2 Tipo de bloque

Gramática textual concreta

<block type definition> ::=

[<virtuality>]
block type {<block type name> | <block type identifier>}
[<formal context parameters>]
[<virtuality constraint>]
[<specialization>] <end>
{<gate definition>}*
{<entity in block>}*
[<block substructure definition>
| <textual block substructure reference>]
endblock type [<block type name> | <block type identifier>] <end>

<textual block type reference>::=

[<virtuality>] **block type** <block type name>
referenced <end>

Un <formal context parameter> de <formal context parameters> no debe ser un <process context parameter>, <variable context parameter> ni <timer context parameter>.

Gramática gráfica concreta

<block type diagram> ::=
<frame symbol>
contains {<block type heading>
{ {<block text area>}* {<macro diagram>}*
{<type in block area>}*
[<process interaction area>]
[<block substructure area>]} ***set***}
is associated with
{{{ <gate> }* { <graphical gate constraint> }* } ***set*** }

<block type heading> ::=
[<virtuality>]
block type {<block type name> | <block type identifier>}
[<formal context parameters>] [<virtuality constraint>]
[<specialization>]

<type in block area>::=
<block type diagram>
| <block type reference>
| <process type diagram>
| <process type reference>
| <service type diagram>
| <service type reference>
| <procedure diagram>
| <graphical procedure reference>

<block type reference>::=
<block type symbol> ***contains***
{[<virtuality>] <block name> }

<block type symbol>::=

Semántica

Una <block type definition> define un tipo de bloque. Todos los bloques de un tipo de bloque tienen las mismas propiedades definidas para ese tipo de bloque.

6.1.1.3 Tipo de proceso

Gramática textual concreta

<process type definition> ::=
[<virtuality>]
process type { <process type name> | <process type identifier> }
[<formal context parameters>]
[<virtuality constraint>]
[<specialization>] <end>
[<formal parameters> <end>] [<valid input signal set>]
{<gate definition> }*
{<entity in process>}*
[<process type body>]
endprocess type [<process type name> | <process type identifier>] <end>

<process type body> ::=
<procedure body>

<textual process type reference> ::=
[<virtuality>] **process type** <process type name>
referenced <end>

Un <formal context parameter> de <formal context parameters> no debe ser un <variable context parameter> ni <timer context parameter>.

Gramática gráfica concreta

<process type diagram> ::=
<frame symbol>
contains {<process type heading>
{ {<process text area>}*
{<type in process area>}*
{<macro diagram> }*
{<process type graph area> | <service interaction area> }}***set*** }
is associated with
{{ {<gate>}* {<graphical gate constraint>}*} ***set*** }

<process type heading> ::=
[<virtuality>]
process type {<process type name> | <process type identifier>}
[<formal context parameters>]
[<virtuality constraint>]
[<specialization>] [<end>]
[<formal parameters>]

<process type graph area> ::=
[<start area>] { <state area> | <in-connector area> }*

<type in process area>::=
<service type diagram>
| <service type reference>
| <procedure diagram>
| <graphical procedure reference>

<process type reference>::=
<process type symbol> ***contains***
{[<virtuality>] <process type name> }

<process type symbol>::=

Semántica

Una <process type definition> define un tipo de proceso. Todos los conjuntos de instancias de proceso de un tipo de proceso tienen las mismas propiedades definidas para ese tipo de proceso.

El conjunto completo de señales de entrada válidas de un tipo de proceso es la unión del conjunto completo de señales de entrada válidas de su supertipo, las <signal list>s en todas las puertas en el sentido de flujo hacia el tipo de proceso, el <valid input signal set>, las señales de entrada implícitas introducidas por los conceptos adicionales de 4.10 a 4.14 y las señales de temporizador.

Las señales mencionadas en <output>s de un tipo de proceso deben estar en el conjunto completo de señales de entrada válidas del tipo de proceso o en la <signal list> de una puerta en el sentido de flujo procedente del tipo de proceso.

6.1.1.4 Tipo de servicio

Gramática textual concreta

<service type definition> ::=
[<virtuality>]
service type{<service type name> | <service type identifier>}
[<formal context parameters>]
[<virtuality constraint>]
[<specialization>] <end>
[<valid input signal set>]
{<gate definition>}*
{<entity in service>}*
[<service type body>]
endservice type [{<service type name> | <service type identifier>}] <end>

<service type body>::=
<process type body>

<textual service type reference>::=
[<virtuality>] **service type** <service type name>
referenced <end>

Un <formal context parameter> de <formal context parameters> no debe ser un <timer context parameter>.

Una <variable definition> en una <service type definition> no debe contener la palabra clave **revealed**.

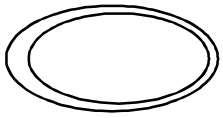
Gramática gráfica concreta

<service type diagram> ::=
<frame symbol> ***contains***
{<service type heading>
{ {<service text area> }*
{<graphical procedure reference>}*
{<procedure diagram>}*
{<macro diagram>}*
<service graph area> } ***set*** }
is associated with
{ { {<gate>}* {<graphical gate constraint>}*} ***set*** }

<service type heading> ::=
[<virtuality>]
service type {<service type name> | <service type identifier>}
[<formal context parameters>]
[<virtuality constraint>]
[<specialization>]

<service type reference>::=
<service type symbol> ***contains***
{[<virtuality>] <service type name>}

<service type symbol> ::=



Semántica

Una <service type definition> define un tipo de servicio. Todos los servicios de un tipo de servicio tienen las propiedades definidas para ese tipo de servicio.

El conjunto completo de señales de entrada válidas de un tipo de servicio es la unión del conjunto completo de señales de entrada válidas de su supertipo, las <signal list>s en todas las puertas en el sentido de flujo hacia el tipo de servicio, el <valid input signal set>, las señales de entrada implícitas introducidas por los conceptos adicionales de 4.10 a 4.14 y las señales de temporizador.

Las señales mencionadas en <output>s de un tipo de servicio deben estar en el conjunto completo de señales de entrada válidas del tipo de servicio o en la <signal list> de una puerta en el sentido de flujo procedente del tipo de servicio.

6.1.2 Expresión de tipo

Una expresión de tipo se utiliza para definir un tipo en función de otro como se indica en 6.3, especialización.

Gramática textual concreta

<type expression>::=
<base type> [<actual context parameters>]

<base type>::=
<identifier>

<actual context parameters> sólo puede especificarse si <base type> denota un tipo parametrizado. Los parámetros de contexto se definen en 6.2.

Fuera de un tipo parametrizado, el tipo parametrizado sólo puede utilizarse refiriéndolo a su <identifier> en <type expression>.

Semántica

Una <type expression> indica el tipo identificado por el identificador de <base type> si no hay parámetros de contexto reales, o un tipo anónimo definido aplicando los parámetros de contexto reales a los parámetros de contexto formales del tipo parametrizado denotado por el identificador de <base type>.

Si se omiten algunos parámetros de contexto reales, el tipo sigue estando parametrizado.

Una <type expression> no representa especialización si el <base type> es un tipo parametrizado (véase 6.3).

NOTA – la definición denotada por el <base type> cumple cualesquiera condiciones estáticas, la utilización de la <type expression> puede violar las propiedades asociadas con el <base type>. Las propiedades pueden ser violadas en los casos siguientes:

1) Cuando una unidad de ámbito tiene parámetros de contexto de señal o parámetros de contexto de temporizador, la condición de que los estímulos para un estado deben estar disjuntos depende de los parámetros de contexto reales que se utilizarán.

2) Cuando una salida en una unidad de ámbito refiere a una puerta, una ruta de señales o un canal que no están definidos en el tipo circundante más próximo que tiene puertas, la instanciación de ese tipo resulta en una especificación errónea si no hay trayecto de comunicación hacia la puerta.

3) Cuando un procedimiento contiene referencias a identificadores de señal, variables remotas y procedimientos remotos, la especialización de ese procedimiento dentro de un proceso o servicio resulta en una especificación errónea si la utilización de esos identificadores dentro del procedimiento viola la utilización válida para el proceso o servicio.

4) Cuando una acción de crear, acción de salida o definición de visión dentro de un tipo de proceso o de servicio definido en un bloque refiere a un conjunto de instancias de proceso, la especialización y/o instanciación del tipo de proceso en una subestructura del bloque resulta en una especificación errónea.

5) Cuando tipos de servicios son instanciados, el proceso resultante es erróneo si dos o más servicios tienen la misma señal en el conjunto completo de señales de entrada válidas.

6) Cuando una unidad de ámbito tiene un parámetro de contexto de proceso que se utiliza en una acción de salida, la existencia de un posible trayecto de comunicación depende del parámetro de contexto real que se utilizará.

7) Cuando una unidad de ámbito tiene un parámetro de contexto de género y un operador en la signatura de género es utilizado en axiomas, la aplicación de un parámetro de contexto de género real, para el cual el operador se define utilizando una definición de operador, resulta en una especificación errónea.

8) Si un parámetro formal de un procedimiento añadido en una especialización tiene el <parameter kind> **in**/**out**, una llamada en el supertipo a un subtipo (utilizando **this**) resultará en la omisión de un parámetro real **in**/**out,** es decir, una especificación errónea.

9) Si un parámetro formal de contexto de procedimiento se define con la limitación **atleast** y el parámetro de contexto real tiene añadido un parámetro de <parameter kind> **in**/**out**, una llamada del parámetro de contexto de procedimiento formal en el tipo parametrizado puede resultar en la omisión de un parámetro real **in**/**out**, es decir una especificación errónea.

Modelo

Si la unidad de ámbito contiene <specialization> y se omite cualesquiera <actual context parameter>s en la <type expression>, los <formal context parameter>s son copiados (conservando su orden) e insertados delante de los <formal context parameter>s (si los hay) de la unidad de ámbito. En lugar de los <actual context parameter>s omitidos, se insertan los nombres de los <formal context parameter>s correspondientes. Estos <actual context parameter>s tienen ahora el contexto definidor en la unidad de ámbito vigente.

6.1.3 Definiciones basadas en tipos

Una definición de sistema, bloque, proceso o servicio basada en tipos define un sistema, bloque, conjunto de instancias de proceso o servicio, respectivamente, de acuerdo con un tipo denotado por <type expression>. Las entidades definidas obtienen las propiedades de los tipos en los cuales se basan.

6.1.3.1 Definición de sistema basada en tipo de sistema

Gramática textual concreta

<textual typebased system definition>::=

<typebased system heading> <end>

<typebased system heading> ::=

system <system name> : <system type expression>

Gramática gráfica concreta

<graphical typebased system definition>::=

<frame symbol> ***contains*** <typebased system heading>

Semántica

Una definición de sistema basada en tipos define una *System-definition* derivada por transformación a partir de un tipo de sistema.

Modelo

Una <textual typebased system definition> o una <graphical typebased system definition> se transforma en una <system definition> que tiene las definiciones del tipo de sistema definido por <system type expression>.

6.1.3.2 Definición de bloque basada en tipo de bloque

Gramática textual concreta

<textual typebased block definition>::=
block <typebased block heading> <end>

<typebased block heading> ::=
<block name> [<number of block instances>] : <block type expression>

<number of block instances>::=
(<Natural simple expression>)

Si se omite <number of block instances> el número de bloques es 1. El <number of block instances> debe ser igual o superior a 1.

Gramática gráfica concreta

<graphical typebased block definition>::=
<block symbol> ***contains***
{ <typebased block heading>
{ <gate>* }***set*** }

<existing typebased block definition>::=
<dashed block symbol> ***contains*** { <block identifier> { <gate>* }***set*** }

<dashed block symbol>::=

Las <gate>s están situadas cerca de la frontera de los símbolos y asociadas con el punto de conexión a canales.

Semántica

Una definición de bloque basada en tipo define *Block-definition*s derivadas por transformación a partir de un tipo de bloque.

Modelo

Una <textual typebased block definition> o una <graphical typebased block definition> se transforma en una <block definition> que tiene las definiciones del tipo de bloque definido por <block type expression>. El número de <block definition>s derivadas es especificado por <number of block instances>.

Una <existing typebased block definition> sólo puede aparecer en una definición de subtipo. Representa el bloque definido en el supertipo de la definición de subtipo. Pueden especificarse canales adicionales conectados a puertas del bloque existente.

6.1.3.3 Definición de proceso basada en tipo de proceso

Gramática textual concreta

<textual typebased process definition>::=
process <typebased process heading> <end>

<typebased process heading> ::=
<process name> [<number of process instances>] : <process type expression>

El tipo de proceso denotado por <base type> en <process type expression> debe contener una transición de arranque.

Gramática gráfica concreta

<graphical typebased process definition>::=
<process symbol> ***contains***
{ <typebased process heading>
{ <gate>* }***set*** }

<existing typebased process definition>::=
<dashed process symbol> ***contains*** { <process identifier> { <gate>* }***set*** }

<dashed process symbol>::=

Las <gate>s están situadas cerca de la frontera de los símbolos y asociadas con el punto de conexión a rutas de señales.

Semántica

Una definición de proceso basada en tipos define una *Process-definition* derivada por transformación a partir de un tipo de proceso.

La creación de instancias de proceso individuales se describe en 2.4.4 (inicialización de sistemas) y 2.7.2 (petición dinámica de crear).

Modelo

Una <textual typebased process definition> o una <graphical typebased process definition> se transforma en una <process definition> que tiene las definiciones del tipo de proceso definidas por <process type expression>.

Una <existing typebased process definition> sólo puede aparecer en una definición de subtipo. Representa el proceso definido en el supertipo de la definición de subtipo. Pueden especificarse rutas de señales adicionales conectadas a puertas del proceso existente.

6.1.3.4 Definición de servicio basada en tipo de servicio

Gramática textual concreta

<textual typebased service definition>::=
service <typebased service heading> <end>

<typebased service heading> ::=
<service name> : <service type expression>

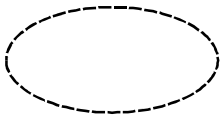
El tipo de servicio denotado por <base type> en <service type expression> debe contener una transición de arranque.

Gramática gráfica concreta

<graphical typebased service definition>::=
<service symbol> ***contains***
{ <typebased service heading>
{ <gate>* }***set*** }

<existing typebased service definition>::=
<dashed service symbol> ***contains*** { <service identifier> { <gate>* }***set*** }

<dashed service symbol> ::=



Las <gate>s están situadas cerca de la frontera de los símbolos y asociadas con el punto de conexión a rutas de señales.

Semántica

Una definición de servicio basada en tipos define una *Service-definition* derivada por transformación a partir de un tipo de servicio.

Modelo

Una <textual typebased service definition> o una <graphical typebased service definition> se transforma en una <service definition> que tiene las definiciones del tipo de servicio definidas por <service type expression>.

Una <existing typebased service definition> sólo puede aparecer en definiciones de subtipo. Representa un servicio especificado en el supertipo de la definición de subtipo. Pueden especificarse rutas de señales adicionales conectadas a puertas del servicio existente.

6.1.4 Puerta

Las puertas se definen en tipos de bloques, de procesos o de servicios (como se indica en 6.1.1) y representan puntos de conexión para canales y rutas de señales, que conectan instancias de estos tipos (como se indica en 6.1.3) con otras instancias de la misma clase de entidad o con el símbolo de recuadro que la contiene.

Gramática textual concreta

<gate definition>::=
gate <gate>
[**adding**] <gate constraint> <end>
[<gate constraint> <end>]

<gate> ::=
<gate name>

<gate constraint>::=
{ **out** [**to** <textual endpoint constraint>]
| **in** [**from**<textual endpoint constraint>] }
[**with** <signal list>]

<textual endpoint constraint>::=
[**atleast**]<identifier>

out o **in** denota el sentido de flujo de <signal list>, desde o hacia el tipo, respectivamente. Los tipos a partir de los cuales se definen instancias deben tener una <signal list> contenida en las <gate constraint>s.

El <identifier> de <textual endpoint constraint> debe denotar una definición de tipo de la misma clase de entidad que la definición de tipo en la cual se define la puerta.

Un canal/ruta de señales conectado a una puerta debe ser compatible con la limitación de puerta. Un canal/ruta de señales es compatible con una limitación de puerta si el otro punto extremo del canal/ruta de señales es un bloque/proceso/servicio del tipo denotado por <identifier> en la limitación del punto extremo o un subtipo de este tipo (en caso de **atleast**<identifier>), y si el conjunto de señales en el canal/ruta de señales es igual al conjunto de señales especificado para la puerta en el sentido de flujo correspondiente, o es un subconjunto del mismo.

<base type> en <textual typebased block definition> o <textual typebased process definition> debe contener, para cada combinación de (puerta, señal, sentido de flujo) definida por el tipo, por lo menos una ruta de señales que mencione **env**, la puerta y la señal para el sentido de flujo dado, si el tipo básico contiene rutas de señales.

Una <block substructure definition> en <base type> en <textual typebased block definition> debe contener, para cada combinación de (puerta, señal, sentido de flujo) definida por el tipo de bloque, por lo menos un canal que mencione **env**, la puerta y la señal para el sentido de flujo dado.

Cuando se especifican dos <gate constraint>s, una debe estar en el sentido de flujo contrario a la otra, y las <textual endpoint constraint>s de las dos <gate constraint>s deben ser iguales.

adding sólo se puede especificar en una definición de subtipo y sólo para una puerta definida en el supertipo. Cuando **adding** se especifica para una <gate>, cualesquiera <textual endpoint constraint>s y <signal list>s son adiciones a las <gate constraint>s de la puerta en el supertipo.

Si se especifica <textual endpoint constraint> para la puerta en el supertipo, el <identifier> de una <textual endpoint constraint> (añadida) debe denotar el mismo tipo que el tipo denotado en la <textual endpoint constraint> del supertipo, o un subtipo del mismo.

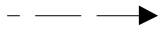
Gramática gráfica concreta

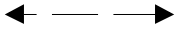
<graphical gate constraint> ::=
{ <gate symbol> | <existing gate symbol> }
is associated with
{<gate> [<signal list area>[<signal list area>]] }
is connected to
{ <frame symbol> [<endpoint constraint>] }

<endpoint constraint> ::=
{ <block symbol> | <process symbol> | <service symbol> }
contains <textual endpoint constraint>

<gate symbol> ::=
<signal route symbol>

<existing gate symbol> ::=
<gate symbol 1> | <gate symbol 2>

<gate symbol 1> ::=


<gate symbol 2> ::=


La <graphical gate constraint> está fuera del recuadro de diagrama.

Los elementos <signal list area> están asociados con los sentidos de flujo del símbolo de ruta de señales.

El símbolo en <endpoint constraint> debe ser el símbolo para definición de instancia correspondiente a la definición de tipo en la cual se define la puerta, es decir, <block symbol>, <process symbol> o <service symbol>.

<signal list area>s y <endpoint constraint> asociadas con un <existing gate symbol> se consideran como adiciones a las de la definición de puerta en el supertipo.

Un <existing gate symbol> sólo puede aparecer en una definición de subtipo, y es entonces un representante de la puerta con el mismo <gate name> especificado en el supertipo de la definición de subtipo.

Para cada punta de flecha en el <gate symbol>, puede haber una <signal list area>. Una <signal list area> debe estar inequívoca y suficientemente cerca de la punta de flecha a la cual está asociada. La punta de flecha indica si la <signal list area> denota **in** o **out** <gate constraint>.

Semántica

La utilización de puertas en definiciones de tipo corresponde a la utilización de trayectos de comunicación en el ámbito circundante en (un conjunto de) especificaciones de instancias.

Modelo

Para cada instancia de un tipo que define una <gate>, se deriva una <signal route to route connection>, una <channel to route connection> o una <channel connection>:

a) Para cada instanciación de un tipo de proceso, una <signal route to route connection> se deriva en la <process definition> resultante donde:

   – Los <external signal route identifiers> son las rutas de señales, definidas en el bloque circundante, que mencionan el proceso y la puerta en un <signal route path>.

   – Los <signal route identifiers> son las rutas de señales, definidas dentro de la <process definition>, que mencionan la palabra clave **env** y la puerta en el <signal route path>.

b) Para cada instanciación de un tipo de bloque, se deriva una <channel to route connection> en la <block definition> resultante donde:

   – Los <channel identifiers> son los canales definidos en la unidad de ámbito que contiene el bloque que mencionan el bloque y la puerta en un <channel path>.

   – Los <signal route identifiers> son las rutas de señales definidas dentro de la <block definition> que menciona la palabra **env** y la puerta en el <channel path>.

c) Para cada instanciación de un tipo de bloque que contiene una <block substructure definition> se deriva una <channel connection> en el ámbito resultante del bloque donde:

   – Los <channel identifiers> son los canales definidos en la unidad de ámbito que contiene el bloque circundante que menciona el bloque y la puerta en un <channel path>.

   – Los <subchannel identifiers> son los canales, definidos dentro de <block substructure definition>, que menciona la palabra clave **env** y la puerta en un <channel path>. Las reglas para utilizar el refinamiento de señal en instancias de bloques basadas en tipos se definen a través de las reglas para la <channel connection> resultante (véase 3.3).

6.2 Parámetro de contexto

Para que una definición de tipo pueda utilizarse en contextos diferentes, tanto en la misma especificación de sistema como en especificaciones de sistemas diferentes, los tipos pueden parametrizarse mediante parámetros de contexto. Los parámetros de contexto son sustituidos por identificadores reales, parámetros de contexto reales como se indica en 6.1.2.

Las siguientes definiciones de tipo pueden tener parámetros de contexto formales: tipo de sistema, tipo de bloque, tipo de proceso, tipo de servicio, procedimiento, señal y género.

Los parámetros de contexto pueden tener limitaciones, es decir las propiedades requeridas que debe tener cualquier entidad denotada por el identificador real correspondiente. Los parámetros de contexto tienen estas propiedades dentro del tipo.

Gramática textual concreta

<formal context parameters>::=
<context parameters start> <formal context parameter>
{<end> <formal context parameter> }* <context parameters end>

<actual context parameters>::=
<context parameters start>
[<actual context parameter>] {, [<actual context parameter>] }* <context parameters end>

<actual context parameter> ::=
<identifier>

<context parameters start> ::=
<

<context parameters end> ::=
>

<formal context parameter>::=
<process context parameter>
| <procedure context parameter>
| <remote procedure context parameter>
| <signal context parameter>
| <variable context parameter>
| <remote variable context parameter>
| <timer context parameter>
| <synonym context parameter>
| <sort context parameter>

NOTA – los caracteres «<» y «>» delimitan parámetros de contexto, y no se utilizan únicamente como metasímbolos.

La unidad de ámbito de una definición de tipo con parámetros de contexto formales define los nombres de los parámetros de contexto formales. Estos nombres son, por lo tanto, visibles en la definición del tipo, y también en la definición de los parámetros de contexto formales.

Los parámetros de contexto formales no pueden utilizarse como <base type> en <type expression> ni en limitaciones **atleast** de <formal context parameters>.

Las limitaciones son especificadas por especificaciones de limitación. Una especificación de limitación introduce la entidad del parámetro de contexto formal seguida por una signatura de limitación o una cláusula **atleast**. Una signatura de limitación introduce directamente suficientes propiedades del parámetro de contexto formal. Una cláusula **atleast** denota que el parámetro de contexto formal debe ser sustituido por un parámetro de contexto real, que es el mismo tipo o un subtipo del tipo identificado en la cláusula **atleast**. Los identificadores que siguen a la palabra clave **atleast** en esta cláusula deben identificar definiciones de tipo de la clase de entidad del parámetro de contexto y no deben ser parámetros de contexto formales ni tipos parametrizados.

Un parámetro de contexto formal de un tipo debe estar vinculado únicamente a un parámetro de contexto real de la misma clase de entidad que cumple las limitaciones del parámetro formal.

El tipo parametrizado sólo puede utilizar las propiedades de un parámetro de contexto que estén dadas por la limitación, salvo en los casos indicados en 6.1.2.

Un parámetro de contexto que utiliza otros parámetros de contexto en su limitación no puede estar vinculado antes de que lo estén los demás parámetros.

La vinculación de una variable real o de un parámetro de contexto de sinónimo con su definición no está resuelta por el contexto.

Las comas finales pueden omitirse en <actual context parameters>.

Semántica

Los parámetros de contexto formales de una definición de tipo que no es una definición de subtipo ni está definida por parámetros de contexto formales vinculantes en una <type expression> son los parámetros especificados en los <formal context parameters>.

Los parámetros de contexto de un tipo están vinculados en la definición de una <type expression> o una <inheritance rule> con parámetros de contexto reales. En esta vinculación, las apariciones de parámetros formales de contexto dentro del tipo parametrizado son sustituidas por los parámetros reales. Durante esta vinculación de identificadores contenidos en <formal context parameter>s con definiciones (es decir, que derivan su calificador, véase 2.2.2), se pasan por alto las definiciones locales que no sean <formal context parameters>s.

