UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Z.100

(11/99)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE Z: LENGUAJES Y ASPECTOS GENERALES DE SOPORTE LÓGICO PARA SISTEMAS DE TELECOMUNICACIÓN

Técnicas de descripción formal – Lenguaje de especificación y descripción

---

Lenguaje de especificación y descripción

Recomendación UIT-T Z.100

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES UIT-T DE LA SERIE Z

LENGUAJES Y ASPECTOS GENERALES DE SOPORTE LÓGICO PARA SISTEMAS DE TELECOMUNICACIÓN

TÉCNICAS DE DESCRIPCIÓN FORMAL	
Lenguaje de especificación y descripción (SDL)	**Z.100–Z.109**
Aplicación de técnicas de descripción formal	Z.110–Z.119
Gráficos de secuencias de mensajes	Z.120–Z.129
LENGUAJES DE PROGRAMACIÓN	
CHILL: el lenguaje de alto nivel del UIT-T	Z.200–Z.209
LENGUAJE HOMBRE-MÁQUINA	
Principios generales	Z.300–Z.309
Sintaxis básica y procedimientos de diálogo	Z.310–Z.319
LHM ampliado para terminales con pantalla de visualización	Z.320–Z.329
Especificación de la interfaz hombre-máquina	Z.330–Z.399
CALIDAD DE SOPORTES LÓGICOS DE TELECOMUNICACIONES	Z.400–Z.499
MÉTODOS PARA VALIDACIÓN Y PRUEBAS	Z.500–Z.599

Para más información, véase la Lista de Recomendaciones del UIT-T.

RECOMENDACIÓN UIT-T Z.100

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN

Resumen

Alcance – Objetivo

En esta Recomendación se define el Lenguaje de Especificación y Descripción (SDL, *specification and description language*) concebido para especificar y describir sin ambigüedades los sistemas de telecomunicaciones. El alcance del SDL se indica en la cláusula 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 para valores y objetos. La estructuración se basa en la descomposición jerárquica y jerarquías de tipos. Estos fundamentos del SDL se explican en las respectivas cláusulas principales de esta 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 1.2, pero este lenguaje suele ser adecuado para describir sistemas reactivos. Las aplicaciones posibles van desde la descripción de requisitos a la implementación de sistemas.

Estado/Estabilidad

Esta Recomendación es el manual de referencia completo de este lenguaje, acompañado por directrices de utilización en el Suplemento 1. El anexo F contiene una definición formal de semánticas SDL. El texto principal de esta Recomendación es estable y debe publicarse inmediatamente para satisfacer las exigencias del mercado, pero se requieren estudios adicionales a fin de completar el anexo F. En el apéndice I se recoge el estado de la Recomendación Z.100 y deberá actualizarse conforme se realicen nuevos estudios. Aunque se prevé que en el futuro se elaboren ampliaciones del lenguaje, el SDL-2000, tal como se define en esta Recomendación, debe cumplir las necesidades de la mayoría de los usuarios durante los próximos años. La versión actual se basa en la amplia experiencia existente en relación con el SDL y en las nuevas necesidades de los usuarios recientemente detectadas.

El texto principal está complementado por los anexos siguientes:

- Anexo A Índice de no terminales.
- Anexo B Reservado para uso futuro – El anexo B de la versión (03/93) ya no es válido.
- Anexo C Reservado para uso futuro – El anexo C de la versión (03/93) ya no está en vigor.
- Anexo D Datos predefinidos del SDL.
- Anexo E Reservado para ejemplos.
- Anexo F Definición formal (se requieren estudios adicionales para el SDL-2000).
- Apéndice I Estado de Z.100, documentos y Recomendaciones conexos.

– Apéndice II Directrices para el mantenimiento del SDL.

– Apéndice III Conversión sistemática de SDL-92 en SDL-2000.

La Recomendación UIT-T Z.100 tiene también publicado independientemente un suplemento:

– Suplemento I Metodología SDL+: Utilización de MSC y SDL (con ASN.1).

Trabajo asociado

En la Recomendación Q.65 se describe un método de utilización del SDL de acuerdo con ciertas normas. En la Recomendación Z.110 se describe una estrategia para introducir en las normas una técnica de descripción formal como el SDL. En http://www.sdl-forum.org pueden encontrarse referencias a otros documentos sobre SDL e información sobre la utilización del lenguaje en el sector industrial.

Antecedentes

Desde 1976 el CCITT y el UIT-T han recomendado varias versiones del SDL. Esta versión es una revisión de la Recomendación Z.100 (03/93) e incorpora el addendum 1 a la Recomendación Z.100 (10/96) y partes de la Recomendación Z.105 (03/95).

En comparación con el SDL definido en 1992, la versión contenida en este documento se ha ampliado en lo tocante a los datos orientados a objetos, la armonización de una serie de características que permiten que el lenguaje sea más simple y características para mejorar las posibilidades de utilización del SDL con otros lenguajes tales como ASN.1, ODL del UIT-T Z.130, CORBA y UML. Se han incluido algunos detalles. Se ha tenido cuidado de no invalidar los documentos SDL existentes; algunos cambios pueden necesitar descripciones para actualizar esta versión. En 1.5 se dan detalles de los cambios introducidos.

Orígenes

La Recomendación UIT-T Z.100, ha sido revisada por la Comisión de Estudio 10 (1997-2000) del UIT-T y fue aprobada por el procedimiento de la Resolución N.° 1 de la CMNT el 19 de noviembre de 1999.

Palabras clave

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

PREFACIO

La UIT (Unión Internacional de Telecomunicaciones) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones. El UIT-T (Sector de Normalización de las Telecomunicaciones de la UIT) es un órgano permanente de la UIT. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Conferencia Mundial de Normalización de las Telecomunicaciones (CMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución N.° 1 de la CMNT.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI.

NOTA

En esta Recomendación, la expresión "Administración" se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT no ha recibido notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB.

ÍNDICE

Página

Página

Recomendación Z.100

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN

(revisada en 1999)

1 Ámbito

La finalidad de recomendar el SDL (lenguaje de especificación y descripción) es proporcionar un lenguaje que permita una especificación y descripción inequívocas del comportamiento de sistemas de telecomunicaciones. Se pretende 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 Objetivos

Los objetivos generales a la hora de 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, ofertas y diseño, al tiempo que permite que algunos aspectos queden abiertos;

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

d) soporte diversas metodologías para la especificación y diseño de sistemas.

1.2 Aplicaciones

Esta Recomendación es el manual de referencia para el SDL. El suplemento 1 de la Recomendación Z.100 que se ha elaborado en el Periodo de Estudios 1992-1996 contiene un documento metodológico con ejemplos de utilización del SDL. El apéndice I de la Recomendación Z.100, que se publicó por primera vez en marzo de 1993, contiene asimismo directrices metodológicas, aunque éstas no explotan completamente todo el potencial del SDL.

El ámbito principal de aplicación del SDL es la especificación del comportamiento de aspectos de sistemas que funcionan en tiempo real y el diseño de tales sistemas. Entre estas aplicaciones en el campo de las telecomunicaciones 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 mediante 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 UIT-T, u otras Normas semejantes (internacionales, regionales o nacionales);

d) especificaciones de diseño de sistemas;

e) especificaciones detalladas;

f) descripciones de diseños de sistemas (tanto de alto nivel como suficientemente detallado como para generar directamente implementaciones);

g) descripciones de pruebas de sistemas (en particular en combinación con MSC y TTCN).

La organización del usuario puede elegir el nivel apropiado de aplicación del SDL.

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 transportan 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.

1.4 Diferencias entre SDL-88 y SDL-92

El lenguaje definido en versión anterior de la presente Recomendación era una ampliación de la Recomendación Z.100 publicada en el *Libro Azul* de 1988. El lenguaje definido en el *Libro Azul* se llama SDL-88 y el definido en la versión anterior de esta Recomendación se llama 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 UIT-T.

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 añadieron algunos constructivos de lenguaje para que el SDL-92 soporte más completa y uniformemente el paradigma de objeto:

a) lotes;

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

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

d) parametrización de tipos mediante parámetros de contexto;

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

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

En algunos casos se ha hecho necesario modificar el SDL-88 en aspectos en que éste carece de consistencia. 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 ha sido necesario para convertir un documento SDL-88 en una documento SDL-92 que contenía nombres consistentes en palabras que son palabras clave del SDL-92.

Para el constructivo **output** se simplificó la semántica con el cambio de SDL-88 a SDL-92, lo cual puede haber invalidado 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 cambiaron algunas propiedades de la propiedad de igualdad de géneros.

Para el constructivo **export/import** se introdujo una definición de variable remota optativa para armonizar la exportación de variables con la exportación de procedimientos (procedimiento remoto) introducida. Para ello se modificaron los documentos SDL-88 que tuvieran calificadores en expresiones de importación o se introdujeron 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 para convertir SDL-88 en SDL-92 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 se hizo localmente al proceso de visión o servicio. Esto exigió modificar los documentos SDL-88 que contuvieran calificadores en definiciones de visión o en expresiones de visión. La corrección para convertir SDL-88 en SDL-92 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 definió como un concepto primitivo, en lugar de una notación taquigráfica, sin ampliar sus propiedades. La utilización del servicio no se ve 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 suprimió el constructivo ruta de señales de servicio, y en su lugar pueden utilizarse rutas de señales. Éste es sólo un pequeño cambio conceptual, y no tiene repercusiones en la utilización concreta (las sintaxis de ruta de señales de servicio SDL-88 y de ruta de señales SDL-92 son las mismas).

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

Algunas de las definiciones del SDL básico se ampliaron considerablemente, como por ejemplo, la definición de **signal**, pero cabe observar que las ampliaciones eran optativas y sólo se debían 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.

1.5 Diferencias entre SDL-92 y SDL-2000

Se había tomado la decisión estratégica de mantener el SDL estable durante el periodo 1992-1996 de forma que al final de dicho periodo sólo se hubieran realizado cambios muy limitados al SDL. Dichos cambios fueron publicados como addendum 1 a la Recomendación Z.100 (10/96) en lugar de realizar una actualización del documento del SDL-92. Aunque en ocasiones se ha llamado a dicha versión SDL-96, sólo se trató de cambios pequeños en comparación con los cambios que hubo entre las versiones SDL-88 y SDL-92. Los cambios fueron los siguientes:

a) armonización de señales con procedimientos remotos y variables remotas;

b) armonización de canales y de rutas de señales;

c) adición de procedimientos y operaciones externos;

d) permitir que un bloque o un proceso se utilizaran como una sistema;

e) expresiones de estado;

f) permitir lotes sobre bloques y procesos;

g) operadores sin parámetros.

Dichos cambios se han incorporado ahora a Z.100, junto con otros cambios para producir la versión conocida como SDL-2000. En esta Recomendación se denomina SDL-92 al lenguaje definido por Z.100 (03/93) y el addendum 1 a la Recomendación Z.100 (10/96).

Las ventajas derivadas de la estabilidad del lenguaje que se ha mantenido desde 1992 a 1996 comienzan a ser superadas por la conveniencia de actualizar el SDL con miras al soporte y una mejor correspondencia con otros lenguajes que se utilizan frecuentemente en combinación con el SDL. Asimismo, las modernas técnicas y herramientas han permitido que sea práctico generar soporte lógico de forma más directa a partir de especificaciones SDL, pudiendo obtenerse sustanciales mejoras al incorporar una mejor soporte de todo ello en el SDL. Si bien el SDL-2000 es en gran medida una mejora del SDL-92, se ha acordado que se justifica una cierta incompatibilidad con el SDL-92, ya que de otra forma el lenguaje resultante hubiera sido excesivamente grande, demasiado complejo y demasiado inconsistente. En esta subcláusula se proporciona información sobre los cambios. En el apéndice III se presenta como la mayoría de las descripciones en SDL-92 pueden transformarse sistemáticamente en SDL-2000.

Se han realizado cambios en una serie de áreas que se enfocan hacia una simplificación del lenguaje y hacia su adaptación a nuevas áreas de aplicación:

a) ajuste de los convenios sintácticos a otros lenguajes que se utilizan junto con el SDL;

b) armonización de los conceptos de sistema, bloque y proceso basados en "agente", y la fusión del concepto de ruta de señal con el concepto de canal;

c) descripciones de interfaces;

d) tratamiento de excepciones;

e) soporte de notación textual de algoritmos en SDL/GR;

f) estados compuestos;

g) sustitución del constructivo servicio por el constructivo agregación de estado;

h) nuevo modelo para los datos;

i) constructivos para soportar la utilización de ASN.1 con SDL previamente en la Recomendación Z.105 (03/95).

Otros cambios son los siguientes: lotes anidados, posibilidad de que los bloques contengan directamente otros bloques y procesos, parámetros sólo **out**.

A nivel sintáctico, el SDL-2000 es sensible a la escritura en mayúscula/minúscula. Las palabras claves están disponibles de dos formas: con todo en mayúsculas y con todo en minúsculas. Las palabras claves del SDL-2000 que no eran palabras claves en SDL-92 son las siguientes:

abstract, aggregation, association, break, choice, composition, continue, endexceptionhandler, endmethod, endobject, endvalue, exception, exceptionhandler, handle, method, object, onexception, ordered, private, protected, public, raise, value.

Las siguientes palabras claves de SDL-92 no son palabras claves del SDL-2000:

all, axioms, constant, endgenerator, endnewtype, endrefinement, endservice, error, fpar, generator, imported, literal, map, newtype, noequal, ordering, refinement, returns, reveal, reverse, service, signalroute, view, viewed.

Un número reducido de construcciones de SDL-92 no están disponibles en SDL-2000: expresiones de visión, generadores, subestructuras de bloques, subestructuras de canales, refinamiento de la señal, definición axiomática de datos y macrodiagramas. Estas construcciones han sido utilizadas muy escasamente (si es que lo fueron alguna vez), no estando justificado el coste adicional de mantenerlas en el lenguaje y en las herramientas.

2 Referencias

Las siguientes Recomendaciones del UIT-T y otras referencias contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones de la presente Recomendación. Al efectuar esta publicación, estaban en vigor las ediciones indicadas. Todas las Recomendaciones y otras referencias son objeto de revisiones por lo que se preconiza que los usuarios de esta Recomendación investiguen la posibilidad de aplicar las ediciones más recientes de las Recomendaciones y otras referencias citadas a continuación. Se publica periódicamente una lista de las Recomendaciones UIT-T actualmente vigentes.

– Recomendación CCITT T.50 (1992), Alfabeto internacional de referencia (anteriormente alfabeto internacional N.° 5 o IA5) – Tecnología de la información – Juego de caracteres codificado de siete bits para intercambio de información.

  ISO/CEI 646:1991, *ISO 7-bit coded character set for information interchange.*

3 Definiciones

En esta Recomendación se definen numerosos términos por lo que una enumeración de los mismos en esta cláusula daría lugar a la repetición de buena parte del texto de esta Recomendación. Por lo tanto, en esta cláusula sólo se presentan unos pocos términos.

3.1 agente: El término agente se utiliza para denotar un sistema, bloque o proceso que contiene una o más máquinas de estado finitas ampliadas.