Los tipos parametrizados no pueden ser parámetros de contexto reales. Para que una definición se admita como parámetro de contexto real debe ser de la misma clase de entidad que el parámetro formal y cumplir la limitación del parámetro formal.

Modelo

Si una unidad de ámbito contiene <specialization>, cualquier parámetro de contexto real omitido en la <specialization> es sustituido por el <formal context parameter> correspondiente del <base type> en la <type expression>, y este <formal context parameter> se convierte en un parámetro formal de contexto de la unidad de ámbito.

6.2.1 Parámetro de contexto de proceso

Gramática textual concreta

<process context parameter>::=
process <process name> <process constraint>

<process constraint>::=
[**atleast**] <process identifier> | <process signature>

<process signature> ::=
[[<end>] <formal parameters signature>]

<formal parameters signature>::=
fpar <sort>{**,** <sort> }*

Semántica

Un parámetro de proceso real debe identificar una definición de proceso. Su tipo debe ser un subtipo del tipo de proceso de limitación (**atleast** <process identifier>), sin adición de parámetros formales a los del tipo de limitación, o debe ser el tipo denotado por <process identifier>, o debe ser compatible con la signatura de proceso formal. Una definición de proceso es compatible con la signatura de proceso formal si los parámetros formales de la definición de proceso tienen los mismos géneros que los <sort>s correspondientes de la <formal parameters signature>.

6.2.2 Parámetro de contexto de procedimiento

Gramática textual concreta

<procedure context parameter>::=
procedure <procedure name> <procedure constraint>

<procedure constraint>::=
atleast <procedure identifier>
| <procedure signature>

<procedure signature>::=
[[<end>] **fpar** <procedure formal parameter constraint>
{**,** <procedure formal parameter constraint> }*
[<end> **returns** <sort>]]
| [<end>] **returns** <sort>

<procedure formal parameter constraint> ::=
<parameter kind> <sort>

Semántica

Un parámetro de procedimiento real debe identificar una definición de procedimiento que es una especialización del procedimiento de la limitación (**atleast** <procedure identifier>), o que es compatible con la signatura de procedimiento formal. Una definición de procedimiento es compatible con la signatura de procedimiento formal:

a) si los parámetros formales de la definición de procedimiento tienen los mismos géneros que los parámetros correspondientes de la signatura, si tienen la misma <parameter kind>, y si ambos devuelven un valor del mismo <sort> o si ninguno retorna un valor; o

b) si esta regla se cumple después de sustituir una especificación de resultado en un parámetro **in/out** adicional; y

c) si cada parámetro **in/out** en la definición de procedimiento tiene el mismo <sort identifier> o <syntype identifier> que el parámetro correspondiente de la signatura.

6.2.3 Parámetro de contexto de procedimiento remoto

Gramática textual concreta

<remote procedure context parameter>::=
remote procedure <procedure name> <procedure signature>

Semántica

Un parámetro real de un parámetro de contexto de procedimiento **remote** debe identificar una <remote procedure definition> con la misma signatura.

6.2.4 Parámetro de contexto de señal

Gramática textual concreta

<signal context parameter>::=
signal <signal name> <signal constraint>
{, <signal name> <signal constraint> }*

<signal constraint>::=
atleast <signal identifier>
| <signal signature>

<signal signature>::=
[<sort list>]
[<signal refinement>]

Semántica

Un parámetro de señal real debe identificar una definición de señal que es un subtipo del tipo de señal de la limitación (**atleast** <signal identifier>) o que es compatible con la signatura de señal formal.

Una definición de señal es compatible con una signatura de señal formal si los géneros de la señal son los mismos que en la lista de limitaciones de género, y si <signal refinement> del <signal context parameter> está indicado, las definiciones de señal del <signal refinement> son compatibles con el <signal refinement> del <signal context parameter>.

Dos <signal refinement>s son compatibles si definen el mismo conjunto de nombres de señal, y si cada <subsignal definition> correspondiente tiene el mismo atributo <reverse>.

6.2.5 Parámetro de contexto de variable

Gramática textual concreta

<variable context parameter>::=

dcl <variable name> {,<variable name>}* <sort>
{, <variable name> {,<variable name>}* <sort> }*

Semántica

Un parámetro real debe ser una variable, un proceso o un parámetro de procedimiento formal del mismo género que el de la limitación.

6.2.6 Parámetro de contexto de variable remota

Gramática textual concreta

<remote variable context parameter>::=

remote <remote variable name> {,<remote variable name>}* <sort>
{, <remote variable name> {,<remote variable name>}* <sort> }*

Semántica

Un parámetro real debe identificar una <remote variable definition> del mismo género.

6.2.7 Parámetro de contexto de temporizador

Gramática textual concreta

<timer context parameter>::=

timer <timer name> <timer constraint>
{, <timer name> <timer constraint> }*

<timer constraint> ::=

[<sort list>]

Semántica

Un parámetro de temporizador real debe identificar una definición de temporizador que es compatible con la lista de limitación de género formal. Una definición de temporizador es compatible con una lista de limitación de género formal si los géneros del temporizador son los mismos que en la lista de limitación de género.

6.2.8 Parámetro de contexto de sinónimo

Gramática textual concreta

<synonym context parameter>::=

synonym <synonym name> <synonym constraint>
{, <synonym name> <synonym constraint> }*

<synonym constraint> ::=

<sort>

Semántica

Un sinónimo real debe ser del mismo género que el género de la limitación.

6.2.9 Parámetro de contexto de género

Gramática textual concreta

<sort context parameter>::=

newtype <sort name> [<sort constraint>]

<sort constraint> ::=

atleast <sort>

| <sort signature>

<sort signature>::=

<operators> **endnewtype** [<sort name>]

Semántica

Si se omite <sort constraint>, el género real puede ser cualquier género. En los demás casos, un género real debe ser un subtipo sin <literal renaming> o <inheritance list> del género de la limitación (**atleast** <sort>), o ser compatible con la signatura de género formal. Un género es compatible con la signatura de género formal si los literales del género incluyen los literales en la signatura de género formal, los operadores del género incluyen los operadores en la signatura de género formal y los operadores tienen las mismas signaturas.

Una <sort signature> incluye implícitamente los operadores igual y no igual (véase 5.3.1.4). **noequality** no se autoriza en <sort signature>.

ordering en <sort signature> da la signatura para los operadores de ordenación como se indica en 5.3.1.8.

6.3 Especialización

Para expresar una especialización de concepto, un tipo se puede definir como una especialización de otro tipo (el supertipo), que da un nuevo subtipo. Un subtipo puede tener propiedades además de las propiedades del supertipo, y puede redefinir tipos virtuales locales y transiciones.

La especialización está en la definición de tipo especializada especificada por «**inherits** <type expression>», donde <type expression> denota el tipo general. Se dice que el tipo general es el supertipo del tipo especializado, y se dice que el tipo especializado es un subtipo del tipo general. Cualquier especialización del subtipo es un subtipo del tipo general.

Obsérvese que toda la <type expression> representa el supertipo. El supertipo se denomina únicamente si la <type expression> sólo contiene <base type>.

Los tipos virtuales pueden tener limitaciones, es decir, propiedades que debe tener cualquier redefinición del tipo virtual. Estas propiedades se utilizan para garantizar propiedades de cualquier redefinición. Los tipos virtuales se definen en 6.3.2.

6.3.1 Adición de propiedades

Gramática textual concreta

<specialization>::=

inherits <type expression> [**adding**]

Si un tipo subT se deriva de una (super)tipo T a través de una especialización (ya sea directa o indirectamente), entonces

a) T no debe contener subT;

b) T no debe derivarse de subT;

c) las definiciones contenidas por T no deben derivarse de subT.

Semántica

El contenido resultante de una definición de tipo especializada con definiciones locales consiste en el contenido del supertipo seguido por el contenido de la definición especializada. Ello implica que el conjunto de definiciones de la definición especializada es la unión de las indicadas en la definición especializada propiamente dicha y las del supertipo. El conjunto de definiciones resultante debe cumplir las reglas para nombres distintos indicadas en 2.2.2. No obstante, hay tres excepciones a esta regla:

1) una redefinición de un tipo virtual es una definición con el mismo nombre que el del tipo virtual;

2) una puerta del supertipo puede tener una definición ampliada (en términos de señales transportadas y limitaciones de punto extremo) en un subtipo, esto es, especificado por una definición de puerta con el mismo nombre que el del supertipo;

3) si la <type expression> contiene <actual context parameters> cualquier ocurrencia del <base type> de la <type expression> es sustituida por el nombre del subtipo.

La <block substructure definition> dada en una definición de tipo de bloque especializado se añade a la definición de subestructura del supertipo de bloque. Si está presente, el nombre de la subestructura del subtipo debe ser el mismo que el de la subestructura del supertipo.

Los parámetros de contexto formales de un subtipo son los parámetros de contexto formales no vinculados de la definición del supertipo, seguidos de los parámetros de contexto formales del tipo especializado (véase 6.2).

Los parámetros formales de un tipo de proceso o de un procedimiento especializado son los parámetros formales del supertipo de proceso o procedimiento seguidos de los parámetros formales añadidos en la especialización.

El conjunto completo de señales de entrada válidas de un tipo de proceso o servicio especializado es la unión del conjunto completo de señales de entrada válidas del tipo de proceso o servicio especializado y el conjunto completo de señales de entrada válidas del supertipo de proceso o de servicio, respectivamente.

El gráfico resultante de un tipo de proceso, tipo de servicio o definición de procedimiento especializados consiste en el gráfico de su definición de supertipo seguido por el gráfico del tipo de proceso, tipo de servicio o definición de procedimiento especializados.

El gráfico de proceso de un determinado tipo de proceso, tipo de servicio o definición de procedimiento puede tener a lo sumo una transición de arranque.

Todos los <connector name>s definidos en el <body> combinado deben ser distintos. Puede tenerse una unión del <body> del proceso/procedimiento/servicio especializado con un conector definido en el supertipo.

Una definición de señal especializada puede añadir (adjuntando) géneros a la lista de géneros del supertipo.

Una definición de tipo parcial especializada puede añadir propiedades en términos de operadores, literales, axiomas, definiciones de operador y asignación por defecto.

NOTA – Una puerta en un subtipo es una extensión de una puerta existente en un supertipo, el <existing gate symbol> se utiliza en SDL/GR.

6.3.2 Tipo virtual

Un tipo que está definido localmente con una definición de tipo (tipo *enclosing*) puede ser especificado para que sea un tipo virtual. Los tipos de bloque, tipos de proceso, tipos de servicio y procedimientos pueden especificarse como tipos virtuales. Un tipo virtual puede estar limitado por otro tipo de la misma clase de entidad y puede estar redefinido en subtipos de su tipo circundante.

Gramática textual concreta

<virtuality>::=

virtual | redefined | finalized

<virtuality constraint>::=

atleast <identifier>

La sintaxis de los tipos virtuales se presentan en 6.1.1.2 (tipo de bloque), 6.1.1.3 (Tipo de proceso), 6.1.1.4 (tipo de servicio) y 2.4.6 (procedimiento).

Un tipo virtual y su limitación no pueden tener parámetros de contexto.

Un tipo virtual puede estar limitado por el tipo identificado por el <identifier> que sigue a la palabra clave **atleast**. Este <identifier> debe identificar una definición de tipo de la misma clase de entidad que el tipo virtual. Si se omite <virtuality constraint>, ello corresponde a la especificación del propio tipo virtual como la limitación.

Si <virtuality> está presente en la referencia y en la definición referenciada, deben ser iguales. Si <procedure preamble> está presente en la referencia de procedimiento y en la definición de procedimiento referenciada, deben ser iguales.

Un tipo virtual debe tener <virtuality> en su definición.

Un tipo virtual debe tener en sus puertas los mismos parámetros formales, puertas y señales que su limitación.

Semántica

Un tipo virtual puede ser redefinido en la definición de un subtipo del tipo que contiene el tipo virtual. En el subtipo, es la definición del subtipo la que define el tipo de instancias del tipo virtual, también cuando se aplica el tipo virtual en partes del subtipo heredado del supertipo. Un tipo virtual que no está redefinido en una definición de subtipo tiene la definición indicada en la definición de supertipo.

El acceso a un tipo virtual mediante un calificador que denota uno de los supertipos implica, no obstante, la aplicación de la (re)definición del tipo virtual dada en el supertipo real denotado por el calificador. Un tipo (T) cuyo nombre está oculto en un subtipo circundante por una redefinición (de T) puede hacerse visible a través de la calificación con un nombre de supertipo (es decir, un nombre de tipo en la cadena de herencia). El calificador consistirá solamente en un ítem de trayecto que denota el supertipo particular, o en caso de acceso a un tipo oculto desde una subestructura del supertipo, el calificador consistirá en dos ítems de trayecto, el primero que denota el tipo de bloque y el segundo que denota la subestructura.

Tanto en la definición de tipo circundante como en cualquier subtipo del tipo circundante, la definición de tipo virtual debe ser una especialización de su limitación. En una especialización del tipo circundante, una nueva definición del tipo virtual viene dada por una definición de tipo con el mismo nombre y la palabra clave **redefined**. Un tipo virtual redefinido sigue siendo un tipo virtual. La palabra clave **finalized** en lugar de **redefined** indica que el tipo no es virtual y no puede, por lo tanto, recibir otras definiciones en redefiniciones de subtipo ulteriores.

Un tipo virtual con una limitación explícita pero sin una herencia explícita hereda implícitamente del tipo de limitación.

Un tipo redefinido o finalizado sin una limitación explícita ni una herencia explícita hereda implícitamente de la limitación del tipo virtual correspondiente.

Un subtipo de un tipo virtual es un subtipo del tipo virtual original y no de una posible redefinición.

6.3.3 Transición/conservación virtual

En esta subcláusula se describe el arranque, la entrada y la conservación virtuales presentadas en otras secciones.

Las transiciones o conservaciones de un tipo de proceso, tipo de servicio o procedimiento se especifican para que sean transiciones o conservaciones virtuales mediante la palabra clave **virtual**. Las transiciones o conservaciones virtuales pueden redefinirse en especializaciones. Esto es indicado por transiciones o conservaciones, respectivamente, con igual (estado, señal) y con la palabra clave **redefined** o **finalized**.

Gramática concreta

Las sintaxis de transición y conservación virtuales se presentan en 2.4.6 (arranque de procedimiento virtual), 2.6.2 (arranque de proceso virtual), 2.6.4 (entrada virtual), 2.6.5 (conservación virtual), 2.6.6 (transición espontánea virtual), 4.10 (entrada prioritaria virtual), 4.11 (señal continua virtual), y 4.14 (entrada y conservación de procedimiento remoto virtuales).

Las transiciones o conservaciones virtuales no deben aparecer en definiciones (de conjunto de instancias) de proceso ni en definiciones de (instancia de) servicio.

Un estado no debe tener más de una transición espontánea virtual.

Una redefinición en una especialización marcada con **redefined** puede definirse diferentemente en otras especializaciones, mientras que una redefinición marcada con **finalized** no debe recibir ninguna nueva definición en especializaciones ulteriores.

Una entrada o conservación con <virtuality> no debe contener <asterisk>.

Semántica

La redefinición de transiciones/conservaciones virtuales corresponde estrechamente a la redefinición de tipos virtuales (véase 6.3.2).

Una transición de arranque virtual puede redefinirse en una nueva transición de arranque.

Una entrada prioritaria virtual o una transición de entrada puede redefinirse en una nueva entrada prioritaria o transición de entrada o en una conservación.

Una conservación virtual puede redefinirse en una entrada prioritaria, una transición de entrada o una conservación.

Una transición espontánea virtual puede redefinirse en una nueva transición espontánea.

Una transición continua virtual puede redefinirse en una nueva transición continua. La redefinición es indicada por el mismo (estado, [prioridad]) que la transición continua redefinida. Si existen varias transiciones continuas virtuales en un estado, cada una de ellas debe tener una prioridad distinta. Si sólo existe una transición continua virtual en un estado, puede omitirse la prioridad.

Una transición de una transición de entrada de procedimiento remoto virtual puede redefinirse en una nueva transición de entrada de procedimiento remoto o una conservación de procedimiento remoto.

Una conservación de procedimiento remoto virtual puede redefinirse en una transición de entrada de procedimiento remoto o una conservación de procedimiento remoto.

La transformación para transición de entrada virtual se aplica también a la transición de entrada de procedimiento remoto virtual.

La transformación de transiciones y conservaciones virtuales en estado asterisco se desarrolla en la etapa 14 de 7.1.

6.4 Ejemplos

La Figura 6.4.1 representa un paquete «lib». La Figura 6.4.2 representa un tipo de bloque (contenido en un paquete) que utiliza este paquete «lib» y otro paquete «slib» que está referenciado. La Figura 6.4.3 representa un diagrama de sistema con una cláusula **use**. Las Figuras 6.4.4 y 6.4.5 representan un tipo de bloque con puertas g1 y g2.

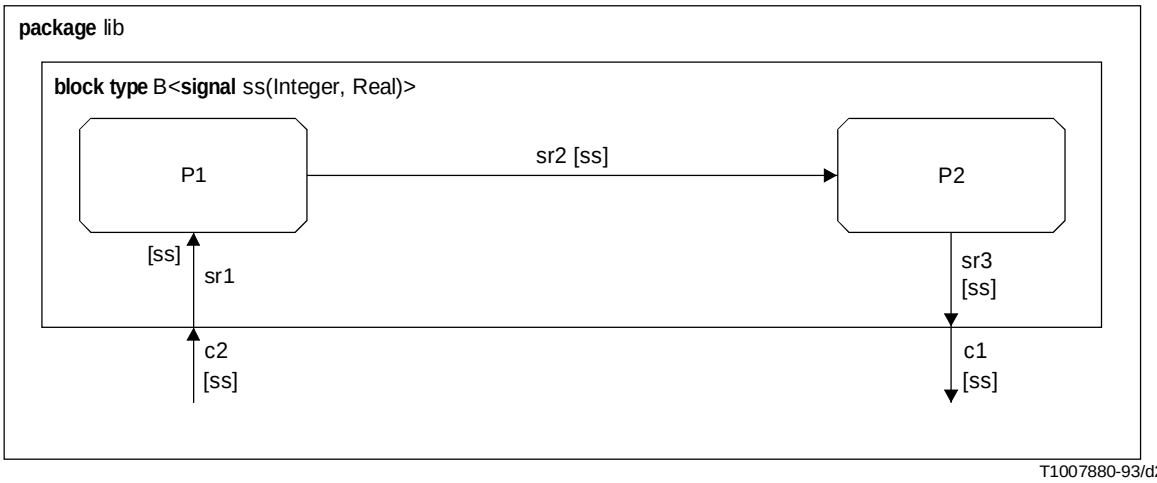


FIGURA 6.4.1/Z.100

Diagrama de paquete (SDL/GR)

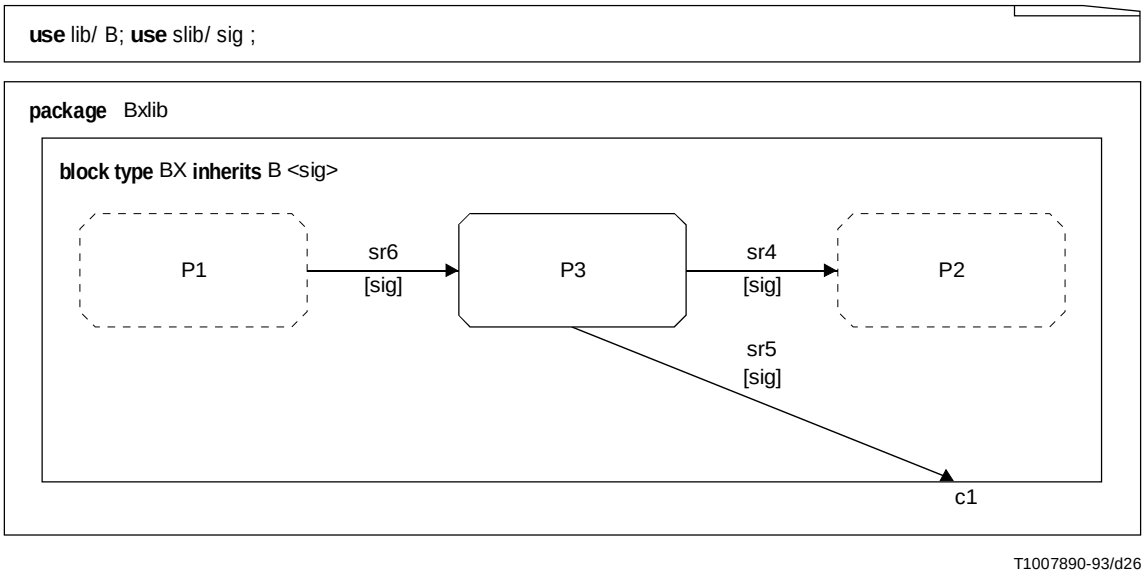


FIGURA 6.4.2/Z.100

Diagrama de paquete con cláusula use (SDL/GR)

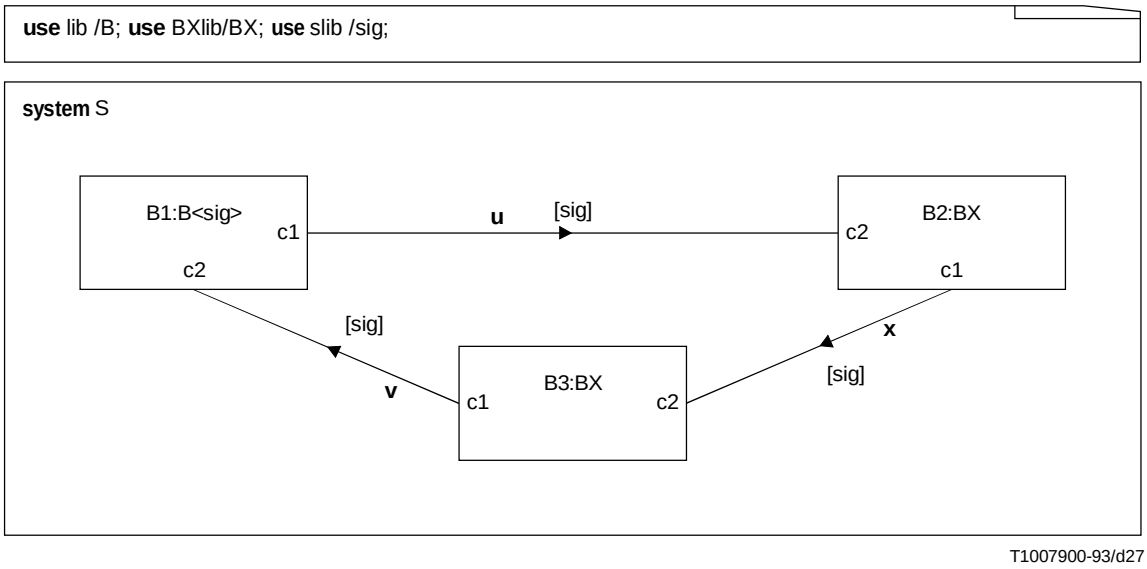


FIGURA 6.4.3/Z.100

Diagrama de sistema con cláusula use (SDL/GR)

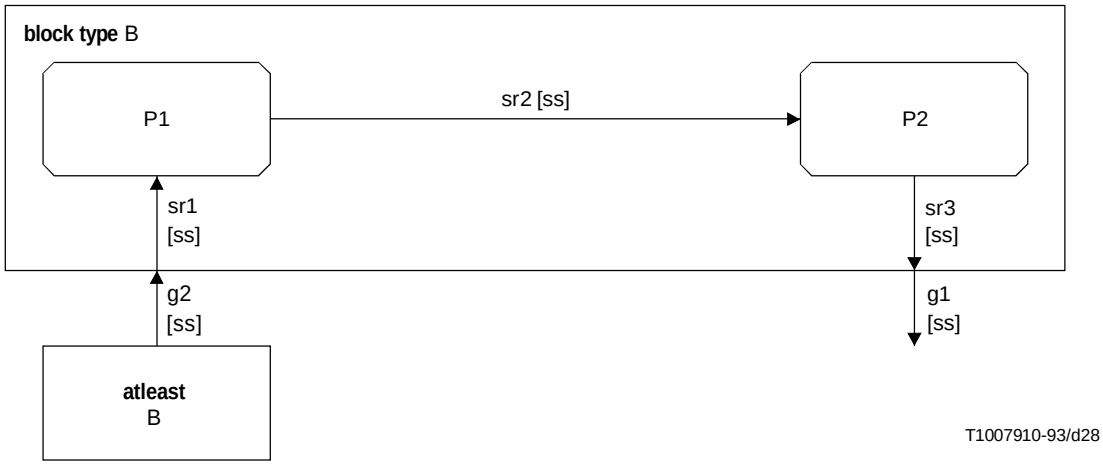


FIGURA 6.4.4/Z.100

Tipo de bloque con puertas g1 y g2 (SDL/GR)

```
block type B;
     gate  g1 out with ss;
     gate  g2 in from atleast B with  ss;
     signalroute sr2 from p1 to p2 with ss;
     signalroute sr1 from env via g2 to p1 with ss;
     signalroute sr3 from p2 to env via g1 with ss;
     process p1 referenced;
     process p2 referenced;
endblock type B;
```

FIGURA 6.4.5/Z.100

Tipo de bloque con puertas g1 y g2 (como en la Figura 6.5.4) (SDL/PR)

La Figura 6.4.6 representa un tipo de bloque BX que es un subtipo de B con una definición de proceso adicional, en SDL/GR, y con referencias a conjuntos de procesos existentes (P1 y P3) para conectar el proceso adicional a los mismos. La Figura 6.4.7 representa el mismo ejemplo en SDL/PR.