3.2 bloque: Un bloque es un agente que contiene uno o más bloques o procesos concurrentes y puede asimismo contener una máquina de estados finitos ampliada que dispone y maneje datos en el bloque.

3.3 **cuerpo**: Un cuerpo es un gráfico de máquina de estados de un agente, servicio, procedimiento, estado compuesto u operación.

3.4 **canal**: Un canal es un trayecto de comunicación entre agentes.

3.5 **entorno**: El entorno de una sistema es todo le que rodea y se comunica con el sistema utilizando SDL.

3.6 **puerta**: Una puerta representa un punto de conexión para la comunicación con un tipo de agente, y cuando dicho tipo se ejemplifica, determina la conexión del ejemplar del agente con otros ejemplares.

3.7 **ejemplar**: Un ejemplar es un objeto creado cuando se ejemplifica un tipo.

3.8 **objeto**: El término objeto se utiliza para ítems de datos que constituyen referencia de valores.

3.9 **pid**: El término pid se utiliza para elementos de datos que constituyen referencias a agentes.

3.10 **procedimiento**: Un procedimiento es una encapsulación de parte del comportamiento de un agente, que se define en un lugar pero que puede ser llamada desde varios lugares dentro del agente. Otros agentes pueden llamar a un procedimiento remoto.

3.11 **proceso**: Un proceso es un agente que contiene una máquina de estados finitos ampliada y que puede contener otros procesos.

3.12 **señal**: La forma de comunicación primaria es mediante señales que son enviadas por el agente de transmisión y recibidas por el agente de recepción.

3.13 **género**: Un género es un conjunto de ítems de datos con propiedades comunes.

3.14 **estado**: Una máquina de estados finitos ampliada de un agente se encuentra en un estado si está esperando un estímulo.

3.15 **estímulo**: Un estímulo es un evento que causa que un agente que se encuentra en un estado pase a una transición.

3.16 **sistema**: Un sistema es el agente más externo que se comunica con el entorno.

3.17 **temporizador**: Un temporizador es un objeto propiedad de un agente y que hace que en un instante de tiempo especificado ocurra un estímulo de señal de temporizador.

3.18 **transición**: Una transición es una secuencia de acciones que realiza un agente hasta que pasa a un estado.

3.19 **tipo**: Un tipo es una definición que puede utilizarse para la creación de ejemplares y que también puede ser heredada y especializarse para formar otros tipos. Un tipo parametrizado es un tipo que tiene parámetros. Cuando a estos parámetros se les da diferentes parámetros reales, se definen diferentes tipos no parametrizados que cuando son ejemplificados dan lugar a ejemplares con propiedades diferentes.

3.20 **valor**: El término valor se utiliza para la clase de datos a los que se accede directamente. Los valores se pueden pasar libremente entre agentes.

4 Abreviaturas

En esta Recomendación se utilizan las siguientes siglas.

SDL/GR Forma de representación gráfica de SDL

SDL/PR Forma de representación de frase textual de SDL

SDL-2000 SDL definido en esta Recomendación

SDL-92 SDL definido en la Recomendación Z.100 (03/93) y su addendum 1 (10/96)

SDL-88 SDL definido en la Recomendación Z.100 (1988)

5 Convenios

El texto de esta cláusula no es normativo. Sólo define los convenios empleados para describir el SDL. La utilización del SDL en esta cláusula sólo es ilustrativa. Los metalenguajes y convenios introducidos sólo lo son con fines de procurar una descripción inequívoca del SDL.

5.1 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.

Aunque el SDL puede escribirse en SDL/PR o SDL/GR, el lenguaje ha sido diseñado teniendo en cuenta que el SDL/PR se utiliza con poca frecuencia para fines tales como el intercambio entre herramientas. El formato de intercambio común que se define en Z.106 (10/96) minimiza aún más la utilización de SDL/PR. La mayoría de los usuarios emplean el SDL/GR.

La figura 5-1 muestra las relaciones entre SDL/PR, SDL/GR, las gramáticas concretas y la gramática abstracta.

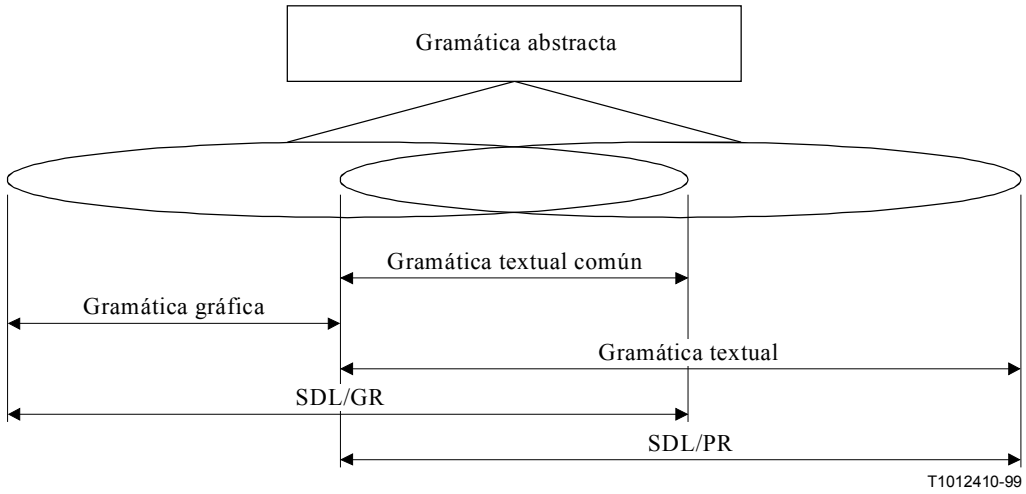


Figura 5-1/Z.100 – Gramáticas SDL

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 mediante 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.

Para algunos constructivos no existe una sintaxis abstracta equivalente directa. En tales casos, se provee un modelo para la transformación desde una sintaxis concreta a la sintaxis concreta de otros constructivos que (directamente o indirectamente a través de modelos adicionales) disponen de una sintaxis abstracta. Los ítems que no se pueden hacer corresponder con la sintaxis abstracta (como los comentarios) no tienen un significado formal.

5.2 Definiciones básicas

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

5.2.1 Definición, tipo y ejemplar

En esta Recomendación, los conceptos de tipo y de ejemplar y sus relaciones son fundamentales. Se utilizan el esquema y la terminología definidos a continuación y representados en la figura 5-2.

En esta subcláusula se presenta la semántica básica de definiciones de tipo, definiciones de ejemplar, definiciones de tipos parametrizados, parametrización, vinculación de parámetros de contexto, especialización y ejemplificación.

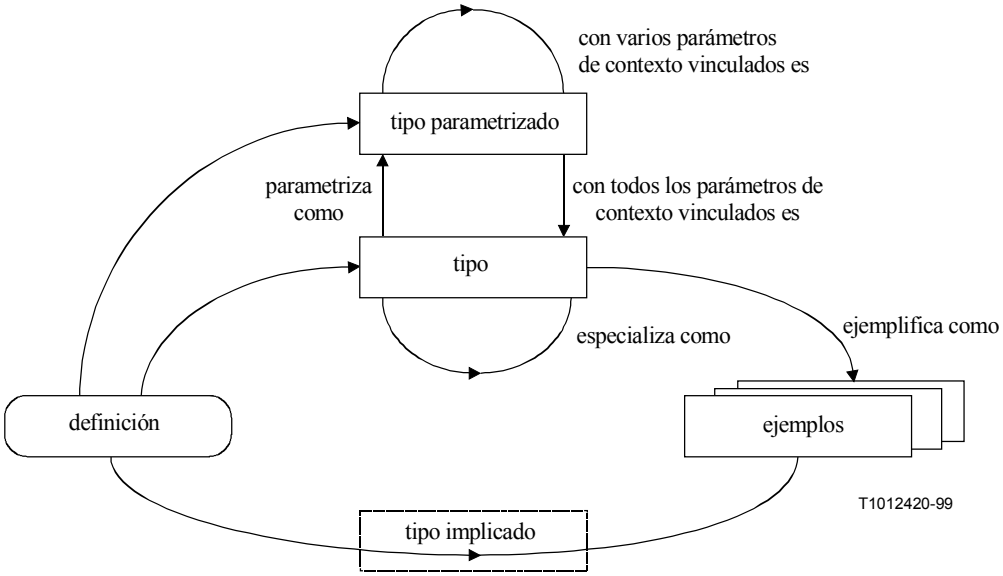


Figura 5-2/Z.100 – Concepto de tipo

Las definiciones presentan entidades denominadas que son tipos o ejemplares con tipos implicados. Una definición de un tipo define todas las propiedades asociadas con ese tipo. Un ejemplo de definición de ejemplar es una definición de proceso. Un ejemplo de definición de tipo es una definición de señal.

Un tipo puede ser ejemplificado en varios ejemplares. Un ejemplar de un tipo particular tiene todas las propiedades definidas para ese tipo. Un ejemplo de tipo es un procedimiento, que puede ser ejemplificado 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.

Un ejemplar se define directamente o mediante la ejemplificación de un tipo. Un ejemplo de ejemplar es un ejemplar de sistema que puede ser definido por una definición de sistema o ser una ejemplificación 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.

NOTA – Para evitar textos recargados, el término *ejemplar* puede omitirse. Por ejemplo "un sistema se interpreta..." significa "un ejemplar de sistema se interpreta...".

5.2.2 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 estos ejemplares de proceso. Estos procesos tienen valores pid que pueden distinguirse de cualquier valor pid dentro del sistema (véase 12.1.6).

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

5.2.3 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. Las excepciones predefinidas (véase D.3.16) se generarán si durante la interpretación de un sistema se encuentra un error. Si no se maneja la excepción, el comportamiento posterior del sistema no puede ser derivado de la especificación.

5.3 Estilo de presentación

El estilo de presentación siguiente se utiliza para agrupar los distintos aspectos del lenguaje en temas.

5.3.1 División del texto

La Recomendación está organizada por temas descritos en una introducción opcional seguida por ítems 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 6.5).

d) *Semántica* – Da significado a un 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 que no tienen una sintaxis abstracta directa y modelada en términos de otros constructivos de sintaxis concreta. Una notación que se modela mediante otros constructivos se denomina notación taquigráfica y se considera que es una sintaxis derivada de la forma transformada.

f) *Ejemplos*

En unos pocos casos (tales como <sdl specification>) se utiliza un ítem de enumeración titulado, *Gramática concreta*, cuando se fusionan la sintaxis gráfica y textual o cuando la mayoría de las reglas gramaticales deben repetirse para los casos textuales y gráficos. En esta Recomendación no se define el mecanismo de fusión de SDL/PR y SDL/GR para tales casos.

5.3.2 Ítems 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 esta Recomendación, presentada solamente para facilitar la comprensión y no para completar esta 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 ítems 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 5.4.1.

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 cláusula numerada de texto.

Se podrá hacer referencia a las reglas de la sintaxis abstracta a partir de cualesquiera ítems 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 un ejemplar. 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 5.4.2.

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 un ejemplar. 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 (véase 5.4.2) son significativas.

La sintaxis textual concreta que no es una forma taquigráfica 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 5.4.3.

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 un ejemplar. 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 termina en la gramática concreta por la palabra "diagrama" y hay un nombre en la gramática abstracta que sólo difiere en que termina 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 (7.3), macros (6.2), debe considerarse antes que la correspondencia entre la sintaxis concreta y la abstracta. Estas expansiones se describen detalladamente en el anexo F.

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 válidas 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*".

Todos los ejemplares 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 sección general sobre identidades en 6.3. 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" encerrados procesos y/o bloques.

Las propiedades son estáticas si pueden determinarse sin interpretación de una especificación de sistema SDL, y son dinámicas si se requiere una interpretación de ellas para determinar la propiedad.

La interpretación se describe de una manera operacional. Toda lista en la sintaxis abstracta deberá interpretarse en el orden dado. Esto es, esta Recomendación describe cómo se crean los ejemplares a

partir de la definición de sistema y cómo estos ejemplares se interpretan en una "máquina SDL abstracta". En la sintaxis abstracta las listas se identifican mediante los sufijos "*" y "+" (véanse 5.4.1 y 5.4.2).

Son condiciones dinámicas aquéllas que deben cumplirse durante la interpretación y no pueden verificarse sin interpretación. Las condiciones dinámicas pueden conducir a errores (véase 5.2.3).

NOTA – El comportamiento del sistema se produce "interpretando" el SDL. La palabra "interpretación" se elige explícitamente (más que una alternativa como por ejemplo "ejecutado") para incluir la interpretación mental de un ser humano y la interpretación del SDL que realiza una computadora.

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.

Las propiedades de una notación taquigráfica se derivan de la forma en la que se modela en términos de (o se transforma en) los conceptos primitivos. Para garantizar la utilización fácil e inequívoca de las notaciones taquigráficas y para reducir los efectos colaterales cuando se combinan varias notaciones taquigráficas, estos conceptos se transforman en un orden especial tal como se define en el anexo F. También se sigue el orden de transformación cuando en esta cláusula se definen los conceptos.

El resultado de la transformación de un fragmento de texto en la sintaxis concreta derivada es normalmente otro fragmento de texto en sintaxis concreta derivada o un fragmento de texto en sintaxis concreta. El resultado de la transformación puede también quedar vacío. En este último caso, se suprime el texto original de la especificación. Cuando se discute el modelo de una sintaxis concreta, el metasímbolo ***transform*** puede utilizarse y se refiere al resultado de la transformación de su argumento. Por ejemplo,

<expression>-***transform***

se refiere a la transformación de <expression>.

Ejemplos

El ítem de enumeración titulado *Ejemplo(s)* contiene uno o varios ejemplos. En el anexo E se facilitan ejemplos adicionales. Los ejemplos son informativos.

5.4 Metalenguajes

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

La gramática incluida en esta Recomendación se ha escrito para facilitar la presentación en la Recomendación de manera que los nombres de las reglas tengan sentido en el contexto en que se dan y puedan utilizarse en texto. Esto significa que cierto número de ambigüedades pueden resolverse fácilmente mediante una reescritura sistemática de las reglas de sintaxis o la aplicación de reglas semánticas.

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

5.4.1 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 canal es:

Channel-path :: *Originating-gate*
Destination-gate
*Signal-identifier****-set***

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

La definición Meta IV

Agent-identifier = *Identifier*

expresa que un *Agent-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 naturales

Ejemplo:

Number-of-instance :: *Nat* [*Nat*]

Number-of-instances denota un dominio compuesto que contiene un valor natural (Nat) obligatorio y un natural opcional ([Nat]) que denotan respectivamente el número inicial y el número máximo opcional de ejemplares.

b) Objetos citación

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

Ejemplo:

Channel-definition :: *Channel-name*
[**NODELAY**]
*Channel-path****-set***

Un canal puede no encontrarse retrasado, en cuyo caso, ello se denota mediante citación opcional **NODELAY**.

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 (constructivos) en BNF (véase 5.4.2) se utilizan también en la sintaxis abstracta: "*" para una lista que puede estar vacía, "+" para una lista no vacía, "|" para una 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:

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

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

5.4.2 BNF

En la forma Backus-Naur (BNF, *Backus-Naur form*) para reglas léxicas, los terminales son <space> y los caracteres impresos en 6.1.