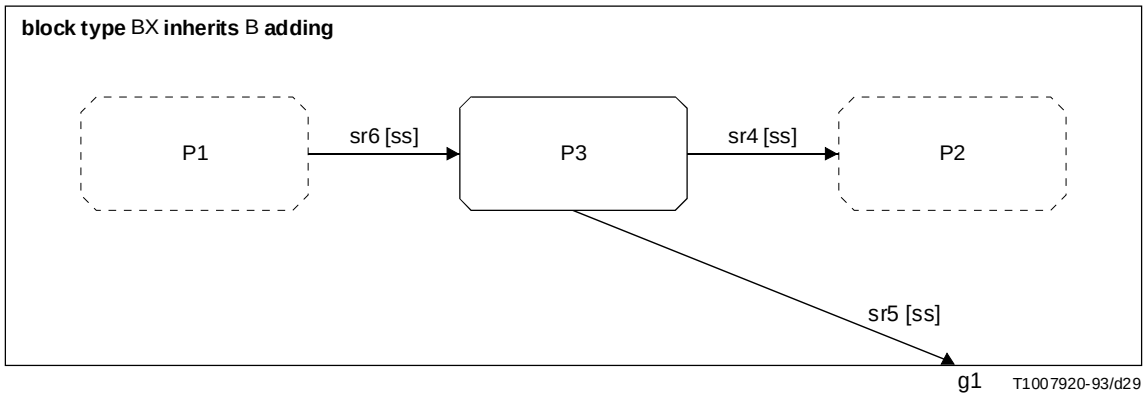


FIGURA 6.4.6/Z.100

Definición de subtipo (SDL/PR)

```
block type BX inherits B adding ;
     signalroute sr4 from p3 to p2 with ss;
     signalroute sr5 from p3 to env via g1 with ss;
     signalroute sr6 from p1 to p3 with ss;
process p3 referenced;
endblock type BX;
```

FIGURA 6.4.7/Z.100

Definición de subtipo en SDL/PR

Las Figuras 6.4.8 y 6.4.9 presentan la utilización de los tipos de bloque B y BX para especificar instancias de bloque, y conexiones a puertas.

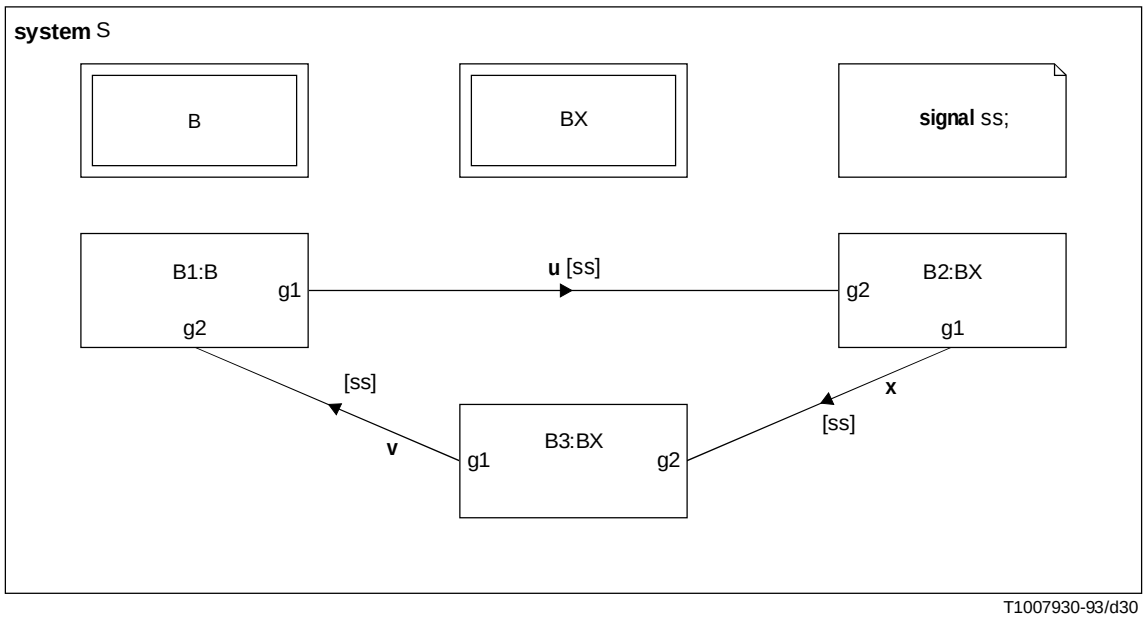


FIGURA 6.4.8/Z.100

Definición de sistema mediante tipos de bloques (SDL/GR)

```
system  S;
     signal ss;
     block type B referenced;
     block type BX referenced;

     block B1:B;
     block B2:BX;
     block B3:BX;

     channel u  from B1 via g1 to B2 via g2 with ss endchannel;
     channel v  from B3 via g1 to B1 via g2 with ss endchannel;
     channel x  from B2 via g1 to B3 via g2 with ss endchannel;

endsystem S;
```

FIGURA 6.4.9/Z.100

Definición de sistema mediante tipos de bloque (SDL/PR)

En la siguiente Figura 6.4.10 se dan ejemplos de tipos virtuales:

```
procedure Proc; fpar i,j Integer ... endprocedure;

process type P
      virtual  procedure VProc1 atleast Proc
        inherits Proc ... endprocedure;

      virtual procedure VProc2 ... endprocedure;

     ...  call VProc1(1,2)  ;   call VProc2; ...
endprocess type ;

process type  P1 inherits P;

     redefined  procedure VProc1  atleast VProc1
        inherits << process type P >> VProc1;
          ...
     endprocedure;

     finalized procedure VProc2 ... endprocedure;
  ...
endprocess type ;

process type  P2 inherits P1;

     finalized procedure VProc1
        inherits << process type P1 >> VProc1;
          ...
     endprocedure;
  ...
endprocess type ;
```

FIGURA 6.4.10/Z.100

Ejemplos de tipos virtuales (SLD/PR)

En P el procedimiento VProc1 es virtual; tiene que ser una especialización de Proc, y la definición de VProc1 hereda de Proc. Si se genera un proceso P, o si un proceso especializado no suministra una definición de VProc1, se aplica esta definición. VProc2 es también virtual en P, con una limitación implícita, siendo él mismo, es decir, las redefiniciones de VProc2 en subtipos de P deben ser subtipos de VProc2.

En P1 el procedimiento VProc1 sigue siendo virtual, limitado por sí mismo, mientras que VProc2 no es virtual. Las definiciones se aplican si un proceso P1 es generado, también para las llamadas efectuadas en la definición P. La definición de VProc1 es una especialización de VProc1 definido en el proceso P, por lo que tiene el calificador (**process type** P).

En P2 se da una nueva definición de VProc1. Esto es posible porque VProc1 es virtual en P1. Una redefinición de VProc2 no está autorizada en este caso, ya que ha sido finalizado en P1.

Las Figuras 6.4.11 a 6.4.16 presentan ejemplos de tipos correspondientes al ejemplo mostrado en las Figuras 2.10.5 a 2.10.10.

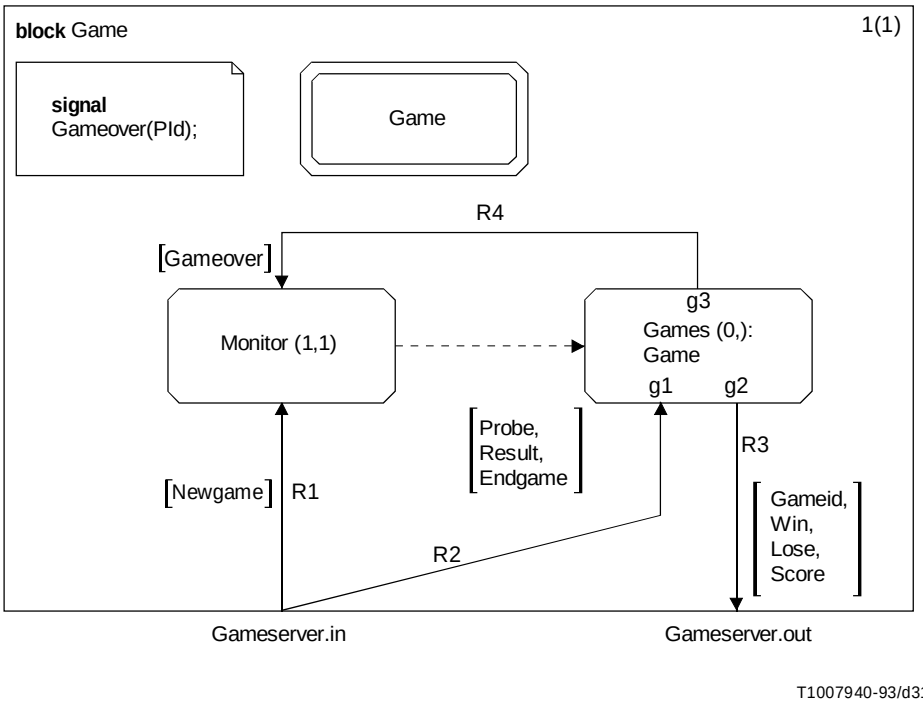


FIGURA 6.4.11/Z.100

Bloque con tipo de proceso (SDL/GR)

```
block Game;
     signal Gameover(PId);
     connect Gameserver.in and R1,R2;
     connect Gameserver.out and R3;
     signalroute R1 from env to Monitor with Newgame;
     signalroute R2 from env to Games via g1 with Probe, Result, Endgame;
     signalroute R3 from Games via g2 to env
        with Gameid, Win, Lose, Score;
     signalroute R4  from Games via g3 to Monitor with Gameover;

     process type Game referenced;

     process Monitor (1,1) referenced;

     process Games (0,) : Game referenced;

endblock Game;
```

FIGURA 6.4.12/Z.100

Bloque con tipo de proceso (SLD/PR)

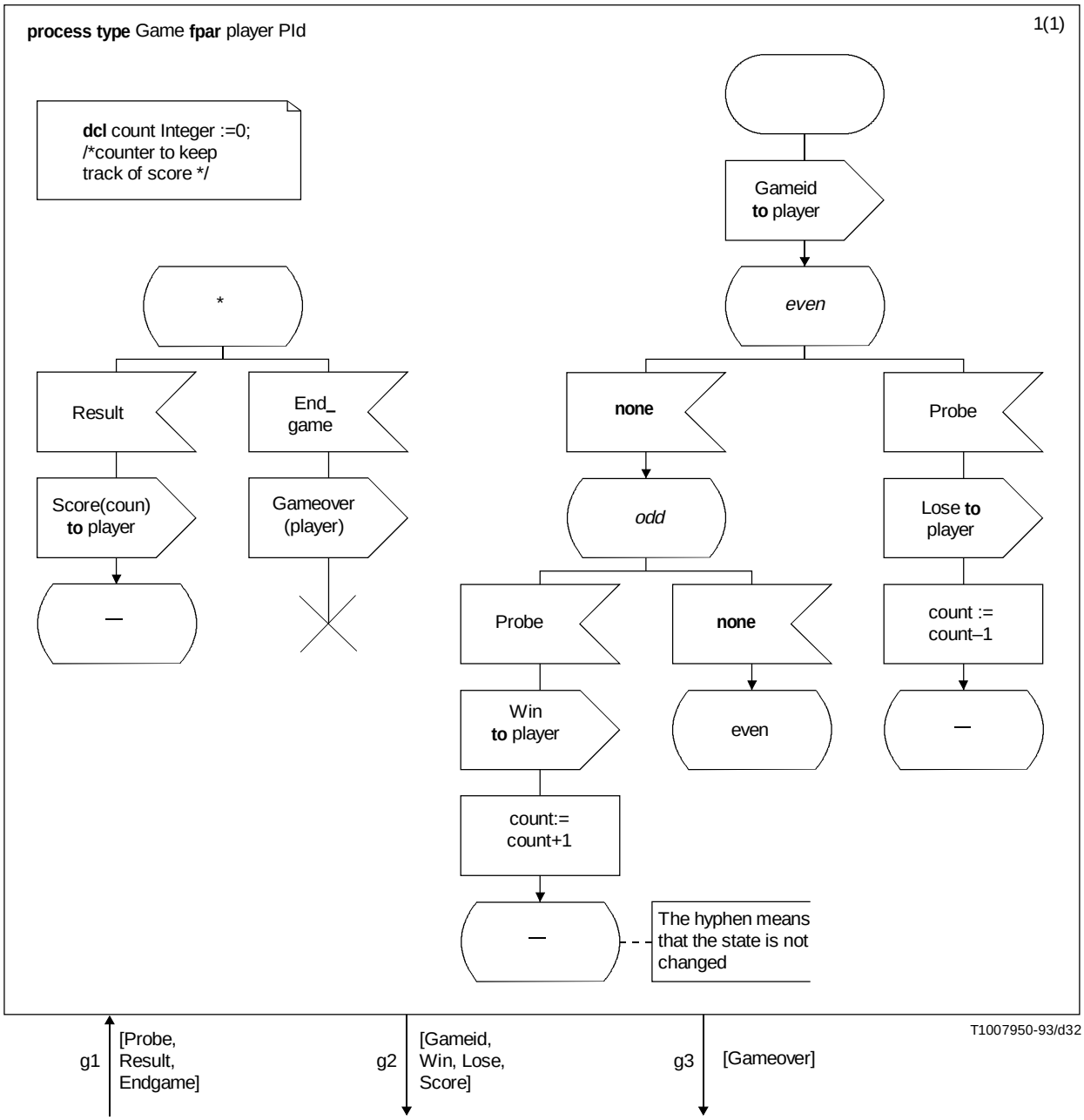


FIGURA 6.4.13./Z.100

Proceso de juego como un tipo de proceso (SDL/GR)

```
process type  Game;
   fpar player Pid;

   gate   g1 in with Probe, Result, Endgame;
          g2 out with Gameid, Win, Lose, Score;
          g3 out with Gameover;
   dcl count Integer:=0;

   start;
output Gameid to player;
   nextstate Even;
   state Even;
    input none;
      nextstate Odd;
    input Probe;
      output Lose to player;
      task count:= count - 1;
      nextstate -;

   state Odd;
    input none;
      nextstate Even;
    input Probe;
      output Win to Player;
      task count:= count + 1;
      nextstate -;

   state *;
    input Result;
      output Score(count) to player;
       nextstate -;

    input Endgame;
      output Gameover(player);
      stop;

endprocess type Game;
```

FIGURA 6.4.14/Z.100

Proceso de juego como un tipo de proceso (SDL/PR)

Las Figuras 6.4.15 y 6.4.16 representan una especialización SpecialGame (Juego especial) del tipo de proceso Game (Juego), teniendo en cuenta una nueva señal Evil de un nuevo proceso, Devil. En cualquiera de los estados de Game, la recepción de esta señal conduce al estado Even, que da al jugador mayores probabilidades de perder. Obsérvese que la parte Game sigue comportándose exactamente como un proceso Game. Si el Devil no envía ninguna señal Evil, el comportamiento es el mismo que en la instancia del supertipo, Game.

Obsérvese que la extensión del tipo de proceso Game requiere extensiones al bloque que contiene instancias del tipo de proceso especializado. Se supone que Evil entra por la misma puerta que Probe, Result y Endgame.

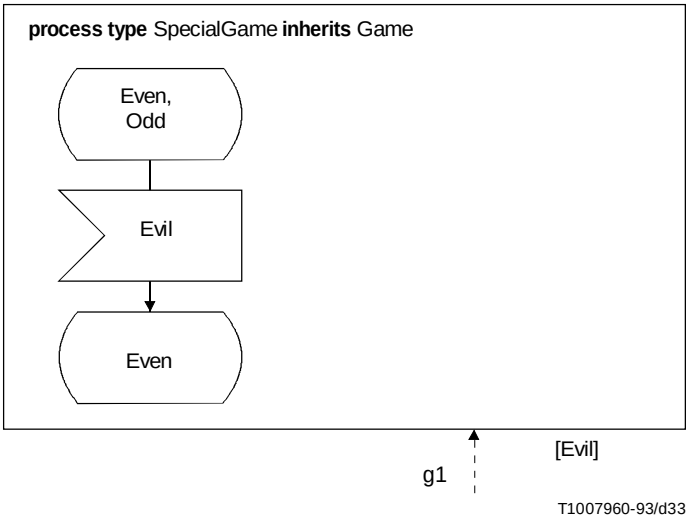


FIGURA 6.4.15/Z.100

Especialización del proceso Game (SDL/GR)

```
process type SpecialGame inherits Game;

gate g1 adding in with Evil;

     state Even, Odd;
        input Evil;
        nextstate Even;

endprocess type SpecialGame;
```

FIGURA 6.4.16/Z.100

Especialización del proceso Game (SDL/PR)

La Figura 6.4.17 muestra ejemplos de parámetros de contexto. En a) el género real debe ser una especialización del género t (visible en este punto). En b) basta que el primer parámetro real sea un subtipo de t, y que el segundo parámetro real sea un género con un operador que acepta dos valores del primer parámetro real y devuelve un valor del segundo parámetro. En c) el primer parámetro real debe ser un subtipo de SuperProc mientras que el segundo parámetro real debe ser cualquier procedimiento que acepte dos parámetros (**in**) del género Integer y Boolean.

```
signal S<newtype t1 atleast t>  (t1);     /* case a) */

procedure P<newtype t1 atleast t;       /* case b) */
                    newtype t2 operators op: t1,t1 -> t2 endnewtype>;
      dcl x11,x12 t1, x2 t2;
           start;
             task x2:=op(x11,x12);
             return;
endprocedure;

procedure P                              /* case c) */
        <  procedure Proc1  atleast SuperProc;
            procedure Proc2  fpar Integer, Boolean >;
        start;
           call Proc1;
           call Proc2(7, True);
           return;
endprocedure;
```

FIGURA 6.4.17/Z.100

Ejemplos de parámetros de contexto (SDL/PR)

7 Transformación de notaciones taquigráficas SDL

En esta cláusula se detalla la transformación de los constructivos SDL, cuya semántica dinámica se indica después de una transformación al subconjunto de SDL para el cual existe la *Gramática abstracta*. Estas notaciones taquigráficas son:

a) constructivos de las cláusulas 2, 3 y 5 para los cuales existe una sección *Modelo*; y

b) constructivos definidos en las cláusulas 4 y 6.

Las propiedades de una notación taquigráfica se derivan de la manera en que se modelan en función de (o transforman en) los conceptos primitivos. Para garantizar una utilización fácil e inequívoca de las notaciones taquigráficas, y reducir las repercusiones cuando se combinan varias notaciones taquigráficas, esos conceptos se transforman en un orden especificado como se indica a continuación.

El orden especificado de transformación significa que en la transformación de una notación taquigráfica de orden *n*, puede emplearse otra notación taquigráfica de orden *m*, siempre y cuando m>n.

Como no hay sintaxis abstracta para las notaciones taquigráficas, se utilizan en sus definiciones términos de sintaxis gráfica o de sintaxis textual. La elección entre términos de sintaxis gráfica y términos de sintaxis textual se basa en consideraciones prácticas, y no restringe la utilización de notaciones taquigráficas a una sintaxis concreta determinada.

Las transformaciones se describen en varios pasos enumerados. Un paso puede describir la transformación de varios conceptos y, por lo tanto, consta de varios subpasos, ya sea porque estos conceptos deben transformarse como un grupo o porque el orden de transformación entre estos conceptos no es importante. Este último caso se indica con un guión (-) en lugar de una enumeración.

7.1 Transformación de conceptos adicionales

1. Transformaciones léxicas

   1) Las <macro definition>s y <macro call>s (4.2) se identifican léxicamente y <macro call>s se amplían.

   2) Los <macro diagram>s se sustituyen por ocurrencias de <macro call area>.

   3) Los <underline> seguido de caracteres separadores se suprimen de nombres y los separadores en nombres se sustituyen por <underline> (2.2.1). Tras esta transformación la <sdl specification> se considera bien formada sintácticamente.

   4) Las <macro definition>s se suprimen (también en <package definition>s).

2. Las referencias de definición son sustituidas por <referenced definition>s (2.4.1.3).

3. Los gráficos se normalizan, es decir:

   1) Las decisiones que no son de terminación y las opciones de transición que no son de terminación se transforman respectivamente en decisiones de terminación y opciones de transición de terminación.

   2) Las acciones y/o el enunciado terminador que siguen a las decisiones y opciones de transición se desplazan para que aparezcan como <free action>s. Estas <free action>s generadas que no tienen etiqueta reciben etiquetas anónimas.

   3) Las listas de acciones (incluido el enunciado terminador que sigue) en las cuales la primera acción (si la hay, y en los demás casos el enunciado terminador siguiente) tienen una etiqueta, son sustituidas por una unión con la etiqueta y la lista de acciones aparece como <free action>.

4. El paquete Predefined se incluye en la <package list>.

5. Transformación de sistema genérico (4.3) y datos externos (5.4.6):

   1) Los identificadores en <simple expression>s contenidos en la <sdl specification> están vinculados a definiciones. Durante esa vinculación sólo se consideran las <data definition>s definidas en el paquete predefinido Predefined y las <external synonym definition>s (es decir que se pasan por alto todas las demás <data definition>s).

2) Los <external synonym>s son sustituidos por <synonym definition>s y el texto informal en opciones de transición es sustituido por <range condition>. La manera de hacerlo no es definida por el SDL.

3) Las <simple expression>s son evaluadas y se suprimen las <select definition>s, <option area>s, <transition option>s y <transition option area>s. Las <transition>s que no son seleccionadas en una <transition option> o <transition option area> reciben etiquetas anónimas y aparecen como <free action>s o <in-connector area> respectivamente.

   Los valores en <number of process instances> son representados por identificadores de literal entero totalmente calificados.

4) Las <external properties> son sustituidas por axiomas y texto informal. La manera de hacerlo no forma parte del SDL.

6. Inclusión de paquete (2.4.1.2)

Los <package>s que son mencionados en una <package reference clause>, pero que no forman parte de la <package list>, se incluyen en la <package list>. La manera de hacerlo no es definida por el SDL.

Si la <sdl specification> no contiene una <system definition> no se aplican otros pasos, aunque las propiedades de los paquetes garantizan que en la práctica pueden aplicarse otros pasos.

7. Transformación de:

– Asignaciones múltiples en <task body> (2.7.1);
– Decisión no determinista (2.7.5);
– <infix operator>s y sus operandos transformados en la forma prefijo (5.4.1.1);
– <procedure result> (2.4.6);
– Género de estructura (5.3.1.10);
– Lista de estados (2.6.3);
– Lista de estímulos (2.6.4);
– Primario de campo (5.3.2.5);
– Primario de estructura (5.3.2.6);
– Axiomas Booleanos (5.3.1.5);
– <syntype definition>s con **newtype** (5.3.1.9);
– Señales múltiples en <output body> (2.7.4);
– Temporizadores múltiples en <set> y <reset> (2.8).

8. Se insertan calificadores completos

De acuerdo con las reglas de visibilidad y las reglas de resolución por contexto (2.2.2), los calificadores se amplían para que denoten el trayecto completo.

NOTA – Como algunos conceptos adicionales (por ejemplo, transformaciones de generador y especialización) tienen repercusiones sobre el conjunto de definiciones adjuntas a una unidad de ámbito, los calificadores se derivan después de suprimir esos conceptos para la unidad de ámbito de que se trata.

Para cada unidad de ámbito:

– Parámetros de contexto (6.2);
– Especialización (6.3);
– Lista de señales (2.5.5);
– Ordenación (5.3.1.8);
– Generadores (5.3.1.12);
– Primario indizado (5.3.3.4);
– Variable de campo (5.4.3.2);
– Variable indizada (5.4.3.1);
– Entrada de campos (2.6.4);
– Diagrama de subestructura de bloque abierto (3.2.2);
– Valor de duración por defecto para conjunto de temporizadores (2.8);
– Inicialización de variables de géneros con inicialización por defecto (5.4.3.3);

se transforman, y después se insertan calificadores. A continuación las transformaciones se aplican para unidades de ámbito contenidas en la unidad de ámbito.