En la forma Backus-Naur para reglas no léxicas, un símbolo terminal es una de las unidades léxicas definidas en 6.1 (<name>, <quoted operation name>, <character string>, <hex string>, <bit string>, <special>, <composite special> o <keyword>). En caso de reglas no léxicas, un terminal puede representarse mediante uno de los siguientes elementos:

a) una palabra clave (tal como estado);

b) el carácter de la unidad léxica si ésta consta de un único carácter (tal como "=");

c) el nombre de la unidad léxica (tal como <quoted operation name> o <bit string>);

d) el nombre de una unidad léxica <composite special> (tal como <implies sign>).

Para evitar la confusión con la gramática BNF, los nombres de las unidades léxicas <asterisk>, <plus sign>, <vertical line>, <left square bracket>, <right square bracket>, <left curly bracket> y <right curly bracket> se utilizan siempre en lugar de los caracteres equivalentes. Nótese que los dos terminales especiales <name> y <character string> pueden tener también semántica reforzada tal como se define posteriormente.

Los paréntesis angulares y la palabra o palabras encerradas son, bien un símbolo no terminal o una de las unidades léxicas. 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 o en gramática gráfica. Por ejemplo,

<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 <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,

<diagram in package> ::=
| <package diagram>
| <package reference area>
| <type in agent area>
| <data type reference area>
| <signal reference area>
| <procedure reference area>
| <interface reference area>
| <create line area>
| <option area>

expresa que un <diagram in package> es un <package diagram>, o un <package reference area>, o un <type in agent area>, o un <data type reference area>, o una <signal reference area>, o un <procedure reference area>, o una <interface reference area>, o una <create line area> o una <option area>.

Los elementos sintácticos pueden agruparse utilizando llaves ("{" y "}"), similares a los paréntesis en Meta IV (véase 5.4.1). Un grupo encerrado entre llaves puede contener una o más barras verticales, que indican elementos sintácticos alternativos. Por ejemplo,

```
<operation definitions> ::=
                              {     <operation definition>
                              |     <textual operation reference>
                              |     <external operation definition> }+
```

La repetición de elementos sintácticos o 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 <operation definitions> contiene por lo menos un <operation definition> o <textual operation reference> o <external operation definition>, y puede contener más de uno de ellos.

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>.

5.4.3 Metalenguaje para gramática gráfica

Para la gramática gráfica el metalenguaje descrito en 5.4.2 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 ítems. Cada ítem puede ser un grupo cualquiera de elementos sintácticos, en cuyo caso debe ser ampliado antes de aplicar el metasímbolo ***set***.

Ejemplo:

{ <operation text area>* <operation graph area> } ***set***

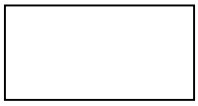
es un conjunto que puede tener alguna <operation text area>s, y una <operation graph area>. El metasímbolo ***set*** se utiliza cuando la posición relativa de los elementos sintácticos en el diagrama no es relevante y los elementos pueden considerarse en cualquier orden.

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".

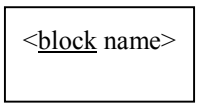
El metasímbolo ***contains*** indica que su argumento de la derecha debe situarse dentro de su argumento de la izquierda y del <text extension symbol> asociado, si existe. Por ejemplo,

<block reference area> ::=
<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 e implica un flow line symbol (véase 6.5).

6 Reglas generales

6.1 Reglas léxicas

Las reglas léxicas definen unidades léxicas. Las unidades léxicas son los símbolos terminales de la *Gramática textual concreta* y de la *Gramática gráfica concreta*.

<lexical unit> ::=
<name>
| <quoted operation name>
| <character string>
| <hex string>
| <bit string>
| <note>
| <composite special>
| <special>
| <keyword>

<name> ::=
<underline>* <word> {<underline>+ <word>}* <underline>*
| {<decimal digit>}+ { {<full stop>} <decimal digit>+ }*

<word> ::=
{<alphanumeric>}+

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

<letter> ::=
<uppercase letter> | <lowercase letter>

<uppercase 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

<lowercase 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

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

<quoted operation name> ::=
<quotation mark> <infix operation name> <quotation mark>
| <quotation mark> <monadic operation name> <quotation mark>

<infix operation name> ::=
or | **xor** | **and** | **in** | **mod** | **rem**
| <plus sign> | <hyphen>
| <asterisk> | <solidus>
| <equals sign> | <not equals sign>
| <greater than sign> | <less than sign>
| <less than or equals sign> | <greater than or equals sign>
| <concatenation sign> | <implies sign>

<monadic operation name> ::=
<hyphen> | **not**

<character string> ::=
<apostrophe> { <general text character>
| <special>
| <apostrophe> <apostrophe>
}* <apostrophe>

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

<hex string> ::=
<apostrophe> { <decimal digit>
| a | b | c | d | e | f
| A | B | C | D | E | F
}* <apostrophe> { H | h }

<bit string> ::=
<apostrophe> { 0 | 1
}* <apostrophe> { B | b }

<note> ::=
<solidus> <asterisk> <note text> <asterisk>+ <solidus>

<note text> ::=
{ <general text character>
| <other special>
| <asterisk>+ <not asterisk or solidus>
| <solidus>
| <apostrophe> }*

<not asterisk or solidus> ::=
<general text character> | <other special> | <apostrophe>

<text> ::=
{ <general text character> | <special> | <apostrophe> }*

<general text character> ::=
<alphanumeric> | <other character> | <space>

<composite special> ::=
| <result sign>
| <composite begin sign>
| <composite end sign>
| <concatenation sign>
| <history dash sign>
| <greater than or equals sign>
| <implies sign>
| <is assigned sign>
| <less than or equals sign>
| <not equals sign>
| <qualifier begin sign>
| <qualifier end sign>

<result sign> ::=
<hyphen> <greater than sign>

<composite begin sign> ::=
<left parenthesis> <full stop>

<composite end sign> ::=
<full stop> <right parenthesis>

<concatenation sign> ::=
<solidus> <solidus>

<history dash sign> ::=
<hyphen> <asterisk>

<greater than or equals sign> ::=
<greater than sign> <equals sign>

<implies sign> ::=
<equals sign><greater than sign>

<is assigned sign> ::=
<colon> <equals sign>

<less than or equals sign> ::=
<less than sign> <equals sign>

<not equals sign> ::=
<solidus> <equals sign>

<qualifier begin sign> ::=
<less than sign> <less than sign>

<qualifier end sign> ::=
<greater than sign> <greater than sign>

<special> ::=
<solidus> | <asterisk> | <other special>

<other special> ::=
<exclamation mark> | <number sign>
| <left parenthesis> | <right parenthesis>
| <plus sign> | <comma> | <hyphen>
| <full stop> | <colon> | <semicolon>
| <less than sign> | <equals sign> | <greater than sign>
| <left square bracket> | <right square bracket>
| <left curly bracket> | <right curly bracket>

<other character> ::=
<quotation mark> | <dollar sign> | <percent sign>
| <ampersand> | <question mark> | <commercial at>
| <reverse solidus> | <circumflex accent> | <underline>
| <grave accent> | <vertical line> | <tilde>

<exclamation mark> ::= **!**

<quotation mark> ::= **"**

<left parenthesis> ::= **(**

<right parenthesis> ::= **)**

<asterisk> ::= *

<plus sign> ::= +

<comma> ::= **,**

<hyphen> ::= -

<full stop> ::= **.**

<solidus> ::= /

<colon> ::= **:**

<semicolon> ::= **;**

<less than sign> ::= **<**

<equals sign> ::= **=**

<greater than sign> ::= **>**

<left square bracket> ::= **[**

<right square bracket> ::= **]**

<left curly bracket> ::= **{**

<right curly bracket> ::= **}**

<number sign> ::= **#**

<dollar sign> ::= **$**

<percent sign> ::= **%**

<ampersand> ::= **&**

<apostrophe> ::= **'**

<question mark> ::= **?**

<commercial at> ::= **@**

<reverse solidus> ::= \

<circumflex accent> ::= ^

<underline> ::= _

<grave accent> ::= `

<vertical line> ::= |

<tilde> ::= ~

<keyword>::=

abstract	**active**	**adding**
aggregation	**alternative**	**and**
any	**as**	**association**
atleast	**block**	**break**
call	**channel**	**choice**
comment	**composition**	**connect**
connection	**constants**	**continue**
create	**dcl**	**decision**
default	**else**	**endalternative**
endblock	**endchannel**	**endconnection**
enddecision	**endexceptionhandler**	**endinterface**
endmacro	**endmethod**	**endobject**
endoperator	**endpackage**	**endprocedure**
endprocess	**endselect**	**endstate**
endsubstructure	**endsyntype**	**endsystem**
endtype	**endvalue**	**env**
exception	**exceptionhandler**	**export**
exported	**external**	**fi**
finalized	**for**	**from**
gate	**handle**	**if**
import	**in**	**inherits**
input	**interface**	**join**
literals	**macro**	**macrodefinition**
macroid	**method**	**methods**
mod	**nameclass**	**nextstate**
nodelay	**none**	**not**
now	**object**	**offspring**
onexception	**operator**	**operators**
optional	**or**	**ordered**
out	**output**	**package**
parent	**priority**	**private**

| **procedure** | **protected** | **process**
| **provided** | **public** | **raise**
| **redefined** | **referenced** | **rem**
| **remote** | **reset** | **return**
| **save** | **select** | **self**
| **sender** | **set**
| **signal** | **signallist** | **signalset**
| **size** | **spelling** | **start**
| **state** | **stop** | **struct**
| **substructure** | **synonym** | **syntype**
| **system** | **task** | **then**
| **this** | **timer** | **to**
| **try** | **type** | **use**
| **value** | **via** | **virtual**
| **with** | **xor**

<space> ::=

Los caracteres de <lexical unit>s y de <note>s así como el carácter <space> y los caracteres de control se definen en la versión internacional de referencia del alfabeto de referencia internacional (Recomendación T.50). La unidad lexical <space> representa el carácter ESPACIO de T.50 (acrónimo SP), que (por razones obvias) no puede mostrarse.

<text> se utiliza en un <comment area> donde es equivalente a <character string> y en un <text extension area> donde debe tratarse como una secuencia de otras unidades léxicas.

Los caracteres de supresión de T.50 se ignoran por completo. Si se utiliza un conjunto de caracteres ampliado, los caracteres que no se definen en T.50 sólo pueden aparecer en el <text> en una <comment area> o en una <character string> en un <comment> o dentro de una <note>.

Cuando a un carácter <underline> le siguen uno o más <space>s o caracteres de control, se ignoran todos los caracteres (incluido <underline>); por ejemplo, A_ B denota el mismo <name> que AB. Esta utilización de <underline> permite que las <lexical unit>s se extiendan a lo largo de más de una línea. Esta regla se aplica antes que cualquier otra regla léxica.

Un carácter de control (no espacio) puede aparecer donde asimismo pueda aparecer un <space>, y tiene el mismo significado que <space>.

La ocurrencia de un carácter de control no es significativa en un <informal text> y en una <note>. A fin de construir una expresión de cadena que contenga caracteres de control, debe utilizarse el operador <concatenation sign> y los literales para caracteres de control. Todos los espacios de una cadena de caracteres son significativos: una secuencia de espacios no se trata como un espacio.

Antes o después de una <lexical unit> puede insertarse cualquier número de <space>s. Los <spaces> o las <note>s insertadas no tienen relevancia sintáctica, pero algunas veces se necesita de un <space> o una <note> para separar entre sí las <lexical unit>.

En todas las <lexical unit>s, excepto las palabras clave, se hace distinción entre las <letter>s en mayúsculas y la que están en minúsculas. Por lo tanto, AB, aB, Ab y ab representan cuatro <word>s distintas. Una <keyword> con todo en mayúsculas se utiliza de la misma forma que la <keyword> con todo en minúsculas, con la misma ortografía (sin tener en cuenta si son mayúsculas o no) pero una secuencia de letras en la que se mezclan mayúsculas y minúsculas con la misma ortografía con la que una <keyword> representa una <word>.

Con el fin de que las reglas léxicas *gramática textual completa* y *gramática gráfica concreta* sean concisas, la <keyword> en minúsculas terminal denota la <keyword> en mayúsculas con las mismas letras y puede utilizarse en el mismo lugar. Por ejemplo, el terminador de sintaxis concreto

endblock

representa las alternativas léxicas

{ **endblock** | **ENDBLOCK** }

NOTA – En esta Recomendación, las letras minúsculas en negrita se utilizan como palabras clave. No es un requisito obligatorio hacer distinción entre atributos de los tipos de letras, aunque puede ser de utilidad para los lectores de una especificación.

Una <lexical unit> se termina con el primer carácter que no puede ser parte de la <lexical unit> conforme a la sintaxis antes especificada. Si una <lexical unit> puede ser tanto un <name> como una <keyword>, entonces se trata de una <keyword>. Si la única diferencia entre dos <quoted operation name> es que están en mayúsculas o en minúsculas, se aplica la semántica del nombre en minúsculas, de tal forma que (por ejemplo) la expresión "OR" (a,b) tiene el mismo significado que (a **o** b).

Las reglas léxicas especiales se aplican dentro de un <macro body>.

6.2 Macros

Una definición macro contiene una colección de unidades léxicas que pueden incluirse en uno o más lugares de la gramática textual completa de una <sdl specification>. Cada uno de dichos 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.

6.2.1 Reglas léxicas adicionales

<formal name> ::=
[<name>**%**] <macro parameter>
{ [**%**<name>] **%**<macro parameter> }*
[**%**<name>]

6.2.2 Definición de macro

<macro definition> ::=
macrodefinition <macro name>
[<macro formal parameters>] <end>
<macro body>
endmacro [<macroname>] <end>

<macro formal parameters> ::=
(<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**.

Una <macro definition> contiene unidades léxicas.

El <macro name> es visible en la totalidad de la definición de sistema, cualquiera que sea el lugar donde aparezca la definición de macro. 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 reemplaza 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 (alfa, c, s, a);
     block alfa referenced;
     channel c from a to alfa with s; endchannel c;
endmacro Exam;
```

6.2.3 Llamada a macro

<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>.

Modelo

Una <sdl specification> puede contener definiciones de macro y llamadas a macro. Antes de que puedan analizarse dicha <sdl specification>, 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 se expande para reemplazar la llamada a macro. Esto significa que se crea una copia del cuerpo de la 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>.

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 6.2.2, da el siguiente resultado.