El paso se define por separado en 7.2.

9. Especialización implícita de procedimientos globales (2.7.3)

Si una ocurrencia de un <procedure identifier> refiere a un procedimiento que no está circundado por un proceso o un servicio, ésta hace que se efectúe una copia local del proceso y que se cambie el <procedure identifier> para denotar la copia. Este paso se repite hasta que ya no haya referencias a procedimientos no locales dentro de los procesos o servicios.

Las <operator definition>s se transforman en procedimientos con nombres anónimos y que tienen el resultado como parámetro **in/out**. La <procedure definition> resultante se desplaza a la unidad de ámbito circundante, se suprimen las signaturas de operador para cada operador y se transforman las aplicaciones de operador en <value returning procedure call>s.

10. Transformación de:

– <textual typebased system definition>s, <graphical typebased system definition>s, <textual typebased block definition>s, <graphical typebased block definition>s, <textual typebased process definition>s, <graphical typebased process definition>s, <textual typebased service definition>s y <graphical typebased service definition>s son sustituidas, respectivamente por <system definition>s, <system diagram>, <block definition>, <block diagram>, <process definition>, <process diagram>, <service definition> y <service diagram> copiando el contenido del tipo denotado por el <base type>. En esta copia los <formal context parameter>s son sustituidos por <identifier>s en <actual context parameters> como se indica en el subpaso 2.2 del paso 8 (véase a continuación).

– La palabra clave **this** en la llamada a procedimiento es sustituida por el identificador de procedimiento.

– Las ocurrencias de nombres de tipo dentro de calificadores son sustituidas por los nombres de instancia respectivos, y ocurrencias de **this** donde denota un <process identifier> son sustituidas por el identificador que denota <process definition> o <process diagram> circundantes.

– Las <channel connection>s, <channel to route connection>s, <signal route to route connection>s, el conjunto de señales completo y las rutas de señales omitidas (si las hay) son derivadas de las <gate>s que aparecen en canales y rutas de señales conectados a la instancia.

  Estas transformaciones se aplican a <process definition>s y <service definition>s.

– Las <gate>s en <output body>s son sustituidas por los canales o rutas de señales conectados a la instancia que transmiten la señal y mencionan la puerta.

– Se suprimen las <gate>s en canales y rutas de señales.

– Los conjuntos de bloques se suprimen y se modifica la **via**:

  Cada definición de canal que tiene un conjunto de bloques como punto extremo es sustituida por varios canales, uno para cada bloque en el conjunto. Cada nuevo canal está conectado a una copia de la definición de bloque definida por el conjunto de bloques. Esta copia tiene un nuevo nombre.

  Para **via** y las conexiones contenidas dentro de una copia de bloque, una ocurrencia del identificador de canal original es sustituida por el identificador de canal que conecta con ese miembro de bloque particular.

  Para **via** y las conexiones contenidas dentro de un bloque conectado a un conjunto de bloques, todas las ocurrencias del identificador de canal original son sustituidas por la lista de identificadores de canal resultante de la ampliación del conjunto de bloques.

  Si los dos puntos extremos del canal original denotan el mismo <block identifier>, el número de canales resultante es igual al cuadrado del número de instancias de bloque menos el número de instancias de bloque.

11. Supresión de:

    1) Tipos que tienen <formal context parameter>s, tipos de sistema, tipos de bloque, tipos de proceso, tipos de servicio, ocurrencias de <specialization> e <inheritance rule>.

    2) Procedimientos definidos fuera de procesos o servicios y <virtuality> para los procedimientos restantes.

12. Las definiciones dentro de paquetes (2.4.1.2) se desplazan al nivel de sistema.

    Cada nombre definido dentro de un paquete se redenomina con un nombre anónimo y los calificadores se cambian para que denoten el nivel de sistema.

    Se suprime la lista de paquetes.

13. Transformación de subestructura de canal (3.2.3).

14. Supresión de estado asterisco, entrada asterisco y conservación asterisco:

    – Un cuerpo que se origina en una definición de proceso, definición de servicio o definición de procedimiento sin especialización, sus estados asterisco son ampliados de acuerdo con el modelo definido en 4.4, seguido por la ampliación de la entrada asterisco de acuerdo con el modelo definido en 4.6, seguido por el modelo para conservación asterisco de acuerdo con el modelo definido en 4.7.

    – Un cuerpo que se forma combinando dos o varios cuerpos de tipo (mediante especialización) retiene información sobre el cuerpo de tipo a partir del cual se originan sus estados. Es decir, puede considerarse que consiste en una lista de cuerpos de tipo, en la cual el primer cuerpo de tipo se origina a partir del tipo que no tiene especialización, y el cuerpo de tipo N (N>1) en la lista se origina a partir del cuerpo de tipo de proceso que es una especialización del cuerpo de tipo N-1.

    – El cuerpo combinado se forma aplicando los pasos siguientes:

        1) Se incluyen todos los estados de todos los cuerpos, pero se excluyen transiciones y conservaciones asociadas, que son redefinidas o finalizadas.

        2) Se amplía el estado asterisco como para un cuerpo que se origina a partir de una definición sin especialización.

        3) Se amplía la entrada asterisco y la conservación asterisco como para un cuerpo que se origina a partir de una definición sin especialización.

        4) Se sustituyen transiciones y conservaciones virtuales y redefinidas con las transiciones y conservaciones redefinidas y finalizadas para los tipos una por una en orden ascendente de N. Si un estado es un estado asterisco, la sustitución se aplica a todos los estados en el cuerpo combinado.

    – Se fusiona la aparición múltiple de estado (4.5).

15. Se insertan transiciones implícitas (4.8).

16. Se cambia **via all** en una lista de salida (2.7.4).

17. Se suprimen <free action>s (2.6.7):

    1) Cualquier <join> o <out-connector area> son sustituidos, respectivamente, por <free action> o <in-connector area> que contiene ese nombre de conector, a menos que <join> o <out-connector area> estén contenidas en la misma <free action> o <in-connector area> que el nombre de conector correspondiente.

    2) A continuación se descartan las <free action>s e <in-connector area>s. Los gráficos resultantes ya no son acíclicos aparte de los bucles dentro de una transición (ninguna transición es compartida por entradas).

18. El estado siguiente indicado por guión es sustituido por un nombre de estado (4.9).

19. Se insertan comas finales en <stimulus> (2.6.4), <create body> (2.7.2), <procedure body> (2.7.3) y <output body>.

20. Los identificadores de sinónimos son sustituidos por la expresión que denotan (5.3.2.3).

21. Se transforman las entradas prioritarias (4.10).

22. Se transforma la señal continua (4.11).

23. Se transforma la condición habilitante (4.12).

24. Se insertan tareas implícitas para expresiones que contienen <now expression> (5.4.4.2), <import expression> (4.13), <view expression> (5.4.4.4), <timer active expression> (5.4.4.5) y llamadas a procedimiento implícitas para expresiones que contienen <value returning procedure call>s (5.4.5).

25. Se transforman valores importados y exportados (4.13) y procedimientos remotos (4.14).

7.2 Inserción de calificadores completos

En esta subcláusula se detalla el contenido del paso 8 (véase 7.1), calificadores completos:

La inserción de calificadores completos y la transformación de los conceptos transformados en esta subcláusula se hace paquete por paquete, (en el orden indicado por <package list>) y termina con <system definition>. La inserción de calificadores completos se efectúa aplicando las reglas de visibilidad y las reglas de resolución por contexto definidas en 2.2.2.

Los pasos 1 a 9 siguientes se repiten hasta que se han transformado todas las unidades de ámbito.

1. Se selecciona una unidad de ámbito (inicialmente el <package> más a la izquierda en <package list> o la <system definition>), que satisfacen las condiciones siguientes:

   1) La unidad de ámbito circundante (si la hay) ha sido transformada.

   2) Los <actual context parameter>s omitidos (6.1.2) están insertados.

   3) Si contiene <specialization>, se ha transformado cualquier unidad de ámbito contenida en ese tipo dado por <base type>.

   4) No ha sido transformada todavía.

2. Si la unidad de ámbito contiene <specialization>, se hace una copia del <base type> y se vincula cualquier <actual context parameter>s contenido en la <specialization>. La copia del tipo denotado por el <base type> se hace de tal modo que:

   1) Los tipos virtuales que también tienen una definición en la unidad de ámbito que ha de transformarse reciben nombres anónimos únicos, pero la unidad de ámbito mantiene el conocimiento de su calificador completo original de tal modo que los identificadores que denotan los tipos virtuales originales (es decir los identificadores calificados con el <base type> dentro de la unidad de ámbito y dentro de especializaciones de la unidad de ámbito) pueden ser sustituidos por la nueva identidad cuando se inserten calificadores (véase el paso 7 siguiente).

   2) Cualquier ocurrencia aplicada (salvo en generadores) de <formal context parameter>s definidos para ese tipo es sustituida por <actual context parameter>s correspondientes. Se cambian los calificadores de subseñales, literales y operadores de <signal context parameter>s y de <sort context parameter>s respectivamente, para denotar el identificador del <actual context parameter>. Los <formal context parameter>s correspondientes a <actual context parameters> omitidos no son sustituidos, pero se convierten en cambio en (partes de los) <formal context parameter>s de la unidad de ámbito.

   3) Las transiciones virtuales no se transforman en este paso, ya que existe una interdependencia con entrada asterisco, conservación asterisco y estado asterisco. En cambio, se inserta en el cuerpo combinado un conocimiento sobre el cuerpo de tipo del que se origina cualquier estado. Por consiguiente, el cuerpo combinado no puede considerarse válido hasta que se han transformado entrada asterisco, conservación asterisco y estado asterisco (véase el paso 14 de 7.1).

   4) Las ocurrencias del nombre de tipo en calificadores contenidos en el tipo son sustituidas por el nombre de la unidad de ámbito (salvo en generadores).

3. Transformación de <generator transformation>

1) se transforman <generator transformation>s dentro de <generator definition>s que aparecen en la unidad de ámbito.

2) se transforman <generator transformation>s dentro de <partial type definition>s que aparecen en la unidad de ámbito.

4. Se transforma <ordering> (5.3.1.8).

5. Se transforman <partial type definition>s:

1) Si una <partial type definition> contiene <type expression>, los literales y operadores se han copiado de acuerdo con la lista de herencia. Si el <base type> para una <partial type definition> tiene <sort context parameter>s formales, se cambia como corresponde cualquier género de argumento o género de resultado que menciona un <sort context parameter>.

2) Se derivan las propiedades de igualdad (5.3.1.4).

3) Se insertan calificadores de identificadores en <axioms>, en las <range condition>s y en la expresión por defecto, y se inserta una cuantificación explícita. Se efectúa la resolución por contexto descartando cualquier <partial type definition>s que tenga <formal context parameter>s (salvo para la circundante, si la hay).

4) Para cualquier <partial type definition> que contiene una <type expression> con <actual context parameter>s, los <axioms>, la <default initialization> y las <range condition>s contenidos en el <base type> se copian a la <partial type definition>, donde <formal context parameter>s han sido sustituidos por los <actual context parameter>s correspondientes.

5) Para cualquier <partial type definition> que contiene una <type expression> sin <actual context parameter>s, se insertan los axiomas implícitos (5.3.1.11).

6. Se suprimen <signal definition>s con <specialization>:

Se selecciona una definición de señal que tiene <specialization> y donde el <base type> denota una <signal definition> sin <specialization>. El modelo para <specialization> para esta <signal definition> se aplica de manera similar a la indicada en el paso 2 anterior. Este paso se repite hasta que no queden <signal definition>s con <specialization> en la unidad de ámbito.

7. Se insertan cualificadores en los identificadores directamente contenidos en la unidad de ámbito y en los contenidos en las definiciones de señal contenidas.

La resolución por contexto se hace descartando cualquier <partial type definition> que tiene <formal context parameter>s.

Las referencias a un tipo virtual en un supertipo de una unidad de ámbito circundante directamente contenidas en la unidad de ámbito, se transforman en referencias a la copia anónima (véase el paso 2.1 anterior).

8. Se transforman <stimulus> que contienen <indexed variable>s y <field variable>s (2.6.4).

9. Transformación de:

– Identificadores de lista de señales en una lista de identificadores de señal (2.5.5)

– Primario indizado (5.3.2.4)

– Variable de campo (5.4.3.2)

– Variable indizada (5.4.3.1)

– <return> con <expression> (2.4.6).

Reemplazada por una versión más reciente

Anexo A

Índice
de las cláusulas 2 a 7 de la Recomendación Z.100 (partes normativas)

(Este anexo es parte integrante de la presente Recomendación)

Las entradas son las siguientes:

a) <palabras clave> de la gramática concreta.

b) No-terminales de la gramática concreta. Están entre paréntesis angulares (es decir < y >). Las páginas donde figuran definiciones se indican en negritas.

c) No-terminales de la gramática abstracta. Las páginas donde figuran definiciones se indican en negritas.

d) Palabras del glosario, señaladas con una (G). En esta versión sólo figuran las palabras del glosario relativas a las cláusulas 2 a 7.

Si en la misma página figuran una definición y una aparición de un no-terminale, la entrada en negritas tiene precedencia. Los no-terminales procedentes de la gramática concreta que no tienen una entrada en negritas:

1) no son parte de las reglas sintácticas;

2) contienen partes subrayadas (el subrayado no se muestra en el índice).

Los no-terminales que contienen partes subrayadas deberían ser vistos ignorando las partes subrayadas para el caso de una aparición con definición.

<basic input part>; **54**; 55	<parte de entrada básica>
<basic save area>; **55**; 56	<área de conservación básica>
<basic save part>; **55**; 56	<parte de conservación básica>
<block area>; **29**; 43; 92; 99	<área de bloque>
<block definition>; 18; 27; 28; **30**; 31; 32; 34; 43; 46; 47; 84; 87; 88; 96; 98; 118; 173; 177; 196	<definición de bloque>
<block diagram>; 18; 27; 29; **31**; 32; 85; 196	<diagrama de bloque>
<block heading>; **31**	<encabezamiento de bloque>
<block identifier>; 30; 31; 43; 87; 88; 173; 196	<identificador de bloque>
<block interaction area>; **29**; 85; 87; 92; 96; 166	<área de interacción de bloques>
<block name>; 28; 29; 30; 31; 168; 173	<nombre de bloque>
<block substructure area>; 31; **85**; 92; 168	<área de subestructura de bloque>
<block substructure definition>; 18; 27; 30; **84**; 85; 86; 88; 118; 167; 176; 177; 183	<definición de subestructura de bloque>
<block substructure diagram>; 18; 27; 43; **85**; 86	<diagrama de subestructura de bloque>
<block substructure heading>; **85**	<encabezamiento de subestructura de bloque>
<block substructure identifier>; 84; 85	<identificador de subestructura de bloque>
<block substructure name>; 84; 85	<nombre de subestructura de bloque>
<block substructure symbol>; **85**	<símbolo de subestructura de bloque>
<block substructure text area>; **85**; 92; 99	<área de texto de subestructura de bloque>
<block symbol>; **29**; 85; 88; 173; 176	<símbolo de bloque>
<block text area>; **31**; 92; 99; 168	<área de texto de bloque>
<block type definition>; 18; 24; 27; 28; 31; 32; 43; 46; 47; 84; 85; 98; 118; **167**; 168	<definición de tipo de bloque>
<block type diagram>; 18; 27; 46; 85; 99; 167; **168**	<diagrama de tipo de bloque>
<block type expression>; 173	<expresión tipo de bloque>
<block type heading>; **168**	<encabezamiento de tipo de bloque>
<block type identifier>; 167; 168	<identificador de tipo de bloque>
<block type name>; 167; 168	<nombre de tipo de bloque>
<block type reference>; 92; 99; 167; **168**	<referencia de tipo de bloque>
<block type symbol>; 167; **168**	<símbolo de tipo de bloque>
<body>; 19; **57**; 60; 102; 103; 183	<cuerpo>
<Boolean active expression>; 156	<expresión activa booleana>
<Boolean axiom>; 122; **129**	<axioma booleano>
<Boolean expression>; 105; 106; 107; 108; 156; 163	<expresión booleana>
<Boolean ground expression>; 153	<expresión fundamental booleana>
<Boolean simple expression>; 98; 99	<expresión simple booleana>
<Boolean term>; 129; 130	<término booleano>
<channel connection>; **84**; 85; 88; 90; 98; 177; 196	<conexión de canal>
<channel definition area>; 29; **43**; 92; 99	<área de definición de canal>
<channel definition>; 28; **43**; 84; 87; 88; 98	<definición de canal>
<channel endpoint connection>; **87**; 88; 90; 98	<conexión de punto extremo de canal>
<channel endpoint>; **43**	<punto extremo de canal>
<channel identifier>; 48; 66; 88	<identificador de canal>
<channel identifiers>; 31; 43; 46; **48**; 84; 85; 177	<identificadores de canal>
<channel name>; 43; 87	<nombre de canal>
<channel path>; **43**; 177	<trayecto de canal>
<channel substructure area>; **87**; 92	<área de subestructura de canal>
<channel substructure association area>; 43; **87**; 93	<área de asociación de subestructura de canal>
<channel substructure definition>; 18; 27; 43; **87**; 88; 118	<definición de subestructura de canal>
<channel substructure diagram>; 18; 27; 43; **87**; 88	<diagrama de subestructura de canal>
<channel substructure heading>; **87**	<encabezamiento de subestructura de canal>
<channel substructure identifier>; 87	<identificador de subestructura de canal>
<channel substructure name>; 87	<nombre de subestructura de canal>
<channel substructure symbol>; 87; **88**	<símbolo de subestructura de canal>
<channel substructure text area>; **87**; 93; 99	<área de texto de subestructura de canal>
<channel symbol 1>; **43**	<símbolo de canal 1>
<channel symbol 2>; **43**	<símbolo de canal 2>
<channel symbol 3>; 43; **44**	<símbolo de canal 3>
<channel symbol 4>; 43; **44**	<símbolo de canal 4>
<channel symbol 5>; 43; **44**	<símbolo de canal 5>
<channel symbol>; 20; **43**; 44; 85; 87; 92; 95	<símbolo de canal>
<channel to route connection>; 30; 47; **48**; 98; 177; 196	<conexión de canal a ruta>
<character string literal identifier>; 126; **127**	<identificador de literal cadena de caracteres>
<character string literal>; 126; **127**; 128; 137; 143; 144; 146	<literal cadena de caracteres>

<character string>; 14; **15**; 16; 17; 19; 20; 21; 22; 68; 95; 127; 128	<cadena de caracteres>

<closed range>; **135**	<intervalo cerrado>
<comment area>; **21**; 93	<área de comentario>
<comment symbol>; 20; 21; **22**	<símbolo de comentario>
<comment>; **21**	<comentario>
<composite special>; 14; **15**	<especial compuesto>
<composite term list>; **122**; 123; 125	<lista de términos compuestos>
<composite term>; **122**; 125	<término compuesto>
<condition>; **130**	<condición>
<conditional composite term>; 125; **130**	<término compuesto condicional>
<conditional equation>; 122; 123; **124**	<ecuación condicional>
<conditional expression>; 154; **156**	<expresión condicional>
<conditional ground expression>; 149; **153**; 156	<expresión fundamental condicional>
<conditional ground term>; 126; **130**	<término fundamental condicional>
<conditional term>; **130**	<término condicional>
<connector name>; 19; 57; 60; 183	<nombre de conector>
<consequence expression>; **156**	<expresión consecuencia>
<consequence ground expression>; **153**	<expresión fundamental de consecuencia>
<consequence>; **130**	<consecuencia>
<constant>; 70; **135**	<constante>
<context parameters end>; 177; **178**	<fin de parámetros de contexto>
<context parameters start>; 177; **178**	<principio de parámetros de contexto>
<continuous signal area>; 93; **105**; 106; 163	<área de señal continua>
<continuous signal association area>; 53; 93; **105**	<área de asociación de señal continua>
<continuous signal>; 52; **105**; 106; 108; 163	<señal continua>
<create body>; **63**; 197	<cuerpo de crear>
<create line area>; **31**; 93; 99	<área de línea de crear>
<create line symbol>; 20; **31**; 32; 92; 95	<símbolo de línea de crear>
<create request area>; 59; **63**; 93	<área de petición de crear>
<create request symbol>; **63**	<símbolo de petición de crear>
<create request>; 58; **63**	<petición de crear>
<dash nextstate>; 59; **104**	<estado siguiente indicado por guión>
<dashed association symbol>; 20; 21; **22**; 87; 92; 95	<símbolo de asociación de trazo discontinuo>
<dashed block symbol>; **173**	<símbolo de bloque de trazo discontinuo>
<dashed process symbol>; **174**	<símbolo de proceso de trazo discontinuo>
<dashed service symbol>; 174; **175**	<símbolo de servicio de trazo discontinuo>
<data definition>; 17; 24; 28; 29; 30; 33; 35; 37; 38; 40; 98; 147; 148; **153**; 194	<definición de datos>
<decimal digit>; **14**	<dígito decimal>
<decision area>; 59; **69**; 93	<área de decisión>
<decision body>; **68**; 69	<cuerpo de decisión>
<decision symbol>; **69**	<símbolo de decisión>
<decision>; 58; **68**; 69; 106; 108	<decisión>
<default initialization>; 51; 117; 118; 133; 138; 140; **159**; 199	<inicialización por defecto>
<definition selection list>; **24**; 25; 26	<lista de selección de definición>
<definition selection>; **24**; 25; 26	<selección de definición>
<definition>; 26; **27**	<definición>
<destination>; **66**; 110; 111; 112; 114	<destino>
<diagram identifier>; 21	<identificador de diagrama>
<diagram in package>; **25**	<diagrama en paquete>
<diagram kind>; 21	<clase de diagrama>
<diagram name>; 21	<nombre de diagrama>
<diagram>; 22; 26; **27**	<diagrama>
<dummy inlet symbol>; 94; **95**; 96	<símbolo de acceso de entrada ficticio>
<dummy outlet symbol>; **92**; 94; 95; 96	<símbolo de acceso de salida ficticio>
<Duration ground expression>; 70; 71	<expresión fundamental de duración>
<else part>; **68**; 69; 100; 101	<parte sí no>
<enabling condition area>; 54; 56; 93; **107**; 113; 163	<área de condición habilitante>
<enabling condition symbol>; 105; **107**	<símbolo de condición habilitante>
<enabling condition>; 54; 56; **107**; 108; 112; 163	<condición habilitante>
<end>; **21**; 24; 28; 30; 31; 33; 35; 37; 39; 43; 45; 48; 49; 50; 51; 52; 54; 55; 56; 57; 58; 68; 70; 84; 87; 91; 95; 98; 100; 104; 105; 107; 110; 112; 113; 119; 122; 136; 137; 141; 144; 147; 153; 159; 164; 166; 167; 168; 169; 170; 172; 173; 174; 175; 177; 179	<fin>
<endpoint constraint>; **176**	<limitación de punto extremo>
<entity in block>; **30**; 167	<entidad en bloque>

<entity in package>; **24**; 25	<entidad en paquete>
<entity in procedure>; 39; **40**	<entidad en procedimiento>
<entity in process>; **33**; 168	<entidad en proceso>
<entity in service>; **37**; 170	<entidad en servicio>
<entity in system>; **28**; 84; 87; 166	<entidad en sistema>
<entity kind>; **24**; 25; 26	<clase de entidad>
<equation>; **122**; 145	<ecuación>
<error term>; 122; **131**	<término de error>
<exclamation>; 19; **126**; 131; 140	<admiración>
<existing gate symbol>; **176**; 183	<símbolo de puerta existente>
<existing typebased block definition>; 29; 93; **173**	<definición de bloque basada en tipo existente>
<existing typebased process definition>; 31; **174**	<definición de proceso basada en tipo existente>
<existing typebased service definition>; 35; **174**; 175	<definición de servicio basada en tipo existente>
<export area>; 59; 93; **110**	<área de exportación>
<export>; 58; **110**; 111; 112	<exportación>
<exported as>; **50**	<exportado como>
<exported variable definition>; 110	<definición de variable exportada>
<expression list>; 70; 151; 152; **154**; 157; 158; 162; 163	<lista de expresiones>
<expression>; 42; 61; 63; 64; 65; 67; 101; 127; 148; **149**; 150; 154; 156; 157; 158; 159; 164; 199	<expresión>
<extended composite term>; 122; **125**; 126	<término compuesto ampliado>
<extended ground term>; 122; **126**	<término fundamental ampliado>
<extended literal identifier>; 122; **126**	<identificador de literal ampliado>
<extended literal name>; 119; 120; **126**	<nombre de literal ampliado>
<extended operator identifier>; 125; **126**	<identificador de operador ampliado>
<extended operator name>; 119; 120; 123; **126**	<nombre de operador ampliado>
<extended primary>; **149**; 150; 154	<primario ampliado>
<extended properties>; 117; 118; **125**; 133	<propiedades ampliadas>
<extended sort>; 119; **120**; 122; 140; 141; 145; 147	<género ampliado>
<external data description>; **164**	<descripción de datos externos>
<external formalism name>; **164**	<nombre de formalismo externo>
<external properties>; 117; 118; **164**; 195	<propiedades externas>
<external signal route identifiers>; 46; **48**; 177	<identificadores de ruta de señales externos>
<external synonym definition item>; **97**	<ítem de definición de sinónimo externo>
<external synonym definition>; **97**; 142; 194	<definición de sinónimo externo>
<external synonym identifier>; 97	<identificador de sinónimo externo>
<external synonym name>; 97	<nombre de sinónimo externo>
<external synonym>; **97**; 150; 195	<sinónimo externo>
<field extract operator name>; 152	<nombre de operador de extracto de campo>
<field list>; **136**	<lista de campos>
<field modify operator name>; 159	<nombre de operador de modificación de campo>
<field name>; 136; 151; 152; 158; 159	<nombre de campo>
<field primary>; 149; 150; **151**	<primario de campo>
<field selection>; **151**; 158	<selección de campo>
<field sort>; **136**	<género de campo>
<field variable>; 55; 157; **158**; 159; 199	<variable de campo>
<fields>; **136**	<campos>
<first constant>; 135	<primera constante>
<first field name>; 152	<primer nombre de campo>
<flow line symbol>; 20; 57; **60**; 69; 92; 95; 101	<símbolo de línea de flujo>
<formal context parameter>; 19; 49; 118; 166; 167; 169; 170; 172; 177; **178**; 179; 196; 197; 198; 199	<parámetro de contexto formal>
<formal context parameters>; 39; 41; 49; 117; 118; 166; 167; 168; 169; 170; **177**; 178	<parámetros de contexto formal>
<formal name>; **91**; 96	<nombre formal>
<formal parameters signature>; **179**	<signatura de parámetros formal>
<formal parameters>; 19; **33**; 35; 147; 148; 168; 169	<parámetro formal>
<formal variable parameters>; 19; **39**	<parámetro de variable formal>
<frame symbol>; 20; 21; 25; **29**; 31; 34; 38; 40; 43; 46; 85; 87; 91; 92; 96; 147; 166; 168; 169; 170; 172; 176	<símbolo de recuadro>
<free action>; 33; 40; **57**; 147; 194; 195; 197	<acción libre>
<full stop>; **14**; 15; 128	<punto>
<gate constraint>; **175**; 176	<limitación de puerta>
<gate definition>; 19; 167; 168; 170; **175**	<definición de puerta>
<gate identifier>; 66	<identificador de puerta>
<gate name>; 19; 175; 176	<nombre de puerta>
<gate symbol 1>; **176**	<símbolo de puerta 1>

<gate symbol 2>; **176**	<símbolo de puerta 2>
<gate symbol>; **176**	<símbolo de puerta>
<gate>; 43; 45; 46; 168; 169; 170; 173; 174; **175**; 176; 177; 196	<puerta>
<generator actual list>; **141**	<lista de generadores reales>
<generator actual>; **141**	<generador real>
<generator definition>; **139**; 140; 153; 199	<definición de generador>
<generator formal name>; 19; 126; **139**; 140; 141	<nombre de generador formal>
<generator identifier>; 141	<identificador de generador>
<generator name>; 139; 140	<nombre de generador>
<generator parameter list>; **139**	<lista de parámetros de generador>
<generator parameter>; **139**; 140; 141	<parámetro de generador>
<generator sort>; 120; **140**	<género de generador>
<generator text>; 126; **139**; 140; 141	<texto de generador>
<generator transformation>; **141**; 142; 198; 199	<transformación de generador>
<generator transformations>; 17; 125; 139; **141**	<transformaciones de generadores>
<graphical answer part>; **69**	<parte de respuesta gráfica>
<graphical answer>; **69**; 101	<respuesta gráfica>
<graphical block reference>; **29**; 93	<referencia gráfica de bloque>
<graphical block substructure reference>; **85**	<referencia gráfica de subestructura de bloque>
<graphical channel substructure reference>; **87**	<referencia gráfica de subestructura de canal>
<graphical else part>; **69**	<parte otro caso gráfica>
<graphical gate constraint>; 43; 46; 168; 169; 170; **176**	<limitación gráfica de puerta>
<graphical procedure reference>; 38; **40**; 93; 167; 168; 169; 170	<referencia gráfica de procedimiento>
<graphical process reference>; **31**; 34; 93	<referencia gráfica de proceso>
<graphical service reference>; **35**	<referencia gráfica de servicio>
<graphical typebased block definition>; 29; 93; **73**; 196	<definición gráfica de bloque basada en tipo>
<graphical typebased process definition>; 31; **174**; 196	<definición gráfica de proceso basada en tipo>
<graphical typebased service definition>; 35; **174**; 175; 196	<definición gráfica de servicio basada en tipo>
<graphical typebased system definition>; 29; **172**; 196	<definición gráfica de sistema basada en tipos>
<ground expression list>; 149; **150**	<lista de expresiones fundamentales>
<ground expression>; 50; 51; 97; 101; 135; 142; 147; **149**; 150; 153; 157; 159	<expresión fundamental>
<ground primary>; **149**; 150	<primario fundamental>
<ground term>; **122**; 126; 141; 145	<término fundamental>
<heading area>; 20; **21**	<área de encabezamiento>
<heading>; **21**	<encabezamiento>
<hyphen>; **104**	<guión>
<identifier>; **17**; 18; 19; 26; 27; 123; 147; 150; 171; 175; 176; 178; 184; 196	<identificador>
<imperative operator>; 147; 154; **160**	<operador imperativo>
<implicit text symbol>; 21	<símbolo de texto implícito>
<import expression>; **110**; 111; 160; 161; 198	<expresión importación>
<import identifier>; 111	<identificador de importación>
<imported procedure specification>; 33; 35; 37; 38; 98; **112**; 113	<especificación de procedimiento importado>
<imported variable specification>; 33; 35; 37; 38; 98; **110**; 111	<especificación de variable importada>
<in-connector area>; 35; 41; **57**; 60; 93; 147; 169; 195; 197	<área de conector de entrada>
<in-connector symbol>; **57**; 60	<símbolo de conector de entrada>
<indexed primary>; 149; 150; **151**	<primario indizado>
<indexed variable>; 55; 157; **158**; 199	<variable indizada>
<infix operator>; 125; 126; **127**; 195	<operador infijo>
<informal text>; 16; **20**; 62; 68; 97; 100; 122	<texto informal>
<inheritance list>; 126; 129; **137**; 138; 182	<lista de herencia>
<inheritance rule>; 125; 129; **137**; 178; 197	<regla de herencia>
<inherited operator name>; **137**; 138	<nombre de operador heredado>
<inherited operator>; **137**	<operador heredado>
<initial number>; **33**; 34	<número inicial>
<inlet symbol>; 94; **95**; 96	<símbolo de entrada>
<input area>; 53; **54**; 93	<área de entrada>
<input association area>; **53**; 93	<área de asociación de entrada>
<input list>; **54**; 103; 105; 107	<lista de entradas>
<input part>; 52; **54**; 55; 103; 104; 106; 107; 112; 114	<parte de entrada>

<input symbol>; 20; **54**; 56; 113	<símbolo de entrada>
<Integer literal name>; 105; 106	<símbolo de entrada entero>
<interface>; 24; **25**; 26	<interfaz>
<internal input symbol>; 20; 54; **71**	<símbolo de entrada interno>

<internal output symbol>; 20; 66; **72**	<símbolo de salida interno>
<internal properties>; **117**	<propiedades internas>
<internal synonym definition>; **142**	<definición de sinónimo interno>
<join>; 57; 58; **60**; 69; 101; 197	<unión>
<kernel heading>; **21**	<encabezamiento de núcleo>
<keyword>; 14; **15**; 17	<palabra clave>
<label>; **57**; 58	<etiqueta>
<left curly bracket>; **14**	<llave izquierda>
<left square bracket>; **14**	<corchete izquierdo>
<letter>; **14**; 16	<letra>
<lexical unit>; **14**; 16; 91; 95	<unidad léxica>
<literal axioms>; 144; **145**	<axioma de literales>
<literal equation>; 129; **144**; 145; 146	<ecuación de literales>
<literal identifier>; **122**; 147; 149	<identificador de literal>
<literal list>; **119**; 132	<lista de literales>
<literal mapping>; 117; 118; 141; **144**; 145; 146	<correspondencia de literales>
<literal name>; 21	<nombre de literal>
<literal operator identifier>; 120; 122; 123; 145; 146	<identificador de operador literal>
<literal operator name>; 119; 120; 137; 146	<nombre de operador literal>
<literal quantification>; 144; **145**; 146	<cuantificación de literal>
<literal rename list>; **137**; 138	<lista de redenominaciones de literal>
<literal rename pair>; **137**	<par de redenominación de literal>
<literal rename signature>; **137**; 138	<signatura de redenominación de literal>
<literal renaming>; **137**; 182	<redenominación de literal>
<literal signature>; 19; **119**; 120; 132; 141; 146	<signatura de literal>
<macro actual parameter>; 91; 94; **95**; 96	<parámetro real de macro>
<macro body area>; 91; **92**	<área de cuerpo de macro>
<macro body port1>; **92**; 96	<puerto de cuerpo de macro1>
<macro body port2>; **92**; 96	<puerto de cuerpo de macro2>
<macro body>; 16; **91**; 94	<cuerpo de macro>
<macro call area>; 93; **95**; 96; 194	<área de llamada a macro>
<macro call body>; **95**	<cuerpo de llamada a macro>
<macro call port1>; **95**; 96	<puerto de llamada a macro1>
<macro call port2>; **95**; 96	<puerto de llamada a macro2>
<macro call symbol>; **95**; 96	<símbolo de llamada a macro>
<macro call>; **95**; 96; 194	<llamada a macro>
<macro definition>; 24; 27; 28; 29; 30; 33; 35; 37; 38; 39; 40; **91**; 94; 98; 147; 194	<definición de macro>
<macro diagram>; 25; 27; 29; 31; 34; 38; 40; 85; 87; **91**; 94; 95; 96; 99; 147; 166; 168; 169; 170; 194	<diagrama de macro>
<macro formal name>; 19	<nombre formal de macro>
<macro formal parameter>; **91**; 94; 95; 96	<parámetro formal de macro>
<macro formal parameters>; **91**; 92	<parámetros formales de macro>
<macro heading>; 91; **92**	<encabezamiento de macro>
<macro inlet symbol>; **92**; 96	<símbolo de acceso de entrada de macro>
<macro label>; **92**; 94; 95; 96	<etiqueta de macro>
<macro name>; 19; 91; 92; 94; 95; 96	<nombre de macro>
<macro outlet symbol>; **92**; 96	<símbolo de acceso de salida de macro>
<macro parameter>; **91**	<parámetro de macro>
<maximum number>; **33**; 34	<número máximo>
<merge area>; 59; **60**; 93	<área de fusión>
<merge symbol>; **60**	<símbolo de fusión>
<monadic operator>; 125; 126; **127**	<operador monádico>
<name class literal>; 126; 141; **143**	<literal de clase de nombre>
<name>; 16; **17**; 18; 19; 24; 26; 27; 85; 91; 92; 94; 111; 120; 123	<nombre>
<national>; **14**; 16	<nacional>
<Natural literal name>; 21; 143; 144	<nombre de literal natural>
<Natural simple expression>; 33; 173	<expresión simple natural>
<nextstate area>; 20; 53; **59**; 93	<área de estado siguiente>
<nextstate body>; **59**	<cuerpo de estado siguiente>
<nextstate>; 58; **59**; 104; 105; 106; 107; 108	<estado siguiente>
<noequality>; 119; 120; **129**	<no igualdad>
<note>; 14; **15**; 16; 21	<nota>
<now expression>; **160**; 161; 198	<expresión ahora>
<number of block instances>; 44; **173**	<número de instancias de bloque>

Reemplazada por una versión más reciente

<priority input>; 52; **104**; 105	<entrada prioritaria>
<procedure area>; **40**; 93; 99	<área de procedimiento>
<procedure body>; 39; **40**; 42; 169; 197	<cuerpo de procedimiento>
<procedure call area>; 59; **64**; 65; 93	<área de llamada a procedimiento>
<procedure call body>; **64**	<cuerpo de llamada a procedimiento>
<procedure call symbol>; **64**; 113	<símbolo de llamada a procedimiento>
<procedure call>; 41; 58; **64**; 163; 164	<llamada a procedimiento>
<procedure constraint>; **179**	<limitación de procedimiento>
<procedure context parameter>; 178; **179**	<parámetro de contexto de procedimiento>
<procedure definition>; 18; 24; 27; 28; 31; 33; 37; **39**; 40; 98; 118; 148; 196	<definición de procedimiento>
<procedure diagram>; 18; 27; 38; **40**; 41; 148; 167; 168; 169; 170	<diagrama de procedimiento>
<procedure formal parameter constraint>; **179**	<limitación de parámetro de procedimiento formal>
<procedure formal parameters>; **39**; 41; 42; 164	<parámetro de procedimiento formal>
<procedure graph area>; 40; **41**; 42; 57; 93	<área de gráfico de procedimiento>
<procedure heading>; 40; **41**	<encabezamiento de procedimiento>
<procedure identifier>; 39; 41; 64; 65; 164; 179; 180; 196	<identificador de procedimiento>
<procedure name>; 33; 39; 40; 41; 179; 180	<nombre de procedimiento>
<procedure preamble>; 33; **39**; 40; 41; 184	<preámbulo de procedimiento>
<procedure result>; 18; 39; **40**; 41; 42; 61; 164; 195	<resultado de procedimiento>
<procedure signature>; 112; 134; **179**; 180	<signatura de procedimiento>
<procedure start area>; **41**; 42; 93	<área de arranque de procedimiento>
<procedure start symbol>; **41**; 147	<símbolo de arranque de procedimiento>
<procedure symbol>; 40; **41**	<símbolo de procedimiento>
<procedure text area>; **40**; 93; 99	<área de texto de procedimiento>
<process area>; **31**; 32; 46; 93; 99	<área de proceso>
<process body>; **33**; 34; 37	<cuerpo de proceso>
<process constraint>; 17; **179**	<limitación de proceso>
<process context parameter>; 17; 166; 167; 178; **179**	<parámetro de contexto de proceso>
<process definition>; 18; 27; 30; 32; **33**; 34; 37; 46; 47; 78; 98; 102; 103; 118; 174; 177; 196	<definición de proceso>
<process diagram>; 18; 27; 31; **34**; 35; 196	<diagrama de proceso>
<process graph area>; 34; **35**; 38; 57; 60; 93; 99	<área de gráfico de proceso>
<process heading>; 34; **35**	<encabezamiento de proceso>
<process identifier>; 17; 33; 35; 45; 63; 64; 66; 67; 174; 179; 196	<identificador de proceso>
<process interaction area>; **31**; 93; 168	<área de interacción de proceso>
<process name>; 17; 31; 33; 35; 173; 179	<nombre de proceso>
<process signature>; **179**	<signatura de proceso>
<process symbol>; **31**; 174; 176	<símbolo de proceso>
<process text area>; 34; **35**; 93; 99; 169	<área de texto de proceso>
<process type body>; 60; 168; **169**; 170	<cuerpo de tipo de proceso>
<process type definition>; 18; 24; 27; 28; 30; 32; 46; 47; 63; 66; 98; 118; **168**; 169	<definición de tipo de proceso>
<process type diagram>; 18; 27; 46; 99; 167; 168; **169**	<diagrama de tipo de proceso>
<process type expression>; 173; 174	<expresión tipo de proceso>
<process type graph area>; 60; 93; 99; **169**	<área de gráfico de tipo de proceso>
<process type heading>; **169**	<encabezamiento de tipo de proceso>
<process type identifier>; 168; 169	<identificador de tipo de proceso>
<process type name>; 168; 169	<nombre de tipo de proceso>
<process type reference>; 93; 99; 167; 168; **169**	<referencia de tipo de proceso>
<process type symbol>; **169**	<símbolo de tipo de proceso>
<properties expression>; **117**; 118; 126; 133; 139; 140; 141; 145	<expresión propiedades>
<qualifier>; **17**; 18; 19; 27; 120; 126; 127; 140; 150; 152	<calificador>
<quantification>; **122**; 123	<cuantificación>
<quantified equations>; **122**; 123	<ecuación cuantificada>
<question expression>; 68	<expresión pregunta>
<question>; **68**; 69; 106; 108	<pregunta>
<quote>; **127**	<comillas>
<quoted operator>; 17; 18; 19; 126; **127**; 150	<operador entre comillas>
<range condition>; 68; 70; 133; 134; **135**; 195; 199	<condición de intervalo>
<referenced definition>; 23; 24; **26**; 27; 194	<definición referenciada>
<regular element>; **143**; 144	<elemento regular>
<regular expression>; **143**; 144	<expresión regular>

<regular interval>; 128; **143**; 144	<intervalo regular>

<remote procedure call area>; 59; 93; **113**	<área de llamada a procedimiento remoto>

<remote procedure call body>; **112**; 113	<cuerpo de llamada a procedimiento remoto>

<remote procedure call>; 58; **112**; 113; 114; 163	<llamada a procedimiento remoto>
<remote procedure context parameter>; 178; **180**	<parámetro de contexto de procedimiento remoto>
<remote procedure definition>; 24; 28; 29; 31; 33; 35; 98; **112**; 113; 180	<definición de procedimiento remoto>
<remote procedure identifier list>; 107; **112**; 113	<lista de identificadores de procedimiento remoto>
<remote procedure identifier>; 39; 40; 112; 113	<identificador de procedimiento remoto>
<remote procedure input area>; 54; 93; **113**; 114	<área de entrada de procedimiento remoto>
<remote procedure input transition>; 54; **112**; 113; 114	<transición de entrada de procedimiento remoto>
<remote procedure name>; 37; 112; 113	<nombre de procedimiento remoto>
<remote procedure save area>; 55; 93; **113**; 114	<área de conservación de procedimiento remoto>
<remote procedure save>; 55; 107; **112**; 113; 114	<conservación de procedimiento remoto>
<remote variable context parameter>; 178; **181**	<parámetro de contexto de variable remota>
<remote variable definition>; 24; 28; 29; 30; 33; 35; 98; **110**; 111; 181	<definición de variable remota>
<remote variable identifier>; 50; 110	<identificador de variable remota>
<remote variable name>; 37; 110; 181	<nombre de variable remota>
<reset area>; 59; **71**; 93	<área de reinicializar>
<reset statement>; **70**; 71	<enunciado de reinicializar>
<reset>; 58; **70**; 71; 195	<reinicializar>
<restricted equation>; **124**	<ecuación restringida>
<restriction>; **124**	<restricción>
<result>; 19; 119; **120**; 121; 136; 147; 150	<resultado>
<return area>; 42; 59; **61**; 93	<área de retorno>
<return symbol>; **61**	<símbolo de retorno>
<return>; 42; 58; **61**; 199	<retorno>
<reverse>; **89**; 180	<inverso>
<right curly bracket>; **14**	<llave derecha>
<right square bracket>; **14**	<corchete derecho>
<save area>; 53; **55**; 93	<área de conservación>
<save association area>; **53**; 93	<área de asociación de conservación>
<save list>; **55**; 103; 107	<lista de conservaciones>
<save part>; 52; **55**; 107; 114	<parte conservación>
<save symbol>; 55; **56**; 113	<símbolo de conservación>
<scope unit kind>; **17**; 18; 117; 118	<clase de unidad de ámbito>
<sdl specification>; **23**; 25; 26; 91; 194; 195	<especificación sdl>
<second constant>; 135	<segunda constante>
<select definition>; 24; 28; 29; 30; 33; 35; 37; 38; 39; 40; **98**; 99; 147; 148; 195	<definición de seleccionar>
<service area>; **35**; 46; 93; 99	<área de servicio>
<service body>; **37**	<cuerpo de servicio>
<service definition>; 18; 27; 33; 34; **37**; 38; 47; 78; 98; 103; 118; 175; 196	<definición de servicio>
<service diagram>; 18; 27; 35; **38**; 196	<diagrama de servicio>
<service graph area>; **38**; 93; 170	<área de gráfica de servicio>
<service heading>; **38**	<encabezamiento de servicio>
<service identifier>; 37; 38; 45; 174	<identificador de servicio>
<service interaction area>; 34; **35**; 93; 169	<área de interacción de servicios>
<service name>; 33; 35; 37; 38; 174	<nombre de servicio>
<service symbol>; **35**; 174; 176	<símbolo de servicio>
<service text area>; **38**; 93; 99; 170	<área de texto de servicio>
<service type body>; **170**	<cuerpo de tipo de servicio>
<service type definition>; 18; 24; 27; 28; 31; 33; 98; 118; **170**; 171	<definición de tipo de servicio>
<service type diagram>; 18; 27; 99; 167; 168; 169; **170**	<diagrama de tipo de servicio>
<service type expression>; 174; 175	<expresión tipo de servicio>
<service type heading>; **170**	<encabezamiento de tipo de servicio>
<service type identifier>; 170	<identificador de tipo de servicio>
<service type name>; 170	<nombre de tipo de servicio>
<service type reference>; 93; 99; 167; 168; 169; **170**	<referencia de tipo de servicio>
<service type symbol>; **170**	<símbolo de tipo de servicio>
<set area>; 59; **70**; 93	<área de inicializar>
<set statement>; **70**; 71	<enunciado de inicializar>
<set>; 58; **70**; 71; 195	<inicializar>
<signal constraint>; **180**	<limitación de señal>
<signal context parameter>; 19; 178; **180**; 198	<parámetro de contexto de señal>
<signal definition item>; **49**	<ítem de definición de señal>

<signal definition>; 18; 19; 24; 28; 29; 30; 33; 35; **49**; 89; 98; 111; 113; 199	<definición de señal>

<signal identifier>; 49; 54; 66; 67; 88; 103; 180	<identificador de señal>

Reemplazada por una versión más reciente

<signal list area>; 43; 44; 46; **49**; 93; 176	<área de lista de señales>
<signal list definition>; 24; 28; 29; 30; 33; 35; **49**; 98	<definición de lista de señales>
<signal list identifier>; 26; 49	<identificador de lista de señales>
<signal list item>; **49**	<ítem de lista de señales>
<signal list name>; 49	<nombre de lista de señales>
<signal list symbol>; 49; **50**	<símbolo de lista de señales>
<signal list>; 33; 43; 45; 47; **49**; 55; 88; 169; 170; 171; 175; 176	<lista de señales>
<signal name>; 49; 111; 113; 180	<nombre de señal>
<signal refinement>; 49; **89**; 118; 180	<refinamiento de señal>
<signal route definition area>; 31; 35; **46**; 48; 93; 99	<área de definición de ruta de señales>
<signal route definition>; 30; 32; 33; 34; **45**; 46; 47; 98	<definición de ruta de señales>
<signal route endpoint>; **45**	<punto extremo de ruta de señales>
<signal route identifier>; 48; 66	<identificador de ruta de señales>
<signal route identifiers>; 34; **48**; 177	<identificadores de ruta de señales>
<signal route name>; 45; 46	<nombre de ruta de señales>
<signal route path>; **45**; 177	<trayecto de ruta de señales>
<signal route symbol 1>; **46**	<símbolo de ruta de señales 1>
<signal route symbol 2>; **46**	<símbolo de ruta de señales 2>
<signal route symbol>; 20; **46**; 92; 95; 176	<símbolo de ruta de señales>
<signal route to route connection>; 33; 47; **48**; 98; 177; 196	<conexión de rutas de señales>
<signal signature>; **180**	<signatura de señal>
<simple expression>; **97**; 100; 101; 194; 195	<expresión simple>
<solid association symbol>; 20; **22**; 53; 92; 95; 104; 105	<símbolo de asociación de trazo continuo>
<sort constraint>; **182**	<limitación de género>
<sort context parameter>; 18; 49; 118; 178; **182**; 198; 199	<parámetro de contexto de género>
<sort identifier>; 120; 180	<identificador de género>
<sort list>; **49**; 70; 180; 181	<lista de géneros>
<sort name>; 17; 117; 118; 120; 182	<nombre de género>
<sort signature>; **182**	<signatura de género>
<sort type expression>; 137; 138	<expresión tipo de género>
<sort>; 17; 18; 26; 33; 40; 42; 49; 50; 51; 110; **120**; 123; 133; 136; 142; 143; 145; 147; 159; 163; 164; 179; 180; 181; 182	<género>
<space>; 15; 16; 17; 128	<espacio>
<special>; 14; **15**; 128	<especial>
<specialization>; 19; 39; 41; 49; 166; 167; 168; 169; 170; 172; 179; **182**; 197; 198; 199	<especialización>
<spelling term>; 122; **145**; 146	<término de ortografía>
<spontaneous designator>; **56**	<designador espontáneo>
<spontaneous transition area>; 53; **56**; 57; 93	<área de transición espontánea>
<spontaneous transition association area>; **53**; 93	<área de asociación de transición espontánea>
<spontaneous transition>; 52; **56**; 57; 107	<transición espontánea>
<start area>; 35; **51**; 52; 93; 169	<área de arranque>
<start symbol>; **51**	<símbolo de arranque>
<start>; 33; 40; **51**; 52; 104; 147	<arranque>
<state area>; 35; 41; **53**; 59; 93; 169	<área de estado>
<state list>; **52**; 53; 102	<lista de estados>
<state name>; 19; 52; 53; 59; 102; 103; 104	<nombre de estado>
<state symbol>; **53**; 59	<símbolo de estado>
<state>; 33; 40; **52**; 53; 102; 103; 104; 106; 108; 112; 114	<estado>
<stimulus>; **54**; 55; 103; 104; 105; 107; 197; 199	<estímulo>
<stop symbol>; 59; **61**; 93	<símbolo de parada>
<stop>; 58; **61**	<parada>
<structure definition>; 125; **136**	<definición de estructura>
<structure primary>; 149; 150; **152**	<primario de estructura>
<sub expression>; **149**	<subexpresión>
<subchannel identifiers>; **84**; 87; 177	<identificadores de subcanal>
<subsignal definition>; **89**; 180	<definición de subseñal>
<synonym constraint>; **181**	<limitación de sinónimo>
<synonym context parameter>; 118; 178; **181**	<parámetro de contexto de sinónimo>
<synonym definition item>; **142**	<ítem de definición de sinónimo>
<synonym definition>; 97; **142**; 150; 153; 195	<definición de sinónimo>
<synonym identifier>; 147; 150	<identificador de sinónimo>