```
.........
block A referenced;
block B referenced;
channel C1 from A to B with S1; endchannel C1;
.........
```

6.3 Reglas de visibilidad, nombres e identificadores

Gramática abstracta

Identifier :: *Qualifier Name*
Qualifier = *Path-item* +
Path-item = *Package-qualifier*
| *Agent-type-qualifier*
| *Agent-qualifier*
| *State-type-qualifier*
| *State-qualifier*
| *Data-type-qualifier*
| *Procedure-qualifier*
| *Signal-qualifier*
| *Interface-qualifier*
Package-qualifier :: *Package-name*
Agent-type-qualifier :: *Agent-type-name*
Agent-qualifier :: *Agent-name*
State-type-qualifier :: *State-type-name*
State-qualifier :: *State-name*
Data-type-qualifier :: *Data-type-name*
Procedure-qualifier :: *Procedure-name*
Signal-qualifier :: *Signal-name*
Interface-qualifier :: *Interface-name*
Package-name = *Name*
Agent-type-name = *Name*
Agent-name = *Name*
State-type-name = *Name*
Data-type-name = *Name*
Interface-name = *Name*
Name :: *Token*

Gramática textual concreta

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

<qualifier> ::=
<qualifier begin sign> <path item> { / <path item> }* <qualifier end sign>

<string name> ::=
<character string>
| <bit string>
| <hex string>

<path item> ::=
[<scope unit kind>] <name>

<scope unit kind> ::=
package
| **system type**
| **system**
| **block**
| **block type**
| **process**
| **process type**
| **state**
| **state type**
| **procedure**
| **signal**
| **type**
| **operator**
| **method**
| **interface**

Las unidades de ámbito se definen mediante los símbolos no terminales siguientes de la gramática concreta. Los símbolos que pertenecen a gramática textual concreta se muestran en la columna de la izquierda; los símbolos correspondientes a la gramática gráfica concreta se muestran en la columna derecha.

<package definition>	<package diagram>
<agent definition>	<agent diagram>
<agent type definition>	<agent type diagram>
<procedure definition>	<procedure diagram>
<data type definition>	
<interface definition>	
<operation definition>	<operation diagram>
<composite state>	<composite state area>
<composite state type definition>	<composite state type diagram>
<sort context parameter>	
<signal definition>	
<signal context parameter>	
<compound statement>	<task area>

Una unidad de ámbito tiene asociada una lista de definiciones. Cada una de las definiciones define una entidad que pertenece a cierta clase de entidad y tiene un nombre asociado, incluyendo <textual gate definition>s, <formal context parameter>s, <agent formal parameters>s y <formal variable parameters> contenidos en la unidad de ámbito.

Aunque los <quoted operation name>s y <string name>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 sintácticamente también fuesen <name>s.

Las entidades pueden agruparse en clases de entidades. Existen las siguientes clases de entidades:

a) lotes;

b) agentes (sistemas, bloques, procesos);

c) tipos de agentes (tipos de sistemas, tipos de bloques, tipos de procesos);

d) canales, puertas;

e) señales, temporizadores, interfaces, tipos de datos;

f) procedimientos, procedimientos remotos;

g) variables (incluyendo parámetros formales), sinónimos;

h) literales, operadores, métodos;

i) variables remotas;

j) géneros;

k) tipos de estados;

l) listas de señales;

m) excepciones.

Un parámetro de contexto formal es una entidad de la misma clase que los correspondientes parámetros de contexto.

Una definición de referencia es una entidad después de la etapa de transformación de <referenced definition> (véase el anexo F).

Se dice que cada entidad tiene su contexto definidor en la unidad de ámbito que la define.

Las entidades se referencian mediante <identifier>s. El <qualifier> de un <identifier> especifica solamente el contexto definidor de la entidad.

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

más a la izquierda. El *Name* de una entidad se representa mediante el calificador, el nombre de la entidad y, sólo para entidades del tipo h), la firma (véase 12.1.7.1, 12.1.4). Todas las entidades de la misma clase deben tener *Names* distintos.

NOTA 1 – En consecuencia, 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 las operaciones que se definen en la misma <data type definition>, en tanto que difieren en al menos un <sort> de argumento o en el <sort> de resultado.

Los <state name>s, <connector name>s, y <gate name>s que existen en las definiciones de canal y los <macro formal name>s y <macro name>s, tienen reglas de visibilidad especiales y no pueden ser calificadas. En las cláusulas pertinentes se explican otras reglas de visibilidad especiales.

NOTA 2 – No existe una <scope unit kind> correspondiente a las unidades de ámbito definidas por los esquemas <task>, <task area>, y <compound statement>. Por lo tanto, no es posible hacer referencia mediante calificadores a los identificadores introducidos en una definición anexa a dichas unidades de ámbito.

Se puede hacer referencia a una entidad utilizando un <identifier>, en caso de que la entidad sea visible. Una entidad visible es una unidad de ámbito si:

a) tiene su contexto definidor en dicha unidad de ámbito; o

b) la unidad de ámbito es una especialización y la entidad es visible en el tipo de base; y

   1) no está protegida de la visibilidad mediante una construcción especial definida en 12.1.9.3; y

   2) no se ha aplicado una denominación de especialización de datos (12.1.3); y

   3) no es un parámetro de contexto formal que se haya vinculado a un parámetro de contexto determinado (8.2); o

c) la unidad de ámbito tiene una <package use clause> que menciona a un <package> de tal forma que:

   1) el <package use clause> omite la <definition selection list>, o el <name> de la entidad se menciona en una <definition selection>; y

   2) el <package> que constituye el contexto definidor de la entidad omite la <package interface> o el <name> de la entidad se menciona en la <package interface>; o

d) la unidad de ámbito contiene una <interface definition> que es el contexto definidor de la entidad (véase 12.1.2); o

e) la unidad de ámbito contiene una <data type definition> que es el contexto definidor de la entidad y no está protegido de la visibilidad mediante un constructivo construcción especial definido en 12.1.9.3; o

f) la entidad es visible en la unidad de ámbito que define dicha unidad de ámbito.

Está permitido omitir alguno de los <path item>s situados más a la izquierda, o bien, todo el <qualifier> de un <identifier>, si el <path item>s omitido puede expandirse inequívocamente hasta un <qualifier> completo.

Cuando la parte del <name> de un <identifier> denota una entidad que no es una clase de entidad h), el <name> se vincula a una entidad que tiene su contexto definidor en la unidad de ámbito envolvente más cercana en la que el <qualifier> del <identifier> es el mismo que en la parte situada más a la derecha del <qualifier> completo que denota a dicha unidad de ámbito (resolución mediante contenedor). Si el <identifier> no contiene un <qualifier>, entonces no se aplica el requisito relativo a la correspondencia del <qualifier>s.

La vinculación de un <name> a una definición mediante resolución por contenedor se realiza siguiendo los pasos siguientes, comenzando por la unidad denotada por el <qualifier> parcial:

a) si en una unidad de ámbito existe una única entidad con el mismo <name> y clase de entidad, el <name> se vincula a dicha entidad; en otro caso

b) si la unidad de ámbito es una especialización, se repite recurrentemente el paso a) hasta que el <name> pueda vincularse a una entidad; en otro caso

c) si la unidad de ámbito tiene una <package use clause> y sólo existe una entidad que es visible en el <package>, el <name> se vincula a dicha entidad; en otro caso

d) si la unidad de ámbito tiene una <interface definition> y sólo existe una entidad que es visible en la <interface definition>, el <name> se vincula a dicha entidad; en otro caso

e) se intenta la resolución por contenedor en la unidad de ámbito que define la unidad de ámbito actual.

Con respecto a la visibilidad y la utilización de calificadores, se considera que una <package use clause> asociada a una unidad de ámbito representa una definición de lote que contiene la unidad de ámbito y que se define en la unidad de ámbito en la que dicha unidad de ámbito se ha definido. Si el <identifier> no contiene un <qualifier>, se considera que una <package use clause> es la unidad de ámbito envolvente más cercana a la unidad de ámbito a la cual está asociada y que contiene las entidades visibles desde el lote.

NOTA 3 – En la sintaxis concreta, los paquetes no pueden definirse dentro de otras unidades de ámbito. La regla anterior solo se aplica a la definición de reglas de visibilidad que se aplican a los lotes. Una consecuencia de esta regla es que puede hacerse referencia a los nombres de un lote utilizando distintos calificadores, uno para cada una de las <package use clause> contenidas en el lote.

Cuando la parte del <name> de un <identifier> denota una entidad del tipo h), la vinculación del <name> con un definición debe ser resuelto por el contexto. La resolución mediante contexto se intenta después de la resolución mediante contenedor, es decir, si puede vincularse un <name> a una entidad mediante resolución realizada por el contenedor, dicha vinculación se utiliza incluso si la resolución mediante contexto puede asimismo vincular dicho <name> a una entidad. El contexto para resolver un <name> es una <assignment> (si el <name> ha tenido lugar en una <assignment>), una <decision> (si el <name> ha tenido lugar la <question> o las <answer>s de una <decision>), o en otro caso, una <expression> que no forma parte de ninguna otra <expression>. La resolución mediante contexto se realiza de la forma siguiente:

a) Para cada <name> que aparece en el contexto, debe identificarse el conjunto de <identifier>s, de tal forma que la parte <name> sea visible, teniendo el mismo <name> y teniendo en cuenta la redenominación del <qualifier> parcial.

b) Se construye el producto de los conjuntos de <identifier>s asociados con cada <name>.

c) Sólo se consideran aquellos elementos del producto que no violan ninguna limitación de género estático, teniendo asimismo en cuenta los géneros de los lotes que no son visibles en una <package use clause>. Cada uno de los elementos restantes representa una vinculación posible y estáticamente correcta de los <name>s de la <expression> con entidades.

d) Debido a la posibilidad de polimorfismo en las <assignment>s (véase 12.3.3), el género estático de una <expression> puede no ser el mismo que el género estático de la <variable>, ocurriendo lo mismo para las asignaciones implícitas de los parámetros. El número de discrepancias se contabiliza para cada elemento.

e) Se comparan los elementos por parejas descartando aquéllos que presentan un número mayor de discrepancias.

f) Si queda más de un elemento, todos los <identifier>s no unívocos deben representar la misma *Dynamic-operation-signature*, ya que si no es así, los <name>s del contexto no pueden vincularse a una definición.

Sólo está permitido omitir la <scope unit kind> opcional en un <path item> si el <name> o el <quoted operation name> determina inequívocamente la unidad de ámbito.

No existe sintaxis abstracta que se corresponda con la <scope unit kind> que se denota mediante **operator** o **method**.

En la gramática textual concreta, el nombre o identificador opcional de una definición que es posterior a las palabras clave de finalización (**endsystem**, **endblock**, etc.) de las definiciones, debe ser sintácticamente el mismo que el nombre o el identificador que sigue a las correspondientes palabras claves de inicio (**system**, **block**, etc., respectivamente).

6.4 Texto informal

Gramática abstracta

Informal-text :: ...

Gramática textual concreta

<informal text> ::=
<character string>

Semántica

Si se utiliza texto informal en una especificación, ello significa que este texto no tiene ninguna semántica definida mediante SDL. La semántica del texto informal puede definirse por otros medios.

6.5 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, que pueden cruzarse unos con otros. El cruce de un símbolo con otro no conlleva una asociación lógica. Los siguientes son símbolos de línea:

<association symbol>
<channel symbol>
<create line symbol>
<dashed association symbol>
<dependency symbol>
<flow line symbol>
<solid association symbol>
<solid on exception association symbol>
<specialization relation symbol>

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>, <raise symbol>, <handle 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 entre sí.

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).

6.6 Partición de dibujos

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

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

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

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

<kernel heading> ::=
[<virtuality>] [**exported**]
<drawing kind> <drawing qualifier> | <drawing name>

<drawing kind> ::=
package | **system** [**type**] | **block** [**type**] | **process** [**type**]
state [**type**] | **procedure** | **operator** | **method**

<extra heading> ::=
part of drawing heading not in kernel heading

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

<page number> ::=
<literal name>

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

<symbol> ::=
any of the terminals defined with a rule name ending in "symbol"

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

Un <symbol> es un símbolo no terminal gráfico (véase 5.4.3).

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

El <extra heading> debe figurar en la primera página de un dibujo, pero es optativo en las páginas siguientes. <heading> y <drawing kind> se elaboran para los dibujos específicos en las distintas cláusulas de esta Recomendación. <extra heading> no se define más detalladamente en la presente Recomendación.

<virtuality> denota la virtualidad del tipo definido por el diagrama (véase 8.3.2) y **exported** indica si un procedimiento es exportado como procedimiento remoto (véase 10.5).

Los dibujos de SDL/GR son <specification area>, <package diagram>, <agent diagram>, <agent type diagram>, <procedure diagram>, <operation diagram>, <composite state area> y <composite state type diagram>.

6.7 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>.

La sintaxis concreta de la segunda forma es:

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

<comment> ::=
comment <character string>

<end> en <package text area>, <agent text area>, <procedure text area>, <composite state text area>, <operation text area>, y <statement list> no contendrán <comment>.

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.

NOTA – <text> en *Gramática gráfica concreta* corresponde a <character string> en *gramática textual concreta* sin tener <apostrophe>s. Cualquier conversión entre las notaciones textual y gráfica duplica o repite los <apostrophe>s incluidos.

6.8 Ampliación de texto

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

<text extension 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.

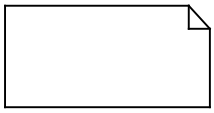
El <text> contenido en el <text extension symbol> es una continuación del texto dentro del símbolo gráfico y se considera contenido en ese símbolo y, por lo tanto, se trata como un número de unidades léxicas.

6.9 Símbolo de texto

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

Gramática gráfica concreta

<text symbol> ::=



7 Organización de las especificaciones SDL

Un sistema SDL no puede describirse normalmente como una porción independiente de texto o mediante un simple diagrama. Por tanto, el lenguaje soporta la partición de la especificación y la utilización del SDL desde cualquier lugar.

7.1 Marco

Una <sdl specification> puede describirse como una <system specification> monolítica (posiblemente aumentada como una colección de <package>s) o como una colección de <package>s y <referenced definition>s. Un <package> permite utilizar definiciones en diferentes contextos "empleando" el lote en esos contextos, es decir, en sistemas o lotes 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. Está "insertada" exactamente en un lugar (el contexto definidor) empleando una referencia. Una <specification area> permite describir gráficamente las relaciones entre <system specification> y <package>s.

Gramática abstracta

SDL-specification :: [*Agent-definition*]
*Package-definition-**set***

Gramática concreta

<sdl specification> ::=
[<specification area>]
{ <package> | <system specification> } <package>* <referenced definition>*

<system specification> ::=
<textual system specification> | <graphical system specification>

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

Gramática textual concreta

<textual system specification> ::=
<agent definition>
| <textual typebased agent definition>

Gramática gráfica concreta

<graphical system specification> ::=
<agent diagram>
| <graphical typebased agent definition>

<specification area> ::=
<frame symbol> ***contains***
{ <agent reference area>
| <graphical typebased agent definition>
[***is connected to*** {<graphical package use area>+ }***set***]
}
{<package reference area>* }***set***}

Semántica

Una *SDL-specification* tiene la semántica combinada del agente de sistema (si se da uno) con la de los lotes. Si no se especifica un agente de sistema, la especificación proporciona un conjunto de definiciones para uso en otras especificaciones.

Modelo

Una <system specification> que constituye un <process definition> o una <textual typebased process definition> es una sintaxis derivada para una < system definition> que tiene el mismo nombre que el proceso, y que contiene canales implícitos y como única definición el <process definition> o <textual typebased process definition>.

Una <system specification> que constituye un <process diagram> o una <graphical typebased process definition> es una sintaxis derivada para un < system diagram> que tiene el mismo nombre que el proceso, y que contiene canales implícitos y como única definición el <process diagram> o <graphical typebased process definition>.

Una <system specification> que constituye una <block definition> o una <textual typebased block definition> es una sintaxis derivada para una <system definition> que tiene el mismo nombre que el bloque, y que contiene canales implícitos y como única definición la <block definition> o <textual typebased block definition>.

Una <system specification> que constituye un <block diagram> o una <graphical typebased block definition> es una sintaxis derivada para un <system diagram> que tiene el mismo nombre que el bloque, y que contiene canales implícitos y como única definición el <block diagram> o <graphical typebased block definition>.

7.2 Lote

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

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

Las definiciones dentro de un lote son visibles para un sistema u otros lotes mediante una cláusula de utilización de lote.

Gramática abstracta

Package-definition	::	*Package-name* *Package-definition-**set*** *Data-type-definition-**set*** *Syntype-definition-**set*** *Signal-definition-**set*** *Exception-definition-**set*** *Agent-type-definition-**set*** *Composite-state-type-definition-**set*** *Procedure-definition-**set***

Gramática textual concreta

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

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

<entity in package> ::=
<agent type definition>
| <agent type reference>
| <package definition>
| <package reference>
| <signal definition>
| <signal reference>
| <signal list definition>
| <remote variable definition>
| <data definition>
| <data type reference>
| <procedure definition>
| <procedure reference>
| <remote procedure definition>
| <composite state type definition>
| <composite state type reference>
| <exception definition>
| <select definition>
| <macro definition>
| <interface reference>
| <association>

<package reference> ::=
package [<qualifier>] <package name> **referenced** <end>

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

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

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

<selected entity kind> ::=
system type
| **block type**
| **process type**
| **package**
| **signal**
| **procedure**
| **remote procedure**
| **type**
| **signallist**
| **state type**
| **synonym**
| **remote**
| **exception**
| **interface**

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

Para cada <package identifier> mencionado en una <package use clause> tiene que existir un <package> correspondiente. Este lote puede ser parte de la <sdl specification> o puede ser un lote contenido en otro lote o, si no, tiene que existir un mecanismo para ganar acceso al <package> referenciado, justamente como si formara parte de la <sdl specification>. Este mecanismo no se define en esta Recomendación.

Si el lote forma parte de la <sdl specification> o si existe un mecanismo para ganar acceso al <package> de referencia, no podrá haber un <qualifier> en un <package identifier>.

Si el <package> correspondiente está contenido en otro lote, el <package identifier> refleja la estructura jerárquica desde el <package> más exterior hasta el <package> definido. Los <path item>s más a la izquierda pueden omitirse.

El <package identifier> tiene que designar un lote visible. Todos los <package>s en el <qualifier> del <package identifier> plenamente calificado tienen que ser visibles. Un lote es visible si forma parte de la <sdl specification>, o si su <identifier> es visible de acuerdo con las reglas de visibilidad de SDL para <identifier>. Las reglas de visibilidad de SDL implican que se puede hacer visible un <package identifier> mediante una <package use clause> y que un lote es visible en el ámbito en que está contenido. Este ámbito abarca también la <package use clause> del lote contenedor.

Asimismo, si en una <sdl specification> se omite la <system specification>, tiene que haber un mecanismo para usar los <package>s en otras <sdl specification>. Antes de que los <package>s sean utilizados en otras <sdl specification>, se aplica el modelo para macros y definiciones referenciadas. Este mecanismo no se define de otra forma en esta Recomendación.

El **procedure** <selected entity kind> se utiliza para seleccionar los procedimientos (normales) y los procedimientos remotos. Si ambos procedimientos, normal y remoto, tienen en <name> dado, **procedure** denota el procedimiento normal. Para hacer que la <definition selection> denote al procedimiento remoto, la palabra clave de **procedure** puede ser precedida por **remote**.

La palabra clave **type** se utiliza para la selección de un nombre de género y de un nombre de sintipo en un lote. **remote** se utiliza para la selección de una definición de variable remota.

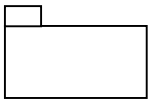
Gramática gráfica concreta

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

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

<package text area> ::=
                        <text symbol> contains
                              {       <agent type reference>
                              |       <package reference>
                              |       <signal definition>
                              |       <signal reference>
                              |       <signal list definition>
                              |       <remote variable definition>
                              |       <data definition>
                              |       <data type reference>
                              |       <procedure definition>
                              |       <procedure reference>
                              |       <remote procedure definition>
                              |       <exception definition>
                              |       <select definition>
                              |       <macro definition>
                              |       <interface reference> }*

<diagram in package> ::=
                        <package diagram>
                    |   <package reference area>
                    |   <type in agent area>
                    |   <data type reference area>
                    |   <signal reference area>
                    |   <procedure reference area>
                    |   <interface reference area>
                    |   <create line area>
                    |   <option area>

<package reference area> ::=
                        <package symbol> contains <package identifier>
                              [       is connected to {<graphical package use area>+ }set]
```

<graphical package use area> ::=

<dependency symbol> ***is connected to*** { <package diagram> | <package reference area> }

<package symbol> ::=



<dependency symbol> ::=

El <package use area> debe situarse encima del <frame symbol>. El <qualifier> y <package name> facultativos de un <package reference area> deben estar contenidos en el rectángulo inferior del <package symbol>.

Las <graphical package use area>s para una <package reference area> son especificaciones parciales de la correspondiente <package use clause> del <package diagram>, (o <package> o <system specification> para una <package reference area> en una <specification area>), y deben ser consistentes con esta <package use clause>.

Semántica

En la cláusula 6.3 se explica cual es la visibilidad del nombre de una entidad definida en un <package>.

Las señales que no son visibles en una cláusula **use**, pueden formar parte de una lista de señales mediante un <signal list identifier> que sea visible en una cláusula **use**, pudiendo dichas señales afectar al conjunto de señales de entrada válidas de un agente.

Si el nombre en una <definition selection> denota un <sort>, la <definition selection> también denota implícitamente el tipo de datos que define el <sort> y todas las descripciones y operaciones que definen los tipos de datos. Si el nombre en una <definition selection> denota un sintipo, la <definition selection> también denota implícitamente el tipo de datos definido en el <parent sort identifier> y todas las descripciones y operaciones que definen los tipos de datos.

La <selected entity kind> de la <definition selection> denota la clase de entidad de <name>. En una <definition selection list> deben distinguirse todas las parejas (<selected entity kind>, <name>). En una <definition selection> de una <package interface>, la <selected entity kind> únicamente puede omitirse si no hay otro nombre que tenga su ocurrencia de definición directamente en el <package>. En una <definition selection> de una <package use clause>, la <selected entity kind> puede omitirse únicamente si se menciona otra entidad con exactamente el mismo nombre en cualquier <definition selection list> para el lote, o si el lote no tiene <definition selection list> y contiene directamente una definición única de ese nombre.

Modelo

Una <system definition> y cada <package definition> tiene una <package use clause> implícita:

use Predefined;

donde Predefined denota un lote que contiene los datos predefinidos tal como se define en el anexo D.

Si un lote se menciona en varias <package use clause>s de una <package definition>, ello corresponde a una <package use clause> que selecciona la unión de las definiciones seleccionadas en las <package use clause>.

7.3 Definición referenciada

Gramática concreta

<referenced definition> ::=

<definition> | <diagram>

Gramática textual concreta

<definition> ::=
<package definition>
| <agent definition>
| <agent type definition>
| <composite state>
| <composite state type definition>
| <procedure definition>
| <operation definition>
| <macro definition>

Gramática gráfica concreta

<diagram> ::=
<package diagram>
| <agent diagram>
| <agent type diagram>
| <composite state area>
| <composite state type diagram>
| <procedure diagram>
| <operation diagram>

Para cada <referenced definition >, excepto <macro definition>, tiene que haber una referencia en el <package> o <system specification> asociados. Las referencias textuales y gráficas se definen como <... reference> y <... reference area> respectivamente (por ejemplo <block reference> y <block reference area>).

Un <qualifier> y <name> facultativos están presentes 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 <name>, el <qualifier> de uno de los <identifier>s no debe constituir la parte izquierda del otro <qualifier> y no se podrá omitir ninguno de los dos <qualifier>s. El <qualifier> tiene que estar presente si la <referenced definition> es una <package definition>.

No se permite especificar un <qualifier> después de la(s) palabra(s) clave inicial(es) para las definiciones que no son <referenced definition>.

Modelo

Antes de derivar las propiedades de una <system specification>, cada referencia es sustituida por la <referenced definition> correspondiente. En esta sustitución, el <qualifier> de la <referenced definition> se suprime. Si la <referenced definition> es un <diagram> referenciado a partir de una <definition>, o es una <definition> referenciada a partir de un <diagram>, se considera que la <referenced definition> ha sido traducida a la gramática adecuada durante la sustitución.

8 Conceptos estructurales

En esta cláusula se presentan una serie de mecanismos del lenguaje para permitir el modelado de fenómenos específicos de una aplicación mediante ejemplares y conceptos específicos de aplicación por tipos. La herencia pretende representar la generalización y especialización de los conceptos.

Los mecanismos del lenguaje presentados proporcionan lo siguiente:

a) definiciones de tipo (puro) que pueden definirse en un sistema o en un lote;

b) definiciones de ejemplares basados en tipos que definen los ejemplares o conjunto de ejemplares de acuerdo a los tipos;

c) las definiciones de tipos parametrizados independientes del ámbito circundante mediante parámetros de contexto y que pueden estar vinculados a ámbitos específicos;

d) la especialización de definiciones de supertipos en definiciones de subtipos añadiendo propiedades y redefiniendo tipos virtuales y transiciones.

8.1 Tipos, ejemplares y puertas

Es necesario distinguir entre la definición de ejemplares (o de conjunto de ejemplares) y la definición de tipos en las descripciones SDL. En esta cláusula se presenta (en 8.1.1) la definición de tipo para agentes y (en 8.1.3) las correspondientes especificaciones de ejemplares, mientras que la presentación de otros tipos se realiza en las subcláusulas sobre procedimientos (9.4), señales (10.3), temporizadores (11.15), géneros (12.1.1) e interfaces (12.1.2). La definición de un tipo de agente no está conectada (mediante canales) a ningun ejemplar; en su lugar, las definiciones de tipo de agente presentan puertas (8.1.6). Éstas son puntos de conexión de los ejemplares basados en tipos para canales.

Un tipo define un conjunto de propiedades. Todos los ejemplares del tipo (5.2.1) tienen este conjunto de propiedades.

Un ejemplar (o un conjunto de ejemplares) siempre tiene un tipo, que está implícito si el ejemplar no está basado explícitamente en un tipo. Por ejemplo, un diagrama de proceso tiene un tipo de proceso anónimo equivalente implicado.

8.1.1 Definiciones de tipo estructural

Son definiciones de tipo para entidades que se utilizan en la estructura de una especificación. En contraste, las definiciones de procedimientos son también definiciones de tipos, pero organizan el comportamiento más bien que la estructura.

8.1.1.1 Tipo de agente

Un tipo de agente es un tipo de sistema, bloque o proceso. Cuando el tipo se utiliza para definir un agente, el agente es de la clase correspondiente (sistema, bloque o proceso).

Gramática abstracta

Agent-type-definition :: *Agent-type-name*
Agent-kind
[*Agent-type-identifier*]
*Agent-formal-parameter**
*Data-type-definition-**set***
*Syntype-definition-**set***
*Signal-definition-**set***
*Timer-definition-**set***
*Exception-definition-**set***
*Variable-definition-**set***
*Agent-type-definition-**set***
*Composite-state-type-definition-**set***
*Procedure-definition-**set***
*Agent-definition-**set***
*Gate-definition-**set***
*Channel-definition-**set***
[*State-machine-definition*]

Agent-type-identifier = *Identifier*

Gramática textual concreta

<agent type definition> ::=
<system type definition> | <block type definition> | <process type definition>

```
<agent type structure> ::=
                        [<valid input signal set>]
                        {       {       <entity in agent>
                                |       <channel definition>
                                |       <gate in definition>
                                |       <agent definition>
                                |       <agent reference>
                                |       <textual typebased agent definition> }*
                                [ <state partitioning>]
                        |       {       <entity in agent>
                                |       <gate in definition> }*
                                 <agent type body> }

<type preamble> ::=
                        [ <virtuality> | <abstract> ]

<agent type additional heading> ::=
                        [<formal context parameters>] [<virtuality constraint>]
                        <agent additional heading>

<agent type reference> ::=
                        <system type reference>
                    |   <block type reference>
                    |   <process type reference>

<agent type body> ::=
                        [ [<on exception>] <start> ]
                        { <state> | <exception handler> | <free action> } *
```

Gramática gráfica concreta

```
<agent type diagram> ::=
                        <system type diagram> | <block type diagram> | <process type diagram>

<type in agent area> ::=
                        <agent type diagram>
                    |   <agent type reference area>
                    |   <composite state type diagram>
                    |   <composite state type reference area>
                    |   <procedure diagram>
                    |   <procedure reference area>
                    |   <data type reference area>
                    |   <signal reference area>
                    |   <association area>

<agent type diagram content> ::=
                        {   <agent text area>*
                            <type in agent area>*
                            { <interaction area> | <agent type body area> } }set

<agent type body area> ::=
                            <type state machine graph area>

<type state machine graph area> ::=
                        {   [<on exception association area>] [<start area>]
                            { <state area> | <exception handler area> | <in connector area> }* }set

<agent type reference area> ::=
                            {       <system type reference area>
                                  | <block type reference area>
                                  | <process type reference area> }
                            is connected to { <gate property area>* }set
```

La <package use area> debe estar situada por encima del <frame symbol>.

Si existe una <agent type reference area> para el agente que se ha definido mediante un <agent type diagram>, las <gate property area>s asociadas con la <agent type reference area>

corresponden a los <gate on diagram>s asociados con el <agent type diagram>. Ninguna <gate property area> asociada con el <agent type reference area> podrá contener <signal list item>s no contenidos en los correspondientes <gate on diagram>s asociados con el <agent type diagram>.

Semántica

Una *Agent-type-definition* define un tipo de agente. Todos los agentes de un tipo de agente tienen las mismas propiedades, que se definen para dicho tipo de agente.

Las señales que se mencionan en las <output>s de la máquina de estados de un tipo de agente deben estar en el conjunto completo de señales de entrada válidas del tipo de agente o en la <signal list> de una puerta en el sentido procedente del tipo de agente.

La definición de un tipo de agente implica la definición de una interfaz en el mismo ámbito del tipo de agente (véase 12.1.2). El género pid implícitamente definido por esta interfaz se identifica mediante *Agent-type-name* y es visible en la misma unidad de ámbito que aquélla en la que está definido el tipo de agente.

NOTA – Como cada tipo de agente tiene una interfaz implícitamente definida con el mismo nombre, el tipo de agente debe tener un nombre distinto de cada interfaz explícitamente definida, y cada agente (éstos también tienen interfaces implícitas) definido en el mismo ámbito; de lo contrario, hay coincidencia de nombres.