<synonym name>; 142; 181	<nombre de sinónimo>

<synonym>; 97; 149; **150**	<sinónimo>

<syntactical unit>; 20; **21**	<unidad sintáctica>
<syntype definition>; **133**; 134; 153; 154; 159; 195	<definición de sintipo>
<syntype identifier>; 132; 133; 134; 180	<identificador de sintipo>
<syntype name>; 133	<nombre de sintipo>
<syntype>; 120; **132**; 133; 150	<sintipo>
<system definition>; 18; 23; 25; **26**; 27; 113; 118; 172; 195; 196; 198	<definición de sistema>
<system diagram>; 18; 26; **29**; 196	<diagrama de sistema>
<system heading>; **29**	<encabezamiento de sistema>
<system name>; 26; 28; 29; 172	<nombre de sistema>
<system text area>; 29; 31; 85; 87; 93; 99; 166	<área de texto de sistema>
<system type definition>; 18; 24; 27; 98; 118; **166**; 167	<definición de tipo de sistema>
<system type diagram>; 18; 25; 27;	<diagrama de tipo de sistema>
<system type expression>; 172	<expresión tipo de sistema>
<system type heading>; 166; **167**	<encabezamiento de tipo de sistema>
<system type identifier>; 166; 167	<identificador de tipo de sistema>
<system type name>; 166; 167	<nombre de tipo de sistema>
<system type reference>; 25; 99; **167**	<referencia de tipo de sistema>
<system type symbol>; **167**	<símbolo de tipo de sistema>
<task area>; 59; **62**; 93	<área de tarea>
<task body>; 55; **62**; 195	<cuerpo de tarea>
<task symbol>; **62**; 70; 71; 110	<símbolo de tarea>
<task>; 42; 55; 58; **62**	<tarea>
<term>; **122**; 123; 125; 127; 130	<término>
<terminator statement>; 57; **58**; 68; 69; 101	<enunciado de terminador>
<terminator>; **58**	<terminador>
<text extension area>; **22**; 94	<área de ampliación de texto>
<text extension symbol>; 20; **22**	<símbolo de ampliación de texto>
<text symbol>; **22**; 25; 29; 35; 38; 40; 148	<símbolo de texto>
<text>; **15**; 21; 22; 164	<texto>
<textual block reference>; **28**; 98	<referencia de bloque textual>
<textual block substructure reference>; 30; **84**; 167	<referencia de subestructura de bloque textual>
<textual block type reference>; 24; 28; 31; 98; **167**	<referencia de tipo de bloque textual>
<textual channel substructure reference>; 43; **87**	<referencia de subestructura de canal textual>
<textual endpoint constraint>; **175**; 176	<limitación de punto extremo textual>
<textual operator reference>; 146; **147**; 148	<referencia de operador textual>
<textual procedure reference>; 24; 28; 31; **33**; 37; 40; 98	<referencia de procedimiento textual>
<textual process reference>; 30; **31**; 34; 98	<referencia de proceso textual>
<textual process type reference>; 24; 28; 30; 98; **169**	<referencia de tipo de proceso textual>
<textual service reference>; 33; 34; 98	<referencia de servicio textual>
<textual service type reference>; 24; 28; 31; **33**; 98; **170**	<referencia de tipo de servicio textual>
<textual system definition>; 18; 26; **28**	<definición de sistema textual>
<textual system type reference>; 24; 98; **166**	<referencia de tipo de sistema textual>
<textual typebased block definition>; 28; 32; 34; 43; 44; 84; 85; 98; **173**; 176; 196	<definición textual de bloque basada en tipo>
<textual typebased process definition>; 30; 32; 34; 45; 47; 98; **173**; 174; 176; 196	<definición textual de proceso basada en tipo>
<textual typebased service definition>; 32; 33; 34; 37; 45; 47; 98; **174**; 175; 196	<definición textual de servicio basada en tipo>
<textual typebased system definition>; 28; **172**; 196	<definición textual de sistema basada en tipo>
<Time expression>; 70; 71	<expresión tiempo>
<timer active expression>; 160; **162**; 163; 198	<expresión temporizador activo
<timer constraint>; **181**	<limitación de temporizador>
<timer context parameter>; 166; 167; 169; 170; 178; **181**	<parámetro de contexto de temporizador>
<timer definition item>; **70**	<ítem de definición de temporizador>
<timer definition>; 33; 35; 37; 38; **70**; 98	<definición de temporizador>
<timer identifier>; 49; 54; 70; 162	<identificación de temporizador>
<timer name>; 70; 181	<nombre de temporizador>
<transition area>; 41; 51; 54; 56; 57; **59**; 60; 69; 94; 101; 104; 105; 113; 147	<área de transición>
<transition option area>; 59; 94; **101**; 195	<área de opción de transición>
<transition option symbol>; **101**	<símbolo de opción de transición>
<transition option>; 32; 58; 68; **100**; 101; 102; 195	<opción de transición>
<transition string area>; **59**; 94	<área de cadena de transición>
<transition string>; 57; **58**; 60; 68; 69; 101	<cadena de transición>
<transition>; 51; 54; 55; 56; 57; **58**; 68; 69; 101; 104; 105; 106; 112; 113; 114; 147; 195	<transición>
<type expression>; **171**; 172; 178; 179; 182; 183; 199	<expresión tipo>
<type in block area>; 31; 94; **168**	<tipo en área de bloque>

<type in process area>; 34; 94; **169**	<tipo en área de proceso>
<type in system area>; 25; 29; 85; 87; 94; 166; **167**	<tipo en área de sistema>
<typebased block heading>; **173**	<encabezamiento de bloque basado en tipo>
<typebased process heading>; **173**; 174	<encabezamiento de proceso basado en tipo>
<typebased service heading>; **174**	<encabezamiento de servicio basado en tipo>
<typebased system heading>; **172**	<encabezamiento de sistema basado en tipo>
<underline>; **14**; 15; 16; 17; 18; 128; 194	<subrayado>
<unquantified equation>; **122**; 123; 124	<ecuación no cuantificada>
<upward arrow head>; **14**	<punta de flecha hacia arriba>
<valid input signal set>; **33**; 34; 35; 36; 37; 38; 46; 47; 168; 169; 170; 171	<conjunto de señales de entrada válidas>
<value identifier>; 19; 122; 145; 146	<identificador de valor>
<value name>; 122; 123; 145	<nombre de valor>
<value returning procedure call>; 42; 148; 149; **163**; 164; 196; 198	<llamada a procedimiento de retorno de valor>
<variable access>; 154; **155**	<acceso a variable>
<variable context parameter>; 166; 167; 169; 178; **181**	<parámetro de contexto de variable>
<variable definition>; 17; 33; 35; 37; 38; 40; **50**; 51; 98; 111; 147; 148; 170	<definición de variable>
<variable identifier>; 55; 64; 110; 155; 157	<identificador de variable>
<variable name>; 33; 40; 42; 50; 147; 181	<nombre de variable>
<variables of sort>; **50**	<variable de género>
<variable>; 54; 55; **157**; 158; 159	<variable>
<vertical line>; **14**	<línea vertical>
<via path element>; **66**; 67	<elemento de trayecto vía>
<via path>; **66**; 67; 88	<trayecto vía>
<view definition>; 17; 33; 35; 37; 38; 50; **51**; 98	<definición de visión>
<view expression>; 160; **162**; 198	<expresión de visión>
<view identifier>; 162	<identificador de visión>
<view name>; 51	<nombre de visión>
<virtuality constraint>; 39; 41; 167; 168; 169; 170; **184**	<limitación de virtualidad>
<virtuality>; 21; 39; 41; 42; 51; 52; 54; 55; 56; 57; 104; 105; 106; 112; 113; 114; 147; 167; 168; 169; 170; **183**; 184; 185; 197	<virtualidad>
<word>; **14**; 17; 164	<palabra

abstract data types (G); 115
action (G); 59
active; 15; 162
active expression (G); 148; 154
active timer (G); 71; 163
Active-expression; 148; **154**
actual context parameters (G); 177
actual parameter (G); 63; 64; 91
adding; 15; 136; 137; 139; 141; 175; 176; 182; 188; 193
all; 15; 66; 67; 122; 124; 129; 132; 137; 138; 139; 145; 146; 197
alternative; 15; 100; 102; 130; 164; 165
Alternative-expression; **155**
and; 15; 48; 74; 78; 84; 86; 87; 88; 100; 127; 129; 132; 135; 137; 139; 149; 190
And-operator-identifier; **134**
any; 15; 68; 69; 163
anyvalue expression (G); 163
Anyvalue-expression; 160; **163**
Argument-list; **119**
as; 15; 39; 40; 50; 110; 113
assignment statement (G); 157
Assignment-statement; 62; **157**
association area (G); 53; 87
atleast; 15; 139; 172; 175; 178; 179; 180; 182; 184; 187; 189; 193
axiom (G); 116; 121
axioms; 15; 117; 137; 139; 140; 141; 142; 145; 146
basic SDL (G); 13
block; 15; 17; 20; 24; 28; 30; 31; 73; 74; 86; 88; 96; 100; 167; 168; 173; 187; 188; 189; 190
block (G); 32
block substructure (G); 84
block tree diagram (G); 84
block type (G); 168
Block-definition; 28; **30**; 83

Reemplazada por una versión más reciente

Anexo B

Glosario SDL

(Este anexo es parte integrante de la presente Recomendación)

La Recomendación Z.100 contiene las definiciones formales de terminología SDL. El glosario SDL se ha compilado para facilitar a los nuevos usuarios del SDL la lectura de la Recomendación y sus anexos, dándoles una breve definición, informal, de cada término y una referencia a la cláusula de la Recomendación en que éste aparece. Las definiciones contenidas en el glosario pueden ser versiones resumidas o expresadas con otras palabras de las definiciones formales, y podrían por tanto ser incompletas.

Los términos escritos en cursiva figuran en el glosario. Si una frase en cursiva, por ejemplo, *identificador de procedimiento*, no figura en el glosario, puede ser la concatenación de dos términos, en este caso, el término *identificador* seguido del término (*de*) *procedimiento*. Las palabras en cursiva que no figuren en el glosario pueden ser palabras derivadas de un término del glosario. Por ejemplo, *exportado* es el participio pasado de *exportar*.

Excepto cuando un término es sinónimo de otro, tras la definición del término hay una referencia principal al empleo del término en la presente Recomendación. Estas referencias se indican entre corchetes [ ] después de las definiciones. Por ejemplo [3.2], indica que la referencia principal está en la subcláusula 3.2.

acceso de entrada *(E: inlet)*

Un *acceso de entrada* representa una línea (por ejemplo, un *canal* o una *línea de flujo*) que entra en una *llamada a macro SDL/GR*. [4.2.3]

acceso de salida *(E: outlet)*

Un *acceso de salida* representa una línea (por ejemplo, un *canal* o una *línea de flujo*) que sale de un *diagrama de macro*. [4.2.2]

acción *(E: action)*

Una *acción* es una operación que se ejecuta dentro de una *cadena de transición*, por ejemplo, una *tarea*, *salida*, *decisión*, *petición de crear, inicializar, reinicializar, exportación* o *llamada a procedimiento*. [2.6.8.1, 2.7]

área; zona *(E: area)*

Un *área* es una región bidimensional en la *sintaxis gráfica concreta*. Las *áreas* a menudo corresponden a *nodos* en la *sintaxis abstracta* y usualmente contienen *sintaxis textual común*. En *diagramas de interacción*, las *áreas* pueden ser conectadas por *canales o rutas de señales*. En *diagramas de control de flujo* las *áreas* pueden ser conectadas por *líneas de flujo*. [2.4.2.6]

área de asociación *(E: association area)*

Un *área de asociación* es una conexión entre *áreas* en un *diagrama de interacción* por medio de un *símbolo* de asociación. Hay cinco áreas de asociación: *área de asociación de subestructura de canal*, *área de asociación de entrada*, *área de asociación de entrada* con prioridad, *área de asociación de señal continua* y *área de asociación de conservación*. [1.5.3, 2.6.3, 3.2.3, 4.10.2, 4.11]

área de fusión *(E: merge area)*

Un *área de fusión* es aquella en que una *línea de flujo* se conecta con otra. [2.6.8.2.2]

área de línea de crear *(E: create line area)*

El *área de línea de crear* en un *diagrama de bloque* conecta el *área de proceso* del *proceso creador* ***(progenitor)*** con el *área de proceso* del *proceso creado* ***(vástago)***. [2.4.3]

arranque *(E: start)*

El *arranque* en un *proceso* o *servicio* es una *cadena de transición* interpretada antes de cualquier *estado* o *acción* en la creación del *proceso*. [2.6.2]

arranque de procedimiento virtual *(E: virtual procedure start)*

Un *arranque de procedimiento* virtual es una *transición* que puede redefinirse en un *procedimiento* especializado. [2.4.6, 6.3.3]

arranque de proceso *(E: procedure start)*

El arranque de *proceso* en un *proceso* indica el punto en el que comienza la ejecución del *proceso* en una *llamada de procedimiento*. [2.4.6]

arranque virtual *(E: virtual start)*

Un *arranque virtual* es una *transición de arranque* que en un subtipo puede redefinirse en una nueva *transición de arranque*. [2.6.2, 6.3.3]

atributo revelado *(E: revealed attribute)*

Una *variable* perteneciente a un *proceso* puede tener un *atributo revelado*, en cuyo caso se permite que otro *proceso* en el *mismo bloque* vea el *valor* asociado con la *variable*. Véase *definición de visión*. [2.6.1.1]

axioma *(E: axiom)*

Un *axioma* es una clase especial de *ecuación* que implica una equivalencia con el *literal booleano* Verdadero. *«Axiomas»* se utiliza como sinónimo de *«axiomas y ecuaciones»*. [5.1.3, 5.2.3]

bloque *(E: block)*

Un *bloque* es una parte de un *sistema* o de un *bloque* y es el contenedor para uno o más procesos o una *subestructura de bloque*. Un *bloque* es una *unidad de ámbito* y proporciona una interfaz estática. Cuando se utiliza solo, *bloque* es sinónimo de *instancia de bloque*. [2.4.3]

BNF (forma Backus-Naur) *(E: BNF (Backus-Naur Form))*

La forma *BNF* (Backus-Naur) es una notación formal utilizada para expresar la *sintaxis textual concreta* de un lenguaje. Se utiliza una forma ampliada de *BNF* para expresar la *gramática gráfica concreta*. [1.5.2, 1.5.3]

boolean

Véase booleano.

booleano; boolean *(E: boolean)*

Booleano es un *género* definido en una *definición parcial de tipo* predefinida y tiene los *valores* Verdadero y Falso. Para el *género booleano* los operadores predefinidos son **not, and, or, xor** e implicación. [5.3.1.3, Anexo D]

cadena; string *(E: string)*

Cadena es un *generador de datos predefinidos* utilizado para introducir listas. Los *operadores* predefinidos incluyen longitud (length), primero (first), último (last), subcadena (substring) y concatenación representada por (//). [Anexo D]

cadena de caracteres; charstring *(E: charstring)*

Cadena de caracteres es un *género de datos predefinidos*, para el cual los *valores* son *cadenas de caracteres* y los *operadores* son los del *generador* predefinido *cadena* instanciado para caracteres. [Anexo D]

cadena de transición *(E: transition string)*

Una *cadena de transición* es una secuencia de cero o más *acciones*. [2.6.8.1]

calificador *(E: qualifier)*

El *calificador* es una parte de un *identificador* que constituye la información adicional al *nombre* del *identificador* para garantizar la unicidad. Los *calificadores* siempre están presentes en la *sintaxis abstracta* pero en la *sintaxis concreta*, sólo han de utilizarse en la medida que lo requiera la unicidad cuando el *calificador* de un *identificador* no pueda derivarse del contexto de utilización de la parte *nombre*. [2.2.2]

campo *(E: field)*

Un *campo* es un elemento de un *género estructurado*. [5.3.1.10]

canal *(E: channel)*

Un *canal* es la conexión que transporta *señales* entre dos *bloques*. Los *canales* transportan también *señales* entre un *bloque* y el *entorno*. Los *canales* pueden ser unidireccionales o bidireccionales. [2.5.1]

carácter; character *(E: character)*

Carácter es un *género* de *datos predefinidos* para el cual los *valores* son los elementos del alfabeto N.º 5 del CCITT (por ejemplo, 1, A, B, C, etc.). Para el *género carácter* los *operadores* de *ordenación* están predefinidos. [Anexo D]

character

Véase carácter.

charstring

Véase cadena de caracteres.

clase de entidad *(E: entity kind)*

Una *clase de entidad* es una categorización de los *tipos SDL* basada en la similitud de uso. [2.2.2]

comentario *(E: comment)*

Un *comentario* es una información completa o aclara la *especificación SDL*. En *SDL/GR* se puede asociar *comentarios* a cualquier *símbolo* mediante una línea de trazo discontinuo. En *SDL/PR* pueden introducirse comentarios por medio de la palabra clave ***comment***. Los *comentarios* no tienen un significado definido en *SDL*. Véase también *nota*. [2.2.6]

comportamiento *(E: behaviour)*

En *SDL*, el *comportamiento* es

1) comportamiento externamente observable, el conjunto de secuencias de respuestas de un *sistema* a secuencias de estímulos; o
2) comportamiento internamente observable, conjunto de acciones y tareas, desencadenadas por un estímulo, ejecutadas antes de que la máquina de estados transicione al estado siguiente. [1.1.3]

condición habilitante (o habilitadora) *(E: enabling condition)*

Una *condición habilitante* es un medio para aceptar condicionalmente una *señal* para *entrada*. [4.12]

conectar *(E: connect)*

Conectar indica la conexión de un *canal* a una o más *rutas de señales* o la interconexión de *rutas de señales*. [2.5.3]

conector *(E: connector)*

Un *conector* en *SDL/PR* es la *etiqueta* en una *acción*. Un *conector* es un *símbolo SDL/GR* que es un *conector de entrada* o un *conector de salida*. Hay una *línea de flujo* implícita desde los *conectores de salida* hasta el *conector de entrada* asociado en el mismo *proceso* o *procedimiento* identificado por el mismo nombre. [2.6.7, 2.6.8, 2.2]

conector de entrada *(E: in-connector)*

Véase *conector*.

conector de salida *(E: out-connector)*

Véase *conector*.

conjuntista *(E: powerset)*

Conjuntista es el *generador de datos predefinidos* utilizado para introducir conjuntos matemáticos. Los *operadores* para *conjuntista* son **in**, Incl, Del, unión, intersección y los *operadores de ordenación*. [Anexo D]

conjunto completo de señales de entrada válidas *(E: complete valid input signal set)*

El *conjunto completo de señales de entrada válidas* de un proceso es la unión del *conjunto de señales de entrada válidas*, las *señales* locales, las *señales de temporizador* y las *señales implícitas* del *proceso*. [2.4.4]

conjunto de señales de entrada válidas *(E: valid input signal set)*

El *conjunto de señales de entrada válidas* de un *proceso* es una lista de las *señales* externas manejadas por una *entrada* al *proceso*. Comprende las *señales* de *rutas de señales* que conducen al *proceso*. Compárese con *conjunto completo de señales de entrada válidas*. [2.4.4, 2.5.2]

conservación *(E: save)*

Una *conservación* es la declaración de las *señales* que no deben ser consumidas en un *estado* dado. [2.6.5]

conservación de procedimiento distante *(E: remote procedure save)*

Una *conservación de procedimiento* distante indica que el *procedimiento* exportado especificado no será ejecutado en el estado. [4.14]

conservación virtual *(E: virtual save)*

Una *conservación virtual* es una *conservación* que en un *subtipo* puede redefinirse en una *transición de entrada*. [2.6.5, 6.3.3]

constricción *(E: constraint)*

La *constricción* de un *parámetro de contexto* constriñe a los *parámetros de contexto efectivos* y define las propiedades del parámetro conocido como el *tipo parametrizado*. La *constricción* de un *tipo virtual* constriñe las redefiniciones y define las propiedades del *tipo virtual* conocido como el *tipo* que lo encierra. Véase también *constricción de puerta*. [6.2]

constricción de género *(E: sort constraint)*

Una *constricción de género* es el requisito de los *parámetros efectivos* de ser un *parámetro de contexto de género* formal, sea en términos de un género (el *género* efectivo debe entonces ser un *subtipo* de este *tipo*) o de una *signatura de género* (el *género* efectivo debe entonces ser compatible con los operadores en esta *signatura de género*). [6.2.9]

constricción de puerta *(E: gate constraint)*

Una *constricción de puerta* constriñe la forma en que los/las *canales/rutas de señales* pueden conectarse a la *puerta*, e impone restricciones sobre sus *señales* entrantes y salientes. [6.1.4]

constricción de puerta gráfica *(E: graphical gate constraint)*

Una *constricción de puerta gráfica* es la representación *SDL/GR* de una *constricción de puerta*. [6.1.4]

constricción de procedimiento *(E: procedure constraint)*

Una *constricción de procedimiento* es el requisito de los *parámetros efectivos* de ser un *parámetro de contexto de procedimiento* formal sea en términos de un *procedimiento* o de una *constricción de procedimiento*. [6.2.2]

constricción de proceso *(E: process constraint)*

Una *constricción de proceso* es el requisito de los *parámetros efectivos* de ser un *parámetro de contexto de proceso* formal, sea en términos de un *tipo de proceso* (la *instancia de proceso* efectiva debe entonces ser un proceso de este *tipo* o de un *subtipo*) o de una *signatura de proceso* (la *instancia de proceso* efectiva debe entonces ser compatible con la *signatura de proceso*). [6.2.1]

constricción de punto extremo *(E: endpoint constraint)*

Una *constricción de punto extremo* es la representación *SDL/GR* de una *constricción* impuesta a una *puerta* que impone requisitos sobre cuáles *bloques/procesos/servicios* puedan estar en el otro extremo de un/una *canal/ruta de señales/ruta de señales de servicio* conectado a la *puerta*. [6.1.4]

constricción de punto extremo textual *(E: textual endpoint constraint)*

Una *constricción de punto extremo textual* es la representación *SDL/PR* de una *constricción de punto extremo* de una *puerta* que impone requisitos sobre cuáles *bloques/procesos/servicios* pueden estar en el otro extremo de un/una *canal/ruta de señales/ruta de señales de servicio* conectado a la *puerta*. [6.1.4]

constricción de señal *(E: signal constraint)*

Una *constricción de señal* es el requisito de los *parámetros efectivos* de ser un *parámetro de contexto de señal* formal, sea en términos de un *tipo de señal* (que especifique que la *definición de señal* debe ser un *subtipo* del *tipo de constricción*) o de una *signatura de señal* (la *instancia de señal* efectiva debe entonces ser compatible con la *signatura de señal*). [6.2.4]

constricción de virtualidad *(E: virtuality constraint)*

La *constricción de virtualidad* de un *tipo virtual* impone requisitos a las redefiniciones del *tipo virtual* por medio de un *tipo de constricción*. Tanto la definición como las redefiniciones de un *tipo virtual* deben ser definiciones de *subtipos* del *tipo de constricción*. El *tipo de constricción* de un *tipo virtual* también determina las propiedades del *tipo virtual* conocido como el *tipo* que lo encierra que tiene el *tipo virtual* local. [6.3.2]

crear *(E: create)*

Crear es sinónimo de *petición de crear*.

datos predefinidos *(E: predefined data)*

Para simplificar la descripción, la expresión *datos predefinidos* se aplica tanto a los *nombres* predefinidos para los *géneros* introducidos por *definiciones parciales de tipo* como a los *nombres* predefinidos para *generadores de tipo de datos*. *Booleano*, *carácter*, *cadena de caracteres*, *duración*, *entero*, *natural*, *PId*, *real* y *tiempo* son *nombres de género* predefinidos. *Matriz*, *conjuntista* y *cadena* son *nombres* predefinidos de *generador de tipo de datos*. Los *datos predefinidos* están definidos en el *paquete predefinido* implícitamente utilizado. [2.4.1.2, 5.1.1, 5.3.1.3, Anexo D]

decisión *(E: decision)*

Una *decisión* es una *acción* dentro de una *transición* que hace una pregunta cuya respuesta puede obtenerse en ese instante y, según la respuesta, selecciona una de las *transiciones* que salen de la *decisión* para continuar la interpretación. [2.7.5]

definición *(E: definition)*

Una *definición* asocia un nombre y un conjunto de propiedades con un *tipo* o una *instancia*. [1.3.1]

definición de puerta *(E: gate definition)*

Una *definición de puerta* es la representación *SDL/PR* de una *puerta*. (6.1.4]

definición de señal *(E: signal definition)*

Una *definición de señal* define un *tipo de señal nombrado* y asocia una lista de cero o más *identificadores de género al nombre señal*. Esta asociación permite a las *señales* vehicular *valores*. [2.5.4]

definición de tipo *(E: type definition)*

Una *definición de tipo* define las propiedades de un *tipo*. [1.3.1]

definición de tipo de datos *(E: data type definition)*

La *definición de tipo de datos* en cualquier punto dado en una *especificación SDL* define la validez de los *operadores, géneros y expresiones,* y la relación entre las *expresiones.* [5.2.1]

definición de variable *(E: variable definition)*

Una *definición de variable* es la declaración de que los *nombres de variable* enumerados serán *visibles* en el *proceso, procedimiento* o *servicio* que contiene la definición. de la variable [2.6.1.1]

definición de visión *(E: view definition)*

Una *definición de visión* define una visión de una *variable* de otro *proceso* (donde tiene el *atributo revelado*). Esto permite al *proceso* que observa *acceder* al *valor* de esa *variable*. [2.6.1.2]

definición parcial de tipo *(E: partial type definition)*

La *definición parcial de tipo* para un *género* define algunas de propiedades relacionadas con el *género*. Una *definición parcial de tipo* es parte de una definición de *tipo de datos*. [5.2.1]

definición referenciada *(E: referenced definition)*

Una *definición referenciada* es un medio sintáctico para distribuir una *definición de sistema* entre varias partes y relacionar las partes entre sí. [2.4.1.3]

descripción *(E: description)*

Una *descripción* de un *sistema* es la descripción de su *comportamiento* efectivo. [1.1]

diagrama *(E: diagram)*

Un *diagrama* es la representación *SDL/GR* de una parte de una *especificación*. [2.4.2]

diagrama arborescente de bloques *(E: block tree diagram)*

Un *diagrama arborescente de bloques* es un documento auxiliar en *SDL/GR* que representa la *partición* de un *sistema* en *bloques* a *niveles de abstracción* sucesivamente inferiores por medio de un diagrama arborescente invertido (es decir, con el *bloque* en la parte superior). [3.2.1]

diagrama de flujo de control *(E: control flow diagram)*

Un *diagrama de flujo de control* es un *diagrama de proceso,* un *diagrama de procedimiento* o un *diagrama de servicio*.

diagrama de interacción *(E: interaction diagram)*

Un *diagrama de interacción* es un *diagrama de bloque, diagrama de sistema, diagrama de subestructura de canal,* o *diagrama de subestructura de bloque.*

duración; duration *(E: duration)*

Duración es un *género de datos predefinidos* en el cual los *valores* están denotados como *reales* y representan el intervalo entre dos instantes de tiempo. [Anexo D]

duration

Véase duración.

ecuación *(E: equation)*

Una *ecuación* es una relación entre *términos* del mismo *género* que se cumple para todos los *valores* posibles de cada *identificador de valor de la ecuación*. Una *ecuación* puede ser un *axioma*. [5.1.3]

emisor

Véase sender.

entero; integer *(E: integer)*

Entero es un *género de datos predefinidos* para el cual los *valores* son los enteros matemáticos (. . ., –2, –1, 0, +1, +2, . . .). Para el *género entero*, los *operadores* predefinidos son +, –, *, /, y los *operadores de ordenación*. [Anexo D]

entorno *(E: environment)*

El término *entorno* es sinónimo de *entorno de un sistema*. Cuando el contexto lo permita, puede ser sinónimo de *entorno de un bloque*, *proceso*, *procedimiento* o *servicio*.

entorno de un sistema *(E: environment of a system)*

El *entorno de un sistema* es el mundo exterior del *sistema* que se especifica. El *entorno* interactúa con el *sistema* enviando/recibiendo *instancias de señal* a/desde el *sistema*. [1.3.2]

entrada *(E: input)*

Una *entrada* es el consumo de una *señal* procedente del *puerto de entrada* que inicia una *transición*. Durante el consumo de una *señal*, los *valores* asociados a la *señal* se hacen disponibles a la *instancia de proceso*. [2.6.4, 4.10.2]

entrada de procedimiento distante virtual *(E: virtual remote procedure input)*

Una *entrada de procedimiento distante virtual* es una *transición* que puede redefinirse en una nueva *transición de entrada de procedimiento distante* o en una *conservación de procedimiento distante*. [6.3.3]

entrada prioritaria *(E: priority input)*

Una *entrada prioritaria* es una *notación taquigráfica* que indica que la recepción de las *señales listadas* debe tener precedencia sobre las *señales* listadas en las *entradas* a ese *estado.* [4.10]

entrada prioritaria virtual *(E: virtual priority input)*

Una *entrada prioritaria virtual* es una *transición* que podría redefinirse en una *entrada prioritaria* o en una *conservación*. [6.3.3]

entrada virtual *(E: virtual input)*

Una *entrada virtual* es una *transición de entrada* que en un *subtipo* puede redefinirse en una *conservación* o en una *transición de entrada*. [2.6.4, 6.3.3]

error *(E: error)*

Se produce un *error* durante la interpretación de una *especificación válida* de un *sistema* cuando se viola una de las condiciones dinámicas del *SDL*. El *comportamiento* del *sistema* después de producirse un *error* no está definido. [1.3.3]

especialización *(E: specialization)*

La *especialización* crea un nuevo tipo, denominado subtipo, añadiendo propiedades y/o redefiniendo propiedades de otro tipo, que se denomina supertipo. [1.3.1]

especificación *(E: specification)*

Una *especificación* es una definición de los requisitos de un *sistema*. Una *especificación* consiste en *parámetros generales* requeridos del *sistema* y la *especificación funcional* de su *comportamiento* requerido. *Especificación* puede utilizarse también como notación taquigráfica de «*especificación y/o descripción*», por ejemplo, en las expresiones *especificación SDL* o *especificación de sistema*. [1.1]

especificación de procedimiento distante *(E: remote procedure specification)*

Una *especificación de procedimiento distante* introduce el nombre y la *signatura de procedimiento* para *procedimientos importados* y *exportados*. [4.14]

especificación de variable distante *(E: remote variable specification)*

Una *especificación de variable distante* introduce el nombre y la *signatura de género* para *valores importados* y *exportados*. [4.13]

especificación válida *(E: valid specification)*

Una *especificación válida* es una *especificación* que cumple la *sintaxis concreta* y las *reglas de formación correcta* estáticas. [1.3.3]

estado *(E: state)*

Un *estado* es una condición en la cual una *instancia de proceso* puede consumir una señal. [2.6.3]

estado siguiente *(E: nextstate)*

El *estado siguiente* termina una transición y especifica el nuevo *estado* de la *instancia de proceso*. [2.8.6.2.1]

estructura jerárquica *(E: hierarchical structure)*

Una *estructura jerárquica* es una estructura de una *especificación de sistema* en la cual una *partición* y un *refinamiento* permiten diferentes visiones del *sistema* a diferentes *niveles de abstracción*. Véase también *diagrama del árbol de bloques.* [3.1]

etiqueta *(E: label)*

Una *etiqueta* es un *nombre* seguido del carácter «:» y se utiliza en la *sintaxis textual concreta* para indicar el objetivo de la transferencia de control desde un *nódulo* que referencia el mismo *identificador* que la *etiqueta*. [2.6.7, 2.6.8.2.2]

exportación *(E: export)*

El término *exportación* es sinónimo de *operación de exportación.*

exportador *(E: exporter)*

Un *exportador* de una *variable* es la *instancia de proceso* que posee la *variable* y *exporta* su *valor*. [4.13]

expresión *(E: expression)*

Una *expresión* es un *literal*, una aplicación de *operador*, un *sinónimo*, un *acceso a variable*, una *expresión condicional* o un *operador imperativo* aplicado a una o más *expresiones*. Cuando se interpreta una *expresión*, se obtiene un *valor* (o el *sistema* está en *error*). [2.3.4, 5.3.3.1]

expresión activa *(E: active expression)*

Una *expresión activa* es una *expresión* en la que el *valor* depende del estado en curso del sistema. Una *expresión activa* accede a una *variable* o contiene un *operador imperativo*. [5.3.3.1, 5.4.2.1]

expresión activa de temporizador *(E: timer active expression)*

Una *expresión activa de temporizador* es una *expresión booleana* que cuando se ejecuta indica si el temporizador identificado es un temporizador activo. [5.4.4.5]

expresión ahora *(E: now expression)*

Una *expresión ahora* es una *expresión* del *género de tiempo* que da el tiempo en curso del sistema. [5.4.4.1]

expresión condicional *(E: conditional expression)*

Una *expresión condicional* es una *expresión* que contiene una *expresión booleana* que controla si es interpretada la *expresión* consecuencia o la *expresión* alternativa. [5.4.2.3]

expresión cualquier valor *(E: anyvalue expression)*

Una *expresión cualquier valor* es una *expresión* que da un *valor* no especificado del *género* o *sintipo* designado. [5.4.4.6]

expresión de importación *(E: import expression)*

Una *expresión de importación* especifica la *palabra clave* **importación** y el *identificador* de una *variable*. [4.13]

expresión de tipo *(E: type expression)*

Una *expresión de tipo* denota un *tipo* que es un *tipo* de base o un *tipo* anónimo definido aplicando *parámetros de contexto efectivos* a un *tipo* de base parametrizado. [6.1.2]

expresión de visión *(E: view expression)*

Una *expresión de visión* se utiliza dentro de una *expresión* para dar el *valor* en curso de una *variable vista* («visionada»). [5.4.4.4]

expresión fundamental *(E: ground expression)*

Una *expresión fundamental* es una *expresión* que contiene solamente *operadores*, *sinónimos* y *literales*. [5.3.3.2]

expresión simple *(E: simple expression)*

Una *expresión simple* es una *expresión* que sólo contiene *operadores*, *sinónimos* y *literales* de los *géneros* predefinidos. [4.3.2]

extract!

Véase extraer!

extraer!; **extract!** *(E: extract!)*

Extraer! es un *operador* implícito para extraer un *campo* de un *género estructurado* y que está implícito en una *expresión* cuando una *variable* va seguida inmediatamente por una o más *expresiones* entre paréntesis. [5.3.1.10, 5.3.3.4, 5.3.3.5]

generador *(E: generator)*

Un *generador* permite definir una plantilla de texto parametrizada que se expande por transformación antes de que se considere la semántica de los tipos de datos. [5.3.1.12, 7.2]

género *(E: sort)*

Un *género* es un conjunto de *valores* con características comunes. Los *géneros* son siempre no vacíos y disjuntos. [2.3.3, 5.1.2, 5.1.3, 5.2.1]

género estructurado *(E: structured sort)*

Un *género estructurado* es un *género* cuyos valores se componen a partir de una lista de *valores* de *géneros (campos).* Un *género estructurado* tiene *operadores* y *ecuaciones* implícitos y una *sintaxis concreta* especial para estos *operadores* implícitos mediante los cuales se puede *acceder* a los *valores* de los *campos* y modificarlos independientemente. [5.3.1.10]

gráfico *(E: graph)*

En la *sintaxis abstracta*, un *gráfico* es una parte de una *especificación SDL*, por ejemplo, un *gráfico de procedimiento* o un *gráfico de proceso*. [2.4.4, 2.4.5, 2.4.6]

gramática abstracta *(E: abstract grammar)*

La *gramática abstracta* define la *semántica* del *SDL*. La *gramática abstracta* está descrita por la *sintaxis abstracta* y las *reglas de formación correcta*. [1.2, 1.4.1]

gramática concreta *(E: concrete grammar)*

Una *gramática concreta* es la *sintaxis concreta* junto con las *reglas de formación correcta* para esa *sintaxis concreta*. *SDL/GR* y *SDL/PR* son las *gramáticas concretas de SDL*. Las *gramáticas concretas* se hacen corresponder con la *gramática abstracta* para determinar su *semántica*. [1.2]

gramática gráfica concreta *(E: concrete graphical grammar)*

La *gramática gráfica concreta* es la *gramática concreta* para la parte gráfica de *SDL/GR*. [1.2, 1.5.3]

gramática textual común *(E: common textual grammar)*

La *gramática textual común* es el subconjunto de la *gramática textual concreta* que se aplica tanto al *SDL/GR* como al *SDL/PR*. [1.2]

hacer!; make! *(E: make!)*

Hacer! es una operación que sólo se utiliza en definiciones de *tipos de datos* para formar un *valor* de un tipo complejo (es decir, *género estructurado*). [5.3.1.10]

heredar *(E: inherit)*

Un *tipo* o *género especializado* tiene o *hereda* todas las propiedades de su *supertipo* o *género* referenciado. [5.3.1.11, 6.3]

identificador *(E: identifier)*

Un *identificador* es la identificación única de un objeto; consta de una parte *calificador* y un *nombre*. [2.2.2]

importación *(E: import)*

El término *importación* es sinónimo de *operación de importación*. [4.13]

inicialización por defecto *(E: default initialization)*

Una *inicialización por defecto* es una notación para asociar el mismo *valor* a todas las *variables* del *género* especificado antes de que se interprete su *proceso* o *procedimiento* asociado. [5.4.3.3]

importador *(E: importer)*

Un *importador* de una *variable importada* es la *instancia de proceso* que *importa* el *valor*. [4.13]

inicializar; poner *(E: set)*

Operación definida para *temporizadores*, que convierte a un *temporizador* en un *temporizador activo*. Especifica el tiempo del sistema en que el *temporizador activo* debe devolver una *señal de temporizador*. [2.8]

instancia *(E: instance)*

Una *instancia* es una parte de un sistema. Una instancia tiene propriedades y puede manifestar comportamiento. Una *instancia* de un tipo es un objeto que tiene todas las propiedades del tipo indicadas en la definición de tipo. [1.3.1]

instancia de proceso *(E: process instance)*

Una *instancia de proceso* es un miembro de un conjunto de *proceso*. Una *instancia* de *proceso* se crea en el momento de creación del sistema o dinámicamente de resultas de un *crear*. Véanse ***self***, ***sender***, ***parent*** y ***offspring***. [2.4.4]

instanciación *(E: instantiation)*

Instanciación es la creación de una *instancia* de un *tipo*. [1.3.1]

integer

Véase entero.

interfaz de paquete *(E: package interface)*

Una *interfaz de paquete* enumera las definiciones visibles de un *paquete*. [2.4.1.2]

línea de flujo *(E: flow line)*

Una *línea de flujo* es un *símbolo* utilizado para conectar *áreas* en un *diagrama de flujo del control*. [2.2.4, 2.6.8.2.2]

lista de señales *(E: signal list)*

Una *lista de señales* es una lista de *identificadores de señal* utilizados en *definiciones de canal* y de *ruta de señales* para indicar qué *señales* pueden ser transportadas por el *canal* o la *ruta de señales* en un sentido. [2.5.5]

literal *(E: literal)*

Un *literal* denota un *valor; es un operador* sin argumento. [2.3.3, 5.1.2, 5.2.2, 5.3.1.14]

llamada a (de) macro *(E: macro call)*

Una *llamada a macro* es una indicación de un lugar en el cual debe expandirse la *definición de macro* que tiene el mismo *nombre*. [4.2.3]

llamada a (de) procedimiento *(E: procedure call)*

Una *llamada a procedimiento* es la invocación de un *procedimiento* que tiene un *nombre*, con el fin de efectuar la interpretación del *procedimiento* y pasar a éste *parámetros efectivos*. [2.7.3]

llamada a (de) procedimiento distante *(E: remote procedure call)*

Una *llamada a procedimiento distante* es una petición de un *proceso* cliente para que un servidor ejecute un *procedimiento* definido en el servidor *proceso.* [4.14]

macro *(E: macro)*

Una *macro* es una colección de elementos sintácticos que tiene un *nombre* y reemplaza a la *llamada a macro* antes de que se considere el significado de la representación *SDL* (es decir, una *macro* sólo tiene significado cuando es reemplazada en un contexto determinado). [4.2]

make

Véase hacer!

matriz *(E: array)*

Matriz es el *generador de datos predefinidos* utilizado para introducir el concepto de matrices, que simplifica la definición de matrices. [Anexo D]

Meta IV *(E: Meta IV)*

Meta IV es una notación formal para expresar la *sintaxis abstracta* de un lenguaje. [1.5.1]

Mismo

Véase SELF.

modelo *(E: model)*

Un *modelo* da la relación de correspondencia para *notaciones taquigráficas* expresadas en términos de una *sintaxis concreta* previamente definida. [1.4.1, 1.4.2]

modificar!; modify! *(E: modify!)*

Modificar! es un *operador* o *géneros estructurados* para modificar un *campo* a partir de un *género estructurado* y está implícito en *expresiones* cuando una *variable* va seguida inmediatamente de expresiones entre paréntesis y, después, de :=. Dentro de axiomas, *modificar!* se utiliza explícitamente (véase tambien *extraer!*). [5.3.1.10, 5.4.3.1, 5.4.3.2]

modify!

Véase modificar!

natural *(E: natural)*

Natural es un *sintipo de datos predefinidos* para el cual los *valores* son enteros no negativos (esto es, 0, 1, 2, . . .). Los *operadores* son los *operadores* del *género entero*. [Anexo D]

neotipo *(E: newtype)*

Un *neotipo* introduce un *género*, un conjunto de *operadores* y un conjunto de *ecuaciones*. Obsérvese que el término *neotipo* podría dar lugar a cierta confusión porque lo que en realidad se introduce es un nuevo *género*; no obstante, se mantiene *neotipo* por razones históricas. [5.2.1]

nivel *(E: level)*

El término *nivel* es sinónimo de *nivel de abstracción*.

nivel de abstracción *(E: level of abstraction)*

Un *nivel de abstracción* es uno de los niveles de un *diagrama de árbol de bloques*. Una descripción de un *sistema* es un *bloque* en el *nivel de abstracción* más elevado y se muestra como un solo *bloque* en la parte superior de un *diagrama de árbol de bloques*. [3.2.1]

nodo *(E: node)*

En la *sintaxis abstracta*, un *nodo* es una designación de uno de los conceptos básicos del *SDL*.

nódulo *(E: join)*

Un *nódulo* indica un cambio en el flujo de ejecución de una *cadena de transición* por indicación al *conector* a o a la *etiqueta* de la *acción* que debe ejecutarse a continuación. [2.6.8.2.2]

nombre *(E: name)*

Un *nombre* es una *unidad léxica* utilizada para nombrar objetos *SDL*. [2.2.1, 2.2.2]

nota *(E: note)*

Una *nota* es un texto delimitado por /* y */ que no tiene una semántica definida en *SDL*. Véase también *comentario*. [2.2.1]

notación taquigráfica (o abreviada) *(E: shorthand notation)*

Una *notación taquigráfica* es una notación de *sintaxis concreta* que proporciona una representación más compacta que hace referencia implícita a conceptos de *SDL básico*. [1.4.2, 4.1]

null; nulo *(E: null)*

Null es el *literal* del *género datos predefinidos PId*. [Anexo D]

nulo

Véase null.

offspring; vástago *(E: offspring)*

offspring es una *expresión* de *género datos predefinidos PId*. Cuando **offspring** se evalúa en un *proceso*, da el *valor PId* del *proceso creado* más recientemente por este *proceso*. Si el *proceso* no ha *creado* ningún *proceso*, el resultado de la evaluación de **offspring** es *null*. [2.4.4]

opción *(E: option)*

Una *opción* es una construcción de *sintaxis concreta* en una *especificación de sistema SDL* genérica que permite elegir diferentes estructuras de *sistema* antes de que el *sistema* sea interpretado. [4.3.3, 4.3.4]

operación de exportación *(E: export operation)*

Una *operación de exportación* es la ejecución de una *acción de exportación* mediante la cual el *exportador* revela el *valor* en curso de una *variable*. Véase también *operación de importación*. [4.13]

operación de importación *(E: import operation)*

Una *operación de importación* es la ejecución de una *expresión de importación* mediante la cual el *importador* accede al *valor* de una *variable exportada*. [4.13]

operador *(E: operator)*

Un *operador* es una denotación para una operación. Los *operadores* se definen en una *definición parcial de tipo*. Por ejemplo, +, –, *, /, son *nombres* de *operadores* definidos para el *género entero*. [5.1.2, 5.1.3, 5.2.1, 5.3.2]

operador imperativo *(E: imperative operator)*

Un *operador imperativo* es una *expresión de visión*, *expresión activa de temporizador*, *expresión de importación*, *expresión de cualquier valor*, *expresión ahora*, o una de las *expresiones PId*: ***self***, ***parent***, ***offspring*** o ***sender***. [5.4.4]

operador infijo *(E: infix operator)*

Un *operador infijo* es uno de los *operadores* diádicos predefinidos del *SDL* (=>, **or**, **xor**, **and**, **in**, /=, =, >, <, <=, >=, +, –, //, *, /, **mod**, **rem**) que van entre sus dos argumentos. [5.3.1.1]

operadores de ordenación *(E: ordering operators)*

Los *operadores de ordenación* son <, <=, >, >=. [5.3.1.8]

página *(E: page)*

Una *página* es uno de los componentes de una partición física de un *diagrama*. [2.2.5]

palabra clave *(E: keyword)*

Una *palabra clave* es una *unidad léxica* en la *sintaxis textual concreta*. [2.2.1]

paquete *(E: package)*

Un *paquete* se compone de un conjunto de definiciones. [2.4.1.2]

parada *(E: stop)*

Una *parada* es una acción que termina una *instancia de proceso*. Cuando una *parada* es interpretada, todas las *variables* pertenecientes a la *instancia de proceso* son destruidas y todas las *señales retenidas* en el *puerto de entrada* dejan de ser accesibles. [2.6.8.2.3]

parámetro de contexto *(E: context parameter)*

Un *parámetro de contexto* (formal) de una definición de *tipo parametrizado* es la asociación de un *identificador* y una *constricción*. Un nuevo *tipo* se define (mediante una *expresión de tipo*) asociando algunos o todos los *parámetros de contexto* con definiciones efectivas. [6.2]

parámetro de contexto de género *(E: sort context parameter)*

Un *parámetro de contexto de género* es un *parámetro de contexto* para el cual el *parámetro efectivo* debe ser un *género* que cumple la *constricción de género*. [6.2.9]

parámetro de contexto de procedimiento *(E: procedure context parameter)*

Un *parámetro de contexto de procedimiento* es un *parámetro de contexto* para el cual el *parámetro de contexto efectivo* debe ser un *procedimiento* que cumpla la *constricción de procedimiento*. [6.2.2]

parámetro de contexto de proceso *(E: process context parameter)*

Un *parámetro de contexto de proceso* es un *parámetro de contexto* para el cual el *parámetro efectivo* debe ser una definición de proceso o una definición de proceso tipificada que cumpla la *constricción de proceso*. [6.2.1]

parámetro de contexto de señal *(E: signal context parameter)*

Un *parámetro de contexto de señal* es un *parámetro de contexto* para el cual el *parámetro efectivo* debe ser un *tipo de señal* que cumpla la *constricción de señal*. [6.2.4]

parámetro de contexto de sinónimo *(E: synonym context parameter)*

Un *parámetro de contexto de sinónimo* es un *parámetro de contexto* para el cual el *parámetro efectivo* debe ser un *sinónimo* que cumpla el requisito de los *parámetros efectivos* de ser un *parámetro de contexto de sinónimo* formal en términos de un *tipo de género*. [6.2.8]

parámetro de contexto de temporizador *(E: timer context parameter)*

Un *parámetro de contexto de temporizador* es un *parámetro de contexto* para el cual el *parámetro efectivo* debe ser una *definición de temporizador*. [6.2.7]

parámetro de contexto de variable *(E: variable context parameter)*

Un *parámetro de contexto de variable* es un *parámetro de contexto* para el cual el *parámetro efectivo* debe ser una *variable* que cumpla el requisito de los *parámetros efectivos* de ser un *parámetro de contexto* de variable formal en términos de un *tipo de género*; la *variable* efectiva debe ser una *variable* de este *tipo*. [6.2.5]

parámetro de contexto efectivo *(E: actual context parameter)*

Un *parámetro de contexto efectivo* es un *identificador* que denota la definición efectiva de un *parámetro de contexto formal* correspondiente. El *identificador de parámetro efectivo* puede ser un *identificador* de un *parámetro de contexto formal* de una definición de *subtipo*, o denotar una definición que es visible en el ámbito que encierra la *expresión de tipo* o visible a través de una *referencia de paquete*. [6.2]

parámetro de contexto formal *(E: formal context parameter)*

Véase *parámetro de contexto*.