Las propiedades definidas en una *Agent-type-definition* tales como el *Procedure-definition-**set**,* *Agent-definition-**set**,* y *Gate-definition-**set*** determinan las propiedades de cualquier *Agent-definition* basada en el tipo, y están por tanto descritas en la cláusula 9.

Modelo

Un tipo de agente con un <agent type body> o una <agent type body area> es una notación taquigráfica para un tipo de agente que sólo tiene una máquina de estados, pero no agentes contenidos. Esta máquina de estados se obtiene sustituyendo el <agent type body> o <agent type body area> por una definición de estado compuesto. Esta definición de estado compuesto tiene el mismo nombre que el tipo de agente y su *State-transition-graph* se representa por el <agent type body> o la <agent type body area>.

8.1.1.2 Tipo de sistema

Una definición de tipo de sistema es una definición de tipo de agente de nivel superior. Se denota mediante la palabra clave **system type**. Un tipo de sistema no debe estar contenido en ninguna otra definición de agente o de tipo de agente. Un tipo de sistema tampoco debe ser abstracto ni virtual.

Gramática textual concreta

<system type definition> ::=
<package use clause>**
<system type heading> <end> <agent type structure>
endsystem type [[<qualifier>] <system type name>] <end>

<system type heading> ::=
system type [<qualifier>] <system type name>
<agent type additional heading>

<system type reference> ::=
system type <system type identifier> <type reference properties>

Un <formal context parameter> de <formal context parameters> no debe ser un <agent context parameter>, <variable context parameter> ni <timer context parameter>.

El <agent type additional heading> en una <system type definition> o un <system type diagram> no puede incluir <agent formal parameters>.

Gramática gráfica concreta

<system type diagram> ::=
<frame symbol> ***contains*** {<system type heading> <agent type diagram content> }
[***is associated with*** <package use area>]

<system type reference area> ::=
<type reference area>

<system type symbol> ::=
<block type symbol>

La <type reference area> que forma parte de una <system type reference area> debe tener un <graphical type reference heading> que contenga un <system type name>.

Semántica

Una <system type definition> define un tipo de sistema.

8.1.1.3 Tipo de bloque

Gramática textual concreta

<block type definition> ::=
<package use clause>*****
<block type heading> <end> <agent type structure>
endblock type [[<qualifier>] <block type name>] <end>

<block type heading> ::=
<type preamble>
block type [<qualifier>] <block type name>
<agent type additional heading>

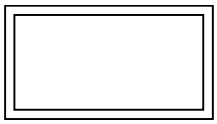
<block type reference> ::=
<type preamble>
block type <block type identifier> <type reference properties>

Gramática gráfica concreta

<block type diagram> ::=
<frame symbol> ***contains*** {<block type heading> <agent type diagram content> }
is connected to {{ <gate on diagram>***** }***set*** }
[***is associated with*** <package use area>]

<block type reference area> ::=
<type reference area>

<block type symbol> ::=



La <type reference area> que forma parte de una <block type reference area> debe tener un <graphical type reference heading> que contenga un <block type name>.

El <gate on diagram> en un <block type diagram> no puede incluir <external channel identifiers>.

El <gate on diagram> en un <block type diagram> tiene que estar fuera del marco del diagrama.

Semántica

Un <block type definition> define un tipo de bloque.

8.1.1.4 Tipo de proceso

Gramática textual concreta

<process type definition> ::=
<package use clause>*****
<process type heading> <end> <agent type structure>
endprocess type [[<qualifier>] <process type name>] <end>

<process type heading> ::=
<type preamble>
process type [<qualifier>] <process type name>
<agent type additional heading>

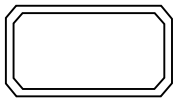
<process type reference> ::=
<type preamble>
process type <process type identifier> <type reference properties>

Gramática gráfica concreta

<process type diagram> ::=
<frame symbol> ***contains*** {<process type heading> <agent type diagram content> }
is connected to {{ <gate on diagram>***** }***set*** }
[***is associated with*** <package use area>]

<process type reference area> ::=
<type reference area>

<process type symbol> ::=



La <type reference area> que forme parte de una <process type reference area> debe tener un <graphical type reference heading> que contenga un <process type name>.

Semántica

Una <process type definition> define un tipo de proceso.

El <gate on diagram> en un <process type diagram> no puede incluir <external channel identifiers>.

El <gate on diagram> en un <process type diagram> tiene que estar fuera del marco del diagrama.

8.1.1.5 Tipo de estado compuesto

Gramática abstracta

Composite-state-type-definition	::	*State-type-name*
		[*Composite-state-type-identifier*]
		*Composite-state-formal-parameter******
		*State-entry-point-definition**-set***
		*State-exit-point-definition**-set***
		*Gate-definition**-set***
		*Connect-node**-set***
		*Data-type-definition**-set***
		*Syntype-definition**-set***
		*Exception-definition**-set***
		*Composite-state-type-definition**-set***
		*Variable-definition**-set***
		*Procedure-definition**-set***
		[*Composite-state-graph* | *State-aggregation-node*]
Composite-state-type-identifier	::	*Identifier*

El *Gate-definition-set* debe estar vacío, a menos que el estado compuesto se utilice como una *State-machine-definition*.

Gramática textual concreta

<composite state type definition> ::=
{<package use clause>}*
<composite state type heading> <end> **substructure**
[<valid input signal set>]
{<gate in definition>}*
{<state connection points>}*
{<entity in composite state>}*
<composite state body>
endsubstructure state type [[<qualifier>] <composite state type name>] <end>
| {<package use clause>}*
<state aggregation type heading> <end> **substructure**
[<valid input signal set>]
{<gate in definition>}*
{<state connection points>}*
{<entity in composite state>}*
<state aggregation body>
endsubstructure state type [[<qualifier>] <composite state type name>] <end>

<composite state type heading> ::=
[<virtuality>]
state type [<qualifier>] <composite state type name>
[<formal context parameters>] [<virtuality constraint>]
[<specialization>]
[<agent formal parameters>]

<state aggregation type heading> ::=
[<virtuality>]
state aggregation type [<qualifier>] <composite state type name>
[<formal context parameters>] [<virtuality constraint>]
[<specialization>]
[<agent formal parameters>]

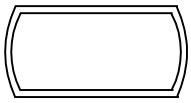
<composite state type reference> ::=
<type preamble>
state type <composite state type identifier> <type reference properties>

Gramática gráfica concreta

<composite state type diagram> ::=
<frame symbol>
contains {
<composite state type heading>
{ {<composite state text area>}*
{<type in composite state area>}*
<composite state body area> }***set*** }
is associated with { <graphical state connection point>* }***set***
is connected to {{ <gate on diagram>* }***set*** }
[***is associated with*** <package use area>]
| <frame symbol>
contains {
<state aggregation type heading>
{ {<composite state text area>}*
{<type in composite state area>}*
<state aggregation body area> }***set*** }
is associated with { <graphical state connection point>* }***set***
is connected to {{ <gate on diagram>* }***set*** }
[***is associated with*** <package use area>]

<composite state type reference area> ::=
<type reference area>
is connected to {<gate property area>*}***set***

<composite state type symbol> ::=



La <package use area> debe estar situada encima del <frame symbol>.

La <type reference area> que forma parte de una <composite state type reference area> debe tener un <graphical type reference heading> que contenga un <composite state type name>.

El <gate on diagram> en un <composite state type diagram> no puede incluir <external channel identifiers>.

Los <gate on diagram>s en un <composite state type diagram> tienen que estar fuera del marco del diagrama.

Si hay una <composite state type reference area> para un estado compuesto definido por un <composite state type diagram>, las <gate property area>s asociadas con la <composite state type area> corresponden a los <gate on diagram>s asociados con el <composite state type diagram>. Ninguna <gate property area> asociada con la <composite state type reference area> podrá contener <signal list item>s que no estén contenidos en el correspondiente <gate on diagram> asociado con el <composite state type diagram>.

Semántica

Una *Composite-state-type-definition* define un tipo de estado compuesto. Todos los estados compuestos de un tipo de estado compuesto tienen las mismas propiedades que se definen para dicho estado compuesto. Las semánticas se definen con más detalle en 11.11.

8.1.2 Expresión de tipo

Una expresión de tipo se utiliza para definir un tipo en términos de otro, como se define por especialización en 8.3.

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 8.2.

Al margen 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 actuales a los parámetros de contexto formales de tipo parametrizado denotados por el identificador de <base type>.

Si se omiten algunos de los parámetros de contexto, el tipo sigue siendo parametrizado.

Además de cumplir las condiciones estáticas de la definición denotada por el <base type>, la utilización de la <type expression> debe igualmente cumplir cualquier condición estática relativa al tipo resultante.

NOTA – Las propiedades estáticas de la utilización de <type expression> pueden ser violadas en los casos siguientes:

– 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.

– Cuando una salida en una unidad de ámbito refiere a una puerta o un canal que no están definidos en el tipo circundante más próximo que tenga puertas, la ejemplificación de ese tipo resulta en una especificación errónea si no hay trayecto de comunicación hacia la puerta.

– Cuando un procedimiento contiene referencias a identificadores de señal, variables remotas y procedimientos remotos, la especialización de ese procedimiento dentro de un agente resulta en una especificación errónea si la utilización de esos identificadores en el procedimiento viola la utilización válida del proceso.

– Cuando tipos de estado son ejemplificados como partes de la misma agregación de estado, el estado compuesto resultante es erróneo si dos o más partes tienen la misma señal en el conjunto de señales de entrada.

– Cuando una unidad de ámbito tiene un parámetro de contexto de agente que se utiliza en una acción de salida, la existencia de un trayecto de comunicación depende de cuales son los parámetros de contexto real que se utilizará.

– Cuando una unidad de ámbito tiene una parámetro de contexto de género, la aplicación de una parámetro de contexto de género real resulta en una especificación errónea si en el tipo especializado se intenta una asignación polimórfica a un valor.

– Si un parámetro formal de un procedimiento añadido en una especialización tiene el <parameter kind> in/out o out una llamada en el supertipo a un subtipo (utilizando this) resultará en la omisión de un parámetro real in/out o out , es decir, se produce una especificación errónea.

– 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, los nombres de los <formal context parameter>s correspondientes se insertan como <actual context parameter>s. Estos <actual context parameter>s tienen ahora el contexto definidor en la unidad de ámbito actual.

8.1.3 Definiciones basadas en tipos

Una definición de agente basada en tipo define un conjunto de ejemplares de agente, de acuerdo con un tipo denotado por <type expression>. Las entidades definidas obtienen sus propiedades de los tipos en los cuales se basan.

Gramática textual concreta

```
<textual typebased agent definition> ::=
                    <textual typebased system definition>
                |   <textual typebased block definition>
                |   <textual typebased process definition>
```

El tipo de agente denotado por <base type> en la expresión de tipo de una <textual typebased agent definition> tiene que contener una transición de arranque no etiquetada en su máquina de estados.

Gramática gráfica concreta

<graphical typebased agent definition> ::=
<graphical typebased system definition>
| <graphical typebased block definition>
| <graphical typebased process definition>

<inherited agent definition> ::=
<inherited block definition>
| <inherited process definition>

El tipo de agente denotado por <base type> en la expresión de tipo de una <graphical typebased agent definition> o <inherited agent definition> debe contener una transición de arranque no etiquetada en su máquina de estados.

En una <graphical typebased agent definition>, la <graphical gate definition> y la <graphical interface gate definition> deben colocarse fuera del <block symbol> o <process symbol>.

8.1.3.1 Definición de sistema basada en el tipo de sistema

Gramática textual concreta

<textual typebased system definition> ::=
<typebased system heading> <end>

<typebased system heading> ::=
system <system name> <colon> <system type expression>

Gramática gráfica concreta

<graphical typebased system definition> ::=
<block symbol> ***contains*** <typebased system heading>

Semántica

Una definición de sistema basada en tipo realiza una *Agent-definition* con un *Agent-kind* **SYSTEM** derivado por transformación a partir de un tipo de sistema.

Modelo

Una <textual typebased system definition> o una <graphical typebased system definition> se transforman en una <system definition> que tiene las definiciones del tipo de sistema definidas por <system type expression>.

8.1.3.2 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 instances>] <colon> <block type expression>

Gramática gráfica concreta

<graphical typebased block definition> ::=
<block symbol> ***contains*** { <typebased block heading> { <gate>* }***set*** }
is connected to { {<gate property area>*}***set*** }

<inherited block definition> ::=
<dashed block symbol> ***contains*** { <block identifier> { <gate>* }***set*** }
is connected to { {<gate property area>*}***set*** }

<dashed block symbol> ::=

Las <gate>s se sitúan cerca del límite de los símbolos y asociadas al punto de conexión con canales.

Semántica

Una definición de bloque basada en tipo define *Agent-definitions* de *Agent-kind* **BLOCK** derivada 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 de tipo de bloque definidas por <block type expression>.

Una <inherited block definition> únicamente 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 las puertas del bloque heredado.

8.1.3.3 Definición de proceso basado en tipo de proceso

Gramática textual concreta

<textual typebased process definition> ::=
process <typebased process heading> <end>

<typebased process heading> ::=
<process name> [<number of instances>] <colon> <process type expression>

Gramática gráfica concreta

<graphical typebased process definition> ::=
<process symbol> ***contains*** { <typebased process heading> { <gate>* }***set*** }
is connected to { {<gate property area>*}***set*** }

<inherited process definition> ::=
<dashed process symbol> ***contains*** { <process identifier> { <gate>* }***set*** }
is connected to { {<gate property area>*}***set*** }

<dashed process symbol> ::=

Las <gate>s se sitúan cerca del límite de los símbolos y asociadas al punto de conexión con canales.

Semántica

Una definición de proceso basada en tipo define una *Agent-definition* con *Agent-kind* **PROCESS** derivada por transformación a partir de un tipo de proceso.

Modelo

Una <textual typebased process definition> o una <graphical typebased process definition> se transforma en una <process definition> que tiene las definiciones de tipo de proceso definidas por <process type expression>.

Una <inherited process definition> únicamente puede aparecer en una definición de subtipo. Representa el proceso definido en el supertipo de la definición de subtipo. Pueden especificarse canales adicionales conectados a las puertas del proceso heredado.

8.1.3.4 Definición de estado compuesto basado en tipo de estado compuesto

Gramática textual concreta

<textual typebased state partition definition> ::=
state aggregation <typebased state partition heading> <end>

<typebased composite state> ::=
<state name> [<actual parameters>] <colon> <composite state type expression>

Semántica

Una definición de estado compuesto basada en tipo y una definición de partición de estado basada en tipo definen un estado compuesto obtenido por transformación a partir del tipo de estado compuesto.

Modelo

Un <typebased composite state> se transforma en un <composite state> que tiene las definiciones del tipo de servicio definidas por <composite state type expression>.

8.1.4 Tipo abstracto

Gramática concreta

<abstract> ::=
abstract

<abstract> es parte de la definición de tipo. Véanse 8.1.1.1, 8.1.5, 9.4, 10.3 y 12.1.1.

Un tipo es un tipo abstracto si su definición contiene <abstract>.

Un procedimiento abstracto no puede ser llamado.