parámetro efectivo *(E: actual parameter)*

Un *parámetro efectivo* es una *expresión* interpretada por un *proceso (*o *procedimiento)* para el *parámetro formal* correspondiente cuando el *proceso* o *procedimiento* es *creado* (o *llamado*). Obsérvese que en ciertos casos, en una *llamada a procedimiento*, un *parámetro efectivo* tiene que ser una *variable* (es decir, un tipo particular de *expresión*; véase *variable IN/OUT*). [2.7.2, 2.7.3, 4.2.2]

parámetro formal *(E: formal parameter)*

Un *parámetro formal* es un *nombre de variable* al cual se asignan *valores efectivos* o que es reemplazado por *variables efectivas*. [2.4.4, 2.4.5, 2.4.6, 4,2]

parámetro in *(E: in parámetro)*

Un *parámetro in* es un atributo de *parámetro formal* en que un *valor* es pasado a un *procedimiento* vía un *parámetro efectivo*. [2.4.6]

parámetro in/out *(E: in/out parameter)*

Un parámetro *in/out* es un atributo de *parámetro formal* en que un *nombre de parámetro formal* se utiliza como un sinónimo de la *variable* (esto es, el *parámetro efectivo* tiene que ser una *variable*). [2.4.6]

parámetros generales *(E: general parameters)*

Los *parámetros generales* en una *especificación* y en una *descripción* de un *sistema* se refieren a factores no *comportamentales* tales como los límites de temperatura, detalles de construcción, capacidad de las centrales, rendimiento, etc., y no están definidos en *SDL*. [1.1]

parent; progenitor *(E: parent)*

parent es una *expresión* del *género de datos predefinidos PId*. Cuando un *proceso* evalúa esta *expresión*, el resultado es el *valor PId* del *proceso* progenitor. Si el *proceso* fue creado en el momento de inicialización del *sistema*, el resultado es *null*. [2.4.4]

partición *(E: partitioning)*

Una *partición* es la división de una unidad en componentes menores que, tomados en su conjunto, tienen el mismo *comportamiento* que la unidad inicial. La *partición* no afecta a la interfaz estático de una unidad. [3.1, 3.2]

petición de crear *(E: create request)*

Una *petición de crear* es la *acción* que causa la creación y el comienzo de una nueva *instancia de proceso* que utiliza, como una plantilla, un *tipo de proceso* especificado. Los *parámetros efectivos* en la *petición de crear* reemplazan a los *parámetros formales* en el *proceso*. [2.7.2]

PId *(E: PId)*

PId es un *género de datos predefinidos* para el cual hay un *literal*, *null*. *PId* es una abreviatura de identificador de instancia de proceso (en inglés process instance identifier), y los *valores* de los *géneros* se utilizan para identificar *instancias de proceso*. [Anexo D]

poner

Véase inicializar.

predefinido *(E: predefined)*

Predefinido es un *paquete* que contiene las *definiciones de tipo parcial* para la *fecha predefinida*. [2.4.1.2, 5.2.1, 5.3.1.3, Anexo D]

procedimiento *(E: procedure)*

Un *procedimiento* es una encapsulación de una parte del *comportamiento* de parte de un *proceso*. Un *procedimiento* está definido en un lugar pero se puede hacer referencia al mismo varias veces dentro de un *proceso*. [2.4.6]

procedimiento que retorna un valor *(E: value returning procedure)*

Un *procedimiento que retorna un valor* es un *procedimiento* que puede retornar un *valor*. [5.4.5]

proceso *(E: process)*

Un *proceso* es una máquina de estados finitos ampliadas que comunica. La comunicación puede tener lugar vía *señales* o *variables* compartidas. El *comportamiento* de un *proceso* depende del orden de llegada de las *señales* a su *puerto de entrada*. [2.4.4]

progenitor

Véase **parent.**

puerta *(E: gate)*

Una *puerta* se define en un *tipo de bloque/proceso/servicio* y representa un punto de conexión para *canales/rutas de señales* que conectan instancias del *tipo* con otras instancias o con el símbolo de trama que lo encierra. [6.1.4]

puerto de entrada *(E: input port)*

Un *puerto de entrada* de un *proceso* es una cola que recibe y retiene *señales* en el orden de llegada hasta que las *señales* son consumidas por una *entrada*. El *puerto de entrada* puede contener cualquier número de *señales retenidas*. [2.4.4]

raya estado siguiente *(E: dash nextstate)*

Una *raya estado siguiente* es una *notación taquigráfica* que indica que el *estado siguiente* de la *instancia de proceso* es el *estado* en curso. [4.9]

real *(E: real)*

Real es un *género de datos predefinidos* para el cual los *valores* son los números que pueden representarse por un *entero* dividido por otro. Los *operadores* predefinidos para el *género real* tienen los mismos *nombres* que los *operadores* del *género entero*. [Anexo D]

referencia de paquete *(E: package reference)*

Una *referencia de paquete* hace definiciones dentro de un *paquete* visible a otro *paquete* o a un *sistema*. Resultan visibles las definiciones individuales o todas las definiciones listadas en la *interfaz de paquete* del *paquete*. [2.4.1.2]

refinamiento *(E: refinement)*

Refinamiento es la adición de nuevos detalles a la funcionalidad de un *nivel de abstracción* dado. El *refinamiento* de un *sistema* causa un enriquecimiento de su *comportamiento* o de sus capacidades para manejar más tipos de *señales* e información, incluidas las *señales* hacia o desde el *entorno*. Compárese con *partición*. [3.3]

reglas de formación correcta *(E: well-formedness rules)*

Las *reglas de formación correcta* son constricciones impuestas a una *sintaxis abstracta o sintaxis concreta* que obligan a cumplir condiciones estáticas no expresadas directamente por las reglas de sintaxis. [1.4.1, 1.4.2]

reglas léxicas *(E: lexical rules)*

Las *reglas léxicas* son reglas que definen la manera de formar *unidades léxicas* a partir de caracteres. [2.2.1, 4.2.1]

reinicializar; reponer *(E: reset)*

Operación definida para *temporizadores* que permite desactivarlos y elimina cualesquiera *señales de temporizador* pendientes del *puerto de entrada* del *proceso*. Véase *temporizador activo*. [2.8]

reponer

Véase reinicializar.

retorno *(E: return)*

Un *retorno* (de un *procedimiento*) es la transferencia de control al *procedimiento* o *proceso* que llama.

ruta de señales *(E: signal route)*

Una *ruta de señales* indica el flujo de *señales* entre un *tipo de proceso* y otro *tipo de proceso* en el mismo *bloque* o los *canales* conectados al *bloque*. [2.5.2]

salida *(E: output)*

Una *salida* es una *acción* dentro de una *transición* que genera una *instancia de señal*. [2.7.4]

selección *(E: selection)*

Acción por la cual se proporcionan los *sinónimos externos* necesarios para hacer una *especificación de sistema* específica partiendo de una *especificación de sistema* genérica. [4.3, 4.3.3, 4.3.4]

self; mismo *(E: self)*

self es una *expresión* del *género de datos predefinidos PId*. Cuando un *proceso* evalúa esta *expresión*, el resultado es el *valor PId* de ese *proceso*. **self** nunca produce como resultado el *valor Null*. Véanse también ***parent***, ***offspring***, *PId*. [2.4.4, 5.5.4.3]

semántica *(E: semantics)*

La *semántica* da significado a una entidad: las propiedades que tiene, la forma de interpretar su *comportamiento* y las condiciones dinámicas que deben cumplirse para que el *comportamiento* de la entidad satisfaga las reglas del *SDL*. [1.4.1, 1.4.2]

sender; emisor *(E: **sender**)*

sender es una *expresión PId*. Cuando se evalúa, **sender** da el *valor PId* del *proceso* emisor de la *señal* que activó la *transición* en curso. [2.4.4, 2.6.4, 5.5.4.3]

sentencia de asignación *(E: assignment statement)*

Una *sentencia de asignación* es una sentencia de una tarea, que cuando es interpretada asocia un *valor* a una *variable* que sustituye el *valor* anterior asociado con la *variable*. [5.4.3]

señal *(E: signal)*

Una *señal* es una instancia de un *tipo* de señal que comunica información a una *instancia de proceso*. [2.5.4]

señal continua *(E: continuous signal)*

Una *señal continua* es una *notación taquigráfica* que permite iniciar una transición cuando la condición *booleana* asociada se hace Verdadero. [4.11]

señal continua virtual *(E: virtual continuous signal)*

Una *señal continua virtual* es una *señal continua* en la que la *transición* puede ser redefinida en un *subtipo*. [4.11, 6.3.3]

señal retenida *(E: retained signal)*

Una *señal retenida* es una *señal* en el *puerto de entrada* de un *proceso*, es decir una *señal* que ha sido recibida pero no consumida por el *proceso*. [2.4.4]

SDL (Lenguaje de Especificación y Descripción del CCITT) *(E: SDL (CCITT Specification and Description Language))*

El *SDL (Lenguaje de Especificación y Descripción) del CCITT* es un lenguaje formal que proporciona un conjunto de construcciones para la *especificación* del *comportamiento* de un *sistema*.

SDL básico *(E: basic SDL)*

El *SDL básico* es el subconjunto del *SDL* definido en 2. [2]

SDL/GR *(E: SDL/GR)*

El *SDL/GR* es la representación gráfica en *SDL*. La *gramática* para *SDL/GR* está definida por la *gramática gráfica concreta* y la *gramática textual común*. [1.2]

SDL/PR *(E: SDL/PR)*

El *SDL/PR* es la representación de frase textual en *SDL*. La *gramática* del *SDL/PR* está definida por la *gramática textual concreta*. [1.2]

servicio *(E: service)*

Un *servicio* es una manera alternativa de especificar una parte del comportamiento de un *proceso*. Cada *servicio* puede definir un *comportamiento* parcial de un *proceso*. [2.4.5]

signatura *(E: signature)*

La *signatura* de un *tipo* es el conjunto de *géneros* y *operadores* para ese *tipo*. [5.2.1, 5.2.2]

signatura de género *(E: sort signature)*

Una *signatura de género* es un requisito de un *parámetro de contexto de género* formal en términos de *constricciones* impuestas a los *operadores* y literales del *género* en curso. [6.2.9]

signatura de operador *(E: operator signature)*

Una *signatura de operador* define el *género* (o los *géneros*) de los *valores* a los cuales puede aplicarse el *operador*, y el *género del valor* resultante. [5.2.2]

signatura de procedimiento *(E: procedure signature)*

Una *signatura de procedimiento* es un requisito de un *parámetro de contexto de procedimiento* en términos de *constricciones* impuestas a los *parámetros formales* del *procedimiento* efectivo. [6.2.2]

signatura de proceso *(E: process signature)*

Una *signatura de proceso* es un requisito de un *parámetro de contexto de proceso* formal en términos de *constricciones* impuestas a los *parámetros formales* de los procesos efectivos y al *conjunto de señales entradas válidas* del *proceso* efectivo. [6.2.1]

signatura de señal *(E: signal signature)*

Una *signatura de señal* es un requisito de un *parámetro de contexto de señal* formal en términos de *constricciones* impuestas a los *géneros* de los *parámetros* del *tipo de señal* efectivo. [6.2.4]

símbolo *(E: symbol)*

Un *símbolo* es un terminal en las *sintaxis concretas*. Un *símbolo* puede ser una de un conjunto de formas en la *sintaxis gráfica concreta*, o una secuencia de caracteres en la *sintaxis textual concreta*. [1.5.2, 1.5.3]

símbolo de ampliación de texto *(E: text extension symbol)*

Un *símbolo de ampliación de texto* es un contenedor de texto que pertenece al *símbolo gráfico* al cual está asociado el *símbolo de ampliación de texto*. El texto incluido en el *símbolo de ampliación de texto* sigue al texto del símbolo al cual está asociado. [2.2.7]

sinónimo *(E: synonym)*

Un *sinónimo* es un *nombre* que representa un *valor* de un *género*. [5.3.1.13]

sinónimo externo *(E: external synonym)*

Un *sinónimo externo* es un *género de datos predefinidos* cuyo *valor* no está especificado en la *especificación de sistema*. [4.3.1]

sintaxis abstracta *(E: abstract syntax)*

La *sintaxis abstracta* es el medio para describir la estructura conceptual de una *especificación SDL*. Las *sintaxis concretas* se hacen corresponder con la *sintaxis abstracta* para asegurar que *SDL/PR* y *SDL/GR* son equivalentes. [1.2]

sintaxis concreta *(E: concrete syntax)*

La *sintaxis concreta* para las diversas representaciones del *SDL* son los *símbolos* efectivos utilizados para representar el *SDL* y la interrelación entre *símbolos* requerida por las reglas sintácticas del *SDL*. Las dos *sintaxis concretas* utilizadas en Z.100 son la *sintaxis gráfica concreta* y la *sintaxis textual concreta*. [1.2]

sintaxis gráfica concreta *(E: concrete graphical syntax)*

La *sintaxis gráfica concreta* es la *sintaxis concreta* para la parte gráfica del *SDL/GR*. La *sintaxis gráfica concreta* se expresa en Z.100 empleando una forma ampliada de *BNF*. [1.2, 1.5.3]

sintaxis textual concreta *(E: concrete textual syntax)*

La *sintaxis textual concreta* es la *sintaxis concreta* para el *SDL/PR* y las partes textuales del *SDL/GR*. La *sintaxis textual concreta* se expresa en Z.100 empleando *BNF*. [1.2, 1.5.2]

sintipo *(E: syntype)*

Un *sintipo* especifica un conjunto de *valores* que corresponde a un subconjunto de los *valores* del *tipo*. Los *operadores* del *sintipo* son los mismos que los del *tipo* progenitor. [5.3.1.9]

sistema *E: system)*

Un *sistema* es un conjunto de *bloques* conectados entre sí y al *entorno* por *canales*. [2.4.2]

string

Véase cadena.

subbloque *(E: subblock)*

Un *subbloque* es un *bloque* contenido en otro *bloque*. Se forman *subbloques* cuando un *bloque* es *fraccionado*. [3.2.1, 3.2.2]

subcanal *(E: subchannel)*

Un *subcanal* es un *canal* formado cuando un *bloque* es *particionado*. Un *subcanal* conecta un *subbloque* a una frontera del *bloque particionado*, o un *bloque* a la frontera de un *canal particionado*. [3.2.2, 3.2.3]

subconjunto de partición consistente *(E: consistent partitioning subset)*

Un *subconjunto de partición consistente* es un conjunto de los *bloques* y *subbloques* en una *especificación de sistema* que proporciona una visión completa del *sistema* con partes conexas a un *nivel de abstracción* correspondiente. Así, cuando un *bloque* o *subbloque* está contenido en un *subconjunto de partición consistente*, sus ancestros y colaterales también lo están. [3.2.1]

subconjunto de refinamiento consistente *(E: consistent refinement subset)*

Un *subconjunto de refinamiento consistente* es un *subconjunto de partición consistente* que contiene todos los *bloques* y *subbloques* que utilizan las *señales* empleadas por cualquiera o cualesquiera de los *bloques* o *subbloques*. [3.3]

subestructura de bloque *(E: block substructure)*

Una *subestructura de bloque* es la *partición* del *bloque* en *subbloques* y nuevos *canales* a un *nivel de abstración* inferior. [3.2.2]

subestructura de canal *(E: channel substructure)*

Una *subestructura de canal* es una *partición* de un *canal* en un conjunto de *canales* y *bloques* a un *nivel de abstracción* inferior. [3.2.3]

subseñal *(E: subsignal)*

Una *subseñal* es un *refinamiento* de una *señal*. Una *subseñal* es una señal y puede ser a su vez *refinada*. [3.3]

subtipo *(E: subtype)*

Un *subtipo* es un *tipo* definido como una *especialización* de otro *tipo* (el *supertipo*). Las propiedades asociadas con un *subtipo* son todas las propiedades del *supertipo*, salvo las *virtuales* redefinidas, más las propiedades añadidas que son especiales para el *subtipo*. [1.3.1, 6.3]

supertipo *(E: supertype)*

Un *supertipo* es el *tipo* que se utiliza al definir un *subtipo*. [1.3.1, 6.3]

tarea *(E: task)*

Una *tarea* es una acción, dentro de una *transición*, que contiene una secuencia de *sentencias de asignación* o *texto informal*. [2.7.1]

temporizador *(E: timer)*

Un *temporizador* es un objeto, perteneciente a una *instancia de proceso*, que puede estar *activo* o *inactivo*. Un *temporizador activo* retorna una *señal de temporización* a la *instancia de proceso* propietaria en un instante especificado. Véanse también *inicializar* y *reinicializar*. [2.8, 5.4.4.5]

temporizador activo *(E: active timer)*

Un *temporizador activo* es un *temporizador* que tiene una *señal de temporizador* en el *puerto de entrada* del *proceso* propietario o está fijado para que produzca una *señal de temporizador* en cierto instante futuro. [2.8, 5.4.4.5]

término *(E: term)*

Un *término* es sintácticamente equivalente a una *expresión*. Los *términos* sólo se utilizan en *axiomas* y se distinguen de las *expresiones* por razones de claridad. [5.2.3, 5.3.3]

texto informal *(E: informal text)*

Texto informal es un texto incluido en una *especificación SDL* para el cual la *semántica* no está definida por el *SDL*, sino por algún otro modelo. Un *texto informal* se escribe entre apóstrofos. [2.2.3]

tiempo; time *(E: time)*

Tiempo es un *género de datos predefinidos* para el cual los *valores* se denotan como los *valores* de *real*. Los *operadores* predefinidos para los *géneros tiempo* y *duración* son + y –. [Anexo D]

time

Véase tiempo.

tipo *(E: type)*

Un *tipo* es un conjunto de propiedades de *instancias*. Como ejemplos de clases de *tipos* en *SDL* cabe citar *bloques*, *procesos*, *servicios*, *señales* y *sistemas*. [1.3.1]

tipo abstracto de datos *(E: abstract data type)*

Los *tipos abstractos de datos* definen los datos en términos de propiedades abstractas y no en términos de realización concreta. Un *tipo abstracto de datos* es una clase de tipo que define los conjuntos de *valores (géneros)*, un conjunto de *operadores* que se aplican a estos *valores* y un conjunto de reglas algebraicas *(ecuaciones)* que definen el *comportamiento* cuando se aplican los *operadores* a los *valores*. [2.3.1, 5.1]

tipo de bloque *(E: block type)*

Un *tipo de bloque* es la asociación de un *nombre* y un conjunto de propiedades que tendrán todas las *instancias de bloque* de este *tipo*. [6.1.1.2]

tipo de datos *(E: data type)*

Tipo de datos es sinónimo de *tipo abstracto de datos*.

tipo de proceso *(E: process type)*

Un *tipo de proceso* es la asociación de un *nombre* y un conjunto de propiedades que tendrán todas *las instancias de proceso* de ese *tipo de proceso*. [6.1.1.3]

tipo de servicio *(E: service type)*

Un *tipo de servicio* es la asociación de un *nombre* y un conjunto de propiedades que tendrán todos los *servicios* de ese *tipo de servicio*. [6.1.1.4]

tipo de sistema *(E: system type)*

Un *tipo de sistema* es la asociación de un *nombre* y un conjunto de propiedades que tendrán todas las *instancias de sistema* de este *tipo*. [6.1.1.1]

tipo parametrizado *(E: parameterized type)*

Un *tipo parametrizado* es un *tipo* en el que algunas de sus dependencias sobre las *unidades de ámbito* están representadas por medio de *parámetros de contexto*. [1.3.1]

tipo virtual *(E: virtual type)*

Un *tipo virtual* es un *tipo* que puede redefinirse en *subtipos* del tipo que lo encierra. [6.3.2]

transición *(E: transition)*

Una *transición* es una secuencia activa que se produce cuando una *instancia de proceso* pasa de un *estado* a otro. [2.6.8]

transición de entrada de procedimiento distante *(E: remote procedure input transition)*

Una *transición de entrada de procedimiento distante* indica un *estado* en el que un *proceso* ejecutará el proceso exportado especificado, seguido posiblemente de una *transición*. [4.14]

transición espontánea *(E: spontaneous transition)*

Una *transición espontánea* permite la interpretación de una *transición* sin ningún consumo de *señal* por el *proceso*. [2.6.6]

transición espontánea virtual *(E: virtual spontaneous transition)*

Una *transición espontánea virtual* es una *transición espontánea* en un *tipo* que puede redefinirse en un *subtipo*. [2.6.6, 6.3.3]

transición implícita *(E: implicit transition)*

En la *sintaxis concreta*, una *transición implícita* es iniciada por una *señal* perteneciente al *conjunto completo de señales de entrada válidas* y no especificada en una *entrada* o *conservación* del *estado*. Una *transición implícita* no contiene *acción* y hace retornar directamente al mismo *estado*. [4.8]

trayecto de comunicación *(E: communication path)*

Un *trayecto de comunicación* es un medio de transporte que vehicula *instancias de señal* desde una *instancia de proceso* o desde el *entorno* a otra *instancia de proceso* o al *entorno*. Un *trayecto de comunicación* comprende uno o más trayectos de *canal*, uno o más trayectos de *ruta de señales* o una combinación de ambas clases de trayectos. [2.7.4]

unidad de ámbito *(E: scope unit)*

En la *gramática concreta*, una *unidad de ámbito* define el intervalo de *visibilidad* de los *identificadores*. Ejemplos de *unidades de ámbito* incluyen las definiciones de *sistema*, *bloque*, *proceso*, *procedimiento*, *definiciones parciales de tipo* y *definiciones de servicio*. [2.2.2]

unidad léxica *(E: lexical unit)*

Las *unidades léxicas* son los *símbolos* terminales de la *sintaxis textual concreta*. [2.2.1]

valor *(E: value)*

Un *valor* de un *género* es uno de los valores que están asociados con una *variable* de ese *género*, y que puede utilizarse con un *operador* que requiere un *valor* de ese *género*. Un *valor* es el resultado de la interpretación de una *expresión*. [2.3.3, 5.1.3]

variable *(E: variable)*

Una *variable* es una entidad que pertenece a una *instancia* de *proceso* o *procedimiento* y que puede ser asociada a un *valor* mediante una *sentencia de asignación*. Cuando se *accede* a una *variable*, ésta da el último *valor* que se le ha asignado. [2.3.2]

variable exportada *(E: exported variable)*

Una *variable exportada* es una *variable* que puede utilizarse en una *operación de exportación*. Su *definición* contiene la palabra clave **exportado**. [2.6.1.1]

variable importada *(E: imported parameter)*

Una *variable importada* es una *variable* utilizada en una *expresión de importación*. [4.13]

vástago

Véase **offspring.**

vinculación *(E: binding)*

Vinculación asocia *parámetros efectivos* con *parámetros formales* (en la creación de un proceso o en una llamada de procedimiento) o *parámetros de contexto efectivos* con *parámetros de contexto* (formales). [1.3.1]

virtualidad *(E: virtuality)*

La *virtualidad* de un *tipo* o *transición* indica si el/la *tipo/transición* es un *tipo virtual*, redefinido en un *subtipo* (o en otro *tipo virtual*), o finalizado (que está redefinido, pero no es un *tipo virtual*). [6.3.2]

visibilidad *(E: visibility)*

La *visibilidad* de un *identificador* viene dada por las *unidades de ámbito* en que puede utilizarse. Dos definiciones en la misma *unidad de ámbito* y pertenecientes a la misma *clase de entidad* no pueden tener el mismo *nombre*. [2.2.2]

zona

Véase área.