Un agente basado en tipo (véase 8.1.3) no puede especificarse con un tipo de agente abstracto como tipo.

Un <operation identifier> no debe representar el *Operation-identifier* para el operador Make definido para un tipo de datos abstracto.

Un <literal identifier> no debe representar un *Literal-identifier* que denota una *Literal-signature* de un tipo de datos abstracto.

Un tipo abstracto no puede ser ejemplificado. Sin embargo, un subtipo de un tipo abstracto puede ser ejemplificado si dicho subtipo no es abstracto.

8.1.5 Referencias de tipo

Los diagramas de tipo y las definiciones de tipo de entidad pueden tener referencias de tipo. Una referencia de tipo especifica que un tipo está definido en la unidad de ámbito de la definición o diagrama contenedor (pero completamente descrito en la definición o diagrama de referencia), y que este tipo tiene propiedades que están parcialmente definidas como parte de la referencia de tipo. La definición o diagrama de referencia define las propiedades del tipo, mientras que las referencias de tipo sólo son definiciones parciales. Es necesario que la especificación parcial que forma parte de una referencia de tipo sea consistente con la especificación de la definición o diagrama del tipo. Una especificación parcial de una variable, por ejemplo, puede dar el nombre de la variable pero no el género de la variable. Tiene que haber una variable de ese nombre en la definición referenciada, y en esta definición hay que especificar un género.

La misma definición de tipo puede tener varias referencias de tipo textuales. Las referencias sin calificador deben estar todas en la misma unidad de ámbito y la definición de tipo se inserta en dicha unidad de ámbito.

Gramática textual concreta

<type reference properties> ::=
[**with** { <attribute property> <end> }+]
[**with** { <behaviour property> <end> }+]
referenced <end>

Si <type reference properties> aparece en <package text area>, <agent text area>, <procedure text area>, <composite state text area>, y <operation text area>, no pueden estar presentes ni <attribute property>s, ni <behaviour property>s.

<attribute property> ::=
<variable property>
| <field property>
| <signal parameter property>
| <interface variable property>

Una <attribute property> proporciona una especificación parcial de las propiedades de variables o campos definidos en la definición de tipo a la que se hace referencia en la referencia de tipo. Los elementos de la <attribute property> deben ser consistentes con las correspondientes propiedades de la definición de tipo referenciada.

<variable property> ::=
[<local> | <exported>] <variable name> [<sort>]

<local> ::=
<hyphen>

<exported> ::=
<plus sign>

Una <variable property> se corresponde con una <variable definition> en un tipo de agente, tipo de servicio, procedimiento o tipo de estado compuesto. <local> indica una variable local; <exported> indica una variable exportada. <variable name> y <sort>, si existen, deben ser los mismos que en la correspondiente definición de variable.

<field property> ::=
[<symbolic visibility>] <field name> [<sort>]

<symbolic visibility> ::=
<private>
| <public>
| <protected>

<private> ::=
<hyphen> | **private**

<public> ::=
<plus sign> | **public**

<protected> ::=
<number sign> | **protected**

Una <field property> se corresponde con un <field> en un tipo de datos. <private> (<public>, <protected>)>se corresponde con <visibility> privada (pública, protegida) en el correspondiente campo. <field name> y <sort>, si existen, deben ser los mismos que en la correspondiente definición de campo.

<signal parameter property> ::=
<sort list>

Una <signal parameter property> se corresponde con una lista de parámetros de señal en una definición de señal. La <sort list> debe corresponder con la <sort list> del <signal definition item> de la correspondiente definición de señal. Tiene que haber una sola <signal parameter property> en las <type reference properties> o <type reference area>.

<interface variable property> ::=
<remote variable name> [<sort>]

Una <interface variable property> se corresponde con una <interface variable definition> en una interfaz. El <sort> debe ser el mismo que el <sort> en la definición de variable de la interfaz.

<behaviour property> ::=
{ [**operator**] <operation property> }
| { [**method**] <operation property> }
| { [**procedure**] <procedure property> }
| { [**signal**] <signal property> }
| { [**exception**] <exception property> }
| { [**timer**] <timer property> }
| { <interface use list> }

Una <behaviour property> proporciona una especificación parcial de las propiedades de procedimientos y operaciones que se definen en la definición de tipo a la que hace referencia la referencia de tipo, y esta especificación debe ser consistente con las correspondientes definiciones de la correspondiente definición de tipo.

<operation property> ::=
[<symbolic visibility>] <operation name>
<procedure signature>

Una <operation property> se corresponde a una <operation definition> en una referencia de tipo objeto o valor. <private> (<public>, <protected>) se corresponde a <visibility> (pública, protegida) en la correspondiente definición de operación. La lista de <formal parameter>s, <result>, y <raises> en <procedure signature>, si está presente, tienen que ser los mismos que los <formal parameter>s, <result>, y <raises>, respectivamente, en la correspondiente definición de operación.

<procedure property> ::=
[<local> | <exported>] <procedure name>
<procedure signature>

Una <procedure property> en una referencia de tipo de agente se corresponde con una <procedure definition> en el tipo de agente. <local> indica un procedimiento local; <exported> indica un procedimiento exportado. La lista de <formal parameter>s, <result>, y <raises> en <procedure signature>, si está presente, tienen que ser los mismos que los <procedure formal parameter>s, <procedure result>, y <raises>, respectivamente, de la correspondiente definición de procedimiento.

Una <procedure property> en una referencia de interfaz se corresponde con una <interface procedure definition> en una interfaz. <local> no puede estar presente en una referencia de interfaz. <procedure signature> debe ser la misma, si existe, que la de la correspondiente definición de procedimiento de interfaz.

<signal property> ::=
<signal name> [<sort list>]

Una <signal property> en una referencia de tipo de agente se corresponde con una señal manejada en una entrada del tipo de agente.

Una propiedad de señal en una referencia de interfaz se corresponde con un <signal definition item> en una <signal definition> de la <interface definition>. La <sort list>, si está presente, debe ser la misma del correspondiente <signal definition item>.

<exception property> ::=
<exception name> [<sort list>]

Una <exception property> en una referencia de tipo corresponde a un <exception definition item> de la definición de tipo referenciada por la referencia de tipo. La <sort list>, si está presente, tiene que ser la misma del correspondiente <exception definition item>.

<timer property> ::=
<timer name> [<sort list>]

Una <timer property> en una referencia de tipo corresponde a un <timer definition item> en la definición de tipo referenciada por la referencia de tipo. La <sort list>, si está presente, tiene que ser la misma del correspondiente <timer definition item>.

Una <interface use list> corresponde a una <interface use list> de la definición de interfaz referenciada por la referencia de tipo. Cada <signal list item> tiene que corresponder a un <signal list item> en la <interface use list> de la definición de interfaz referenciada.

Gramática gráfica concreta

<type reference area> ::=
{ <basic type reference area> | <iconized type reference area> }
is connected to { <graphical package use area>* <specialization area>* } ***set***

La <graphical package use area> para una <type reference area> es una especificación parcial de la correspondiente <package use clase> para el diagrama de tipo, y debe ser consistente con éste.

La <specialization area> se conectará a la parte superior de la <basic type reference area>, o <iconized type reference area>, utilizando el extremo del <specialization relation symbol> que no tiene flecha. Tiene que haber una sola <specialization area> para todas las <type reference area>s excepto la de referencia de interfaz.

La <specialization area> corresponde a la <specialization> del tipo referenciado. La <type reference area> conectada tiene que corresponder al <base type> en la <type expression> de la <specialization>. Los <actual context parameters> en la <specialization area> tienen que corresponder a los <actual context parameters> en la <type expression>.

<basic type reference area> ::=
<class symbol>
contains {
<graphical type reference heading>
<attribute properties area>
<behaviour properties area> }

<class symbol> ::=

La posición relativa de las dos líneas que dividen el <class symbol> puede ser diferente de la mostrada.

<graphical type reference heading> ::=
{ <type reference kind symbol> ***contains*** **system** | <system type symbol> }
<system type type reference heading>
| { <type reference kind symbol> ***contains*** **block** | <block type symbol> }
<block type type reference heading>
| { <type reference kind symbol> ***contains*** **process** | <process type symbol> }
<process type type reference heading>
| { <type reference kind symbol> ***contains*** **state** | <composite state type symbol> }
<composite state type type reference heading>
| { <type reference kind symbol> ***contains*** **procedure** | <procedure symbol> }
<procedure type reference heading>
| <type reference kind symbol> ***contains*** **signal**
<signal type reference heading>
| { <type reference kind symbol> ***contains*** { **value** | **object** } | <data symbol> }
<data type type reference heading>
| { <type reference kind symbol> ***contains*** **interface** | <data symbol> }
<interface type reference heading>

El <graphical type reference heading> se colocará en el compartimiento más elevado del <class symbol> contenedor.

<type reference heading> ::=
<type preamble> [<qualifier>] <name> [<formal context parameters>]

<type preamble> tienen que corresponder con <type preamble> del tipo referenciado. Si la referencia es virtual, el tipo referenciado tiene que ser virtual. Si la referencia es abstracta, el tipo referenciado tiene que ser abstracto.

<formal context parameters> corresponde a <formal context parameters> del tipo referenciado. La <formal context parameter list> tiene que corresponder a la <formal context parameter list> del tipo referenciado.

<type reference kind symbol> ::=
« »

El <type reference kind symbol> se coloca por encima o a la izquierda del <type reference heading>.

<data symbol> ::=

NOTA 1 – El <data symbol> es un rectángulo sin ningún marco visible. Esto implica que un <graphical type reference heading> que no contiene <type reference kind symbol> contiene realmente un <data symbol>.

El <data symbol> corresponde a una <data type definition> o una <interface definition>.

Si el <graphical type reference heading> contiene un símbolo diferente de <type reference kind symbol>, dicho símbolo tiene que colocarse en la esquina superior derecha del <graphical type reference heading>.

<iconized type reference area> ::=
<system type symbol> ***contains*** <system type type reference heading>
| <block type symbol> ***contains*** <block type type reference heading>
| <process type symbol> ***contains*** <process type type reference heading>
| <composite state type symbol> ***contains*** <composite state type type reference heading>
| <procedure symbol> ***contains*** <procedure type reference heading>

NOTA 2 – No hay una <iconized type reference area> que corresponda a señales, interfaces, así como a tipos de objeto y valor.

<attribute properties area> ::=
{ { <attribute property> <end> }* }***set***

La primera <attribute property> en una <attribute properties area> debe colocarse en la parte superior del compartimiento central del <class symbol> contenedor. Cada <attribute property> subsiguiente debe colocarse debajo de la precedente.

<behaviour properties area> ::=
{ { <behaviour property> <end> }* }***set***

La primera <behaviour property> en una <behaviour properties area> debe colocarse en la parte superior del compartimiento más bajo del <class symbol> contenedor. Cada <behaviour property> subsiguiente debe colocarse debajo de la precedente.

Modelo

Un <type reference heading> sin <qualifier> antes del <name> es sintaxis derivada en la cual la entidad identificada por <qualifier> es el contexto circundante.

Se considera que una referencia de tipo en la que la identidad identificada por el <qualifier> del <type reference heading> es diferente del contexto circundante ha sido transferida al contexto dado por el calificador, por lo que se aplican las reglas de visibilidad de ese contexto.

Múltiples referencias de tipo en un mismo contexto, que se refieren a la misma clase de entidad y tienen el mismo calificador y el mismo nombre, equivalen a una referencia de tipo, desde ese contexto, con todos los elementos <attribute property> y <behaviour property> de todas las referencias.

Después de reducir múltiples referencias de tipo, la referencia de tipo en la que el <qualifier> del <type reference heading> es el mismo que el contexto circundante se reemplaza por el tipo referenciado definido en 7.3.

NOTA 3 – Cuando, en un modelo para las referencias de tipo, la entidad identificada por el <qualifier> del <type reference heading> es diferente del contexto circundante, ello significa que el tipo referenciado puede ser un tipo, invisible en otro caso, dentro de un ámbito directamente abarcado por el contexto circundante.

8.1.6 Puerta

Las puertas se definen en tipos de agentes (tipos de bloques y tipos de procesos) o en tipos de estados y representan puntos de conexión para canales, que conectan ejemplares de estos tipos (como se definen en 8.1.3) con otros ejemplares o con el símbolo de marco circundante.

También es posible definir puertas en agentes y estados compuestos, representando ello una notación para especificar que la entidad considerada tiene un punto de conexión al que se le ha dado un nombre.

Gramática abstracta

Gate-definition	::	*Gate-name* *In-signal-identifier**-set*** *Out-signal-identifier**-set***
Gate-name	=	*Name*
In-signal-identifier	=	*Signal-identifier*
Out-signal-identifier	=	*Signal-identifier*

Gramática textual concreta

<gate in definition> ::=
<textual gate definition> | <textual interface gate definition>

<textual gate definition> ::=
gate <gate> [**adding**] <gate constraint> [<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>

<textual interface gate definition> ::=
gate { **in** | **out** } **with** <interface identifier> <end>

out o **in** denota el sentido de <signal list>, desde o hacia el tipo, respectivamente. Los tipos a partir de los cuales se definen ejemplares deben tener una <signal list> en las <gate constraint>s.

Un *In-signal-identifier* representa un elemento en la <signal list> hacia la puerta. Un *Out-signal-identifier* representa un elemento en la <signal list> procedente de la puerta.

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 conectado a una puerta debe ser compatible con la constricción de dicha puerta. Un canal es compatible con una constricción de puerta si el otro punto extremo del canal es una agente o un estado del tipo denotado por <identifier> en la constricción del punto extremo o un subtipo de este tipo (en caso de que contenga una <textual endpoint constraint> con **atleast**), y si el conjunto de

señales (si se especifica) en el canal es igual al conjunto de señales especificadas por la puerta en el sentido respectivo, o es un subconjunto del mismo.

Si el tipo denotado por <base type> en una <textual typebased block definition> o <textual typebased process definition> contiene canales, se aplican las reglas siguientes: para cada combinación de (puerta, señal, sentido) definida por el tipo, dicho tipo debe contener al menos un canal que, para el sentido específico, mencione **env** y la puerta y mencione la señal o no tenga <signal list> asociada. En este último caso, es posible que el canal pueda transportar la señal en el sentido dado.

Si el tipo contiene canales que mencionan los procedimientos remotos o las variables remotas, se aplica una regla similar.

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 solamente para una puerta definida en el supertipo. Cuando **adding** se especifica para una <gate>, cualesquiera <textual endpoint constraint>s y <signal list>s son adicionales 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.

El <interface identifier> de una <textual interface gate definition> no deberá identificar la interfaz definida implícitamente por la entidad a que está conectada la puerta (véase 12.1.2).

Gramática gráfica concreta

<gate on diagram> ::=
<graphical gate definition> | <graphical interface gate definition>

<gate property area> ::=
<graphical gate definition> | <graphical interface gate definition>

<graphical gate definition> ::=
{ <gate symbol> | <inherited gate symbol> }
is associated with { <gate> [<signal list area>] [<signal list area>] }***set***
[***is connected to*** <endpoint constraint>]

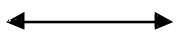
<endpoint constraint> ::=
{ <block symbol> | <process symbol> | <state symbol> }
contains <textual endpoint constraint>

<graphical interface gate definition> ::=
<gate symbol 1>
is associated with <interface identifier>

<gate symbol> ::=
<gate symbol 1> | <gate symbol 2>

<gate symbol 1> ::=

<gate symbol 2> ::=



<inherited gate symbol> ::=
<inherited gate symbol 1> | <inherited gate symbol 2>

<inherited gate symbol 1> ::=

<inherited gate symbol 2> ::=

El <gate on diagram> queda fuera del marco del diagrama.

Una <graphical gate definition> que forma parte de una <gate property area> no podrá contener una <endpoint constraint>.

Los elementos de la <signal list area> están asociados a los sentidos de los símbolos de puerta.

Las <signal list area>s y la <endpoint constraint> asociadas a un <inherited gate symbol> se consideran adiciones a las de la definición de puerta en el supertipo.

Un <inherited gate symbol> sólo puede aparecer en una definición de subtipo, y es entonces representativo 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 una <gate constraint> **in** o **out.**

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 ejemplares.

Modelo

<textual interface gate definition> y <graphical interface gate definition> son representaciones taquigráficas de una <textual gate definition> o una <graphical state definition> respectivamente, siendo <gate name> el nombre de la interfaz y <interface identifier> la <gate constraint> o la <signal list area>.

8.2 Parámetros 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 se sustituyen por parámetros de contexto reales como se define en 8.1.2.

Las definiciones de tipo siguientes pueden tener parámetros de contexto formales: tipo de sistema, tipo de bloque, tipo de proceso, procedimiento, señal, estado compuesto, interfaz y tipo de datos.

Los parámetros de contexto pueden tener constricciones, 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 list> <context parameters end>

<formal context parameter list> ::=
<formal context parameter> {<end> <formal context parameter> }*

<actual context parameters> ::=
<context parameters start> <actual context parameter list> <context parameters end>

<actual context parameter list> ::=
[<actual context parameter>] {**,** [<actual context parameter>] }*

<actual context parameter> ::=
<identifier> | <constant primary>

<context parameters start> ::=
<less than sign>

<context parameters end> ::=
<greater than sign>

<formal context parameter> ::=
<agent type context parameter>
| <agent 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>
| <exception context parameter>
| <composite state type context parameter>
| <gate context parameter>
| <interface context parameter>

La unidad de ámbito de una definición de tipo con parámetros de contexto formales define los nombres de dichos parámetros de contexto formales. Estos nombres son por tanto visibles en la definición del tipo y asimismo en la definición de los parámetros de contexto formal.

Un <actual context parameter> no será un <constant primary>, a menos que sea para un parámetro de contexto de sinónimo. Un <constant primary> es un <primary> que es una <constant expression> válida (véase 12.2.1).

Los parámetros de contexto formal no pueden utilizarse como <base type> en <type expression> ni en constricciones **atleast** de <formal context parameters>.

Las constricciones son especificadas por especificaciones de constricción. Una especificación de constricción introduce la entidad del parámetro de contexto formal seguida por una signatura de constricción o una cláusula **atleast**. Una signatura de constricción 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 únicamente debe estar vinculado a un parámetro de contexto real de la misma clase de entidad que cumpla las constricciones del parámetro formal.

El tipo parametrizado sólo puede utilizar las propiedades de un parámetro de contexto que estén dadas por la constricción, salvo en los casos enumerados en 8.1.2.

Un parámetro de contexto que utiliza otros parámetros de contexto en su constricción no puede estar vinculado antes de que lo estén los demás parámetros.

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>, 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 6.3), se pasan por alto las definiciones locales que no sean <formal context parameters>.

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 constricción del mismo.

Modelo

Si una unidad de ámbito contiene <specialization>, cualquier parámetro de contexto real omitido en la <specialization> es sustituido por el correspondiente <formal context parameter> 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.

8.2.1 Parámetros de contexto de tipo de agente

Gramática textual concreta

<agent type context parameter> ::=
{**process type** | **block type**} <agent type name> [<agent type constraint>]

<agent type constraint> ::=
atleast <agent identifier> | <agent signature>

Un parámetro de tipo de agente real debe ser un subtipo del tipo de agente de constricción (**atleast** <agent type identifier>) sin adición de parámetros formales a los del tipo de constricción, o debe ser compatible con la signatura de agente formal.

Una definición de tipo de agente es compatible con la signatura de agente formal si es de la misma clase y si los parámetros formales de la definición de tipo de agente tienen los mismos géneros que los correspondientes <sort>s de la <agent signature>.

8.2.2 Parámetros de contexto de agente

Gramática textual concreta

<agent context parameter> ::=
{ **process** | **block** } <agent name> [<agent constraint>]

<agent constraint> ::=
{ **atleast** | <colon> } <agent type identifier> | <agent signature>

<agent signature> ::=
<sort list>

Un parámetro de agente real debe identificar una definición de agente. Su tipo debe ser un subtipo del tipo de agente de constricción (**atleast** <agent type identifier>), sin adición de parámetros formales a los del tipo de constricción, o debe ser del tipo denotado por <agent type identifier>, o debe ser compatible con la <agent signature> formal.

Una definición de agente es compatible con la <agent signature> formal si los parámetros formales de la definición de agente tienen los mismos géneros que los correspondientes <sort>s de la <agent signature> o <agent formal parameters>, y ambas definiciones tienen la misma *Agent-kind*.

8.2.3 Parámetros de contexto de procedimiento

Gramática textual concreta

<procedure context parameter> ::=
procedure <procedure name> <procedure constraint>

<procedure constraint> ::=
atleast <procedure identifier> | <procedure signature in constraint>

<procedure signature in constraint> ::=
[**(**<formal parameter> { **,** <formal parameter> }***)**] [<result>]

Un parámetro de procedimiento real debe identificar una definición de procedimiento que es una especialización del procedimiento de la constricción (**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 cada parámetro **in/out** y **out** la definición de procedimiento tiene el mismo <sort identifier> o <syntype identifier> que el parámetro correspondiente de la signatura.

8.2.4 Parámetro de contexto de procedimiento remoto

Gramática textual concreta

<remote procedure context parameter> ::=
remote procedure <procedure name> <procedure signature in constraint>

Un parámetro real de un parámetro de contexto de procedimiento **remote** debe identificar una <remote procedure definition> con la misma signatura.

8.2.5 Parámetros 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>

Un parámetro de señal real debe identificar una definición de señal que es un subtipo del tipo de señal de la constricción (**atleast** <signal identifier>) o compatible con la signatura de señal formal.

Semántica

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 constricciones de género.

8.2.6 Parámetros de contexto de variable

Gramática textual concreta

<variable context parameter> ::=
dcl <variable name> { **,** <variable name>}* <sort>
{ **,** <variable name> { **,** <variable name>}* <sort> }*

Un parámetro real debe ser una variable, un agente formal o un parámetro de procedimiento del mismo género que el género de la constricción.

8.2.7 Parámetros 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> }*

Un parámetro real debe identificar una <remote variable definition> del mismo género.

8.2.8 Parámetros de contexto de temporizador

Gramática textual concreta

<timer context parameter> ::=
timer <timer name> [<timer constraint>]
{ **,** <timer name> [<timer constraint>] }*

<timer constraint> ::=
<sort list>

Un parámetro de temporizador real debe identificar una definición de temporizador que es compatible con la lista de constricciones de género formal. Una definición de temporizador es compatible con una lista de constricciones de género formal si los géneros del temporizador son los mismos que en la lista de constricciones de género.

8.2.9 Parámetros de contexto de sinónimo

Gramática textual concreta

<synonym context parameter> ::=
synonym <synonym name> <synonym constraint>
{, <synonym name> <synonym constraint> }*

<synonym constraint> ::=
<sort>

Un sinónimo real debe ser una expresión constante del mismo género que el género de la constricción.

Modelo

Si el parámetro real es una <constant expression> (y no un <synonym identifier>), hay una definición implícita de un sinónimo anónimo en el contexto que circunda al tipo que se define con el parámetro contexto.

8.2.10 Parámetros de contexto de género

Gramática textual concreta

<sort context parameter> ::=
[{ **value** | **object** }] **type** <sort name> [<sort constraint>]

<sort constraint> ::=
atleast <sort> | <sort signature>

<sort signature> ::=
literals <literal signature> { **,** <literal signature> }*
[**operators** <operation signature in constraint> { **,** <operation signature in constraint> }*]
[**methods** <operation signature in constraint> { **,** <operation signature in constraint> }*]
| **operators** <operation signature in constraint> { **,** <operation signature in constraint> }*
[**methods** <operation signature in constraint> { **,** <operation signature in constraint> }*]
| **methods** <operation signature in constraint> { **,** <operation signature in constraint> }*

<operation signature in constraint> ::=
<operation name>[**(**<formal parameter> { **,** <formal parameter> }* **)**] [<result>]
| <name class operation> [<result>]

Si se omite <sort constraint>, el género real puede ser cualquier género. En los demás casos, un género real debe ser un subtipo sin <renaming> del género de la constricción (**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, las operaciones definidas por los tipos de datos que han introducido el género incluyen las operaciones en la signatura de género formal y las operaciones tienen las mismas signaturas.

La <literal signature> no podrá contener <named number>.

Modelo

Si se da la palabra clave **value** y el género real es un género de objeto, el parámetro real se trata como el género expandido **value** <sort identifier>. Si se da la palabra clave **object** y el género real es un género de valor, el parámetro real se trata como el género de referencia **object** <sort identifier>.

8.2.11 Parámetros de contexto de excepción

Gramática textual concreta

<exception context parameter>::=
exception <exception name> [<exception constraint>]
{ **,** <exception name> [<exception constraint>] }*

<exception constraint>::=
<sort list>

Un parámetro de excepción real debe identificar una excepción con la misma signatura.

8.2.12 Parámetros de contexto de tipo de estado compuesto

Gramática textual concreta

<composite state type context parameter> ::=
state type <composite state type name> [<composite state type constraint>]

<composite state type constraint> ::=
atleast <composite state type identifier> | <composite state type signature>

<composite state type signature> ::=
<sort list>

Un parámetro de tipo de estado compuesto real debe identificar una definición de tipo de estado compuesto. Su tipo debe ser un subtipo del tipo de estado compuesto de constricción (**atleast** <composite state type identifier>), sin adición de parámetros formales a los del tipo de constricción, o ser compatible con la signatura de tipo de estado compuesto formal.

Una definición de tipo de estado compuesto es compatible con la signatura de tipo de estado compuesto formal si los parámetros formales de la definición de tipo de estado compuesto tienen los mismos géneros que los correspondientes <sort>s de la <composite state type constraint>.

8.2.13 Parámetros de contexto de puerta

Gramática textual concreta

<gate context parameter> ::=
gate <gate> <gate constraint> [<gate constraint>]

Un parámetro de puerta real debe identificar una definición de puerta. Su constricción de puerta en sentido salida debe contener todos los elementos mencionados en la <signal list> del correspondiente parámetro de contexto de puerta formal. La constricción de puerta en sentido entrada del parámetro de contexto de puerta formal debe contener todos los elementos de la <signal list> del parámetro de puerta real.

8.2.14 Parámetro de contexto de interfaz

Gramática textual concreta

<interface context parameter> ::=
interface <interface name> [<interface constraint>]
{ **,** <interface name> [<interface constraint>] }*

<interface constraint> ::=
atleast <interface identifier>

Un parámetro de interfaz real debe identificar una definición de interfaz. El tipo de la interfaz debe ser un subtipo del tipo de la constricción (**atleast** <interface identifier>).

8.3 Especialización

Un tipo puede definirse en una especialización de otro tipo (el supertipo), produciendo un nuevo subtipo. Un subtipo puede tener propiedades adicionales a las del supertipo, y puede redefinir tipos locales virtuales y transiciones. Salvo en el caso de interfaces, hay a lo sumo un supertipo.

Los tipos virtuales pueden tener constricciones, 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 8.3.2.

8.3.1 Adición de propiedades

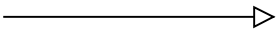
Gramática textual concreta

<specialization> ::=
inherits <type expression> [**adding**]

Gramática gráfica concreta

<specialization area> ::=
<specialization relation symbol>
[***is associated with*** <actual context parameters>]
is connected to <type reference area>

<specialization relation symbol> ::=



La punta de flecha del <specialization relation symbol> apunta hacia la <type reference area>. El tipo conectado a la punta de flecha es el supertipo, mientras que el otro tipo es el subtipo. Las referencias conectadas deben ser ambas de la misma clase. La vinculación asociada de parámetros de contexto se corresponde con el supertipo que es una expresión de tipo con parámetros de contexto reales.

<type expression> denota el tipo base. Se dice que el tipo base es el supertipo del tipo especializado y que el tipo especializado es un subtipo del tipo base. Cualquier especialización del subtipo es asimismo un subtipo del tipo base.

Si un tipo subT es un subtipo de un (súper) tipo T (directa o indirectamente), entonces:

a) T no debe contener subT;

b) T no debe ser una especialización de subT;

c) las definiciones contenidas por T no deben ser especializaciones de subT.

En el caso de tipos de agentes, estas reglas también se aplican para definiciones contenidas en T y, además, las definiciones directa o indirectamente contenidas por T no deben ser definiciones basadas en tipo de subT.

La sintaxis concreta para especialización de tipos de datos se muestra en 12.1.3.

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 6.3. No obstante, hay tres excepciones a esta regla:

a) una redefinición de un tipo virtual es una definición con el mismo nombre que el del tipo virtual;

b) una puerta del supertipo puede tener una definición ampliada (en términos de señales transportadas y constricciones de punto extremo) en un subtipo – esto es, especificada por una definición de puerta con el mismo nombre que el del supertipo;

c) si la <type expression> contiene <actual context parameters>, cualquier ocurrencia del <base type> de la <type expression> es sustituida por el nombre del subtipo;

d) un operador del supertipo no se hereda si la signatura del operador especializado es la misma que la del operador de tipo base;

e) un operador o un método de tipo no virtual (es decir, un método que no es virtual ni redefinido) del supertipo no se hereda si otro operador o método con la misma signatura que el operador o método especializado está presente en el subtipo.

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 8.2).

Los parámetros formales de un tipo de agente especializado son los parámetros formales del supertipo de agente seguidos de los parámetros formales añadidos en la especialización.

Los parámetros formales de un tipo de un procedimiento especializado son los parámetros formales del procedimiento con los parámetros formales añadidos en la especialización. Si el procedimiento anterior a la especialización tiene un <procedure result>, los parámetros añadidos en la especialización se insertan antes del último parámetro (el parámetro **out** del resultado), en otro caso, se insertan después del último parámetro.

El conjunto completo de señales de entrada válidas de un tipo de agente especializado es la unión del conjunto completo de señales de entrada válidas del tipo de agente especializado y el conjunto completo de señales de entrada válidas del supertipo de agente, respectivamente.

El gráfico resultante de un tipo de agente especializado, o definición de procedimiento o tipo de estado consiste en el gráfico de su definición de supertipo seguido por el gráfico del agente especializado, definición de procedimiento o tipo de estado.

El gráfico de transición de estados de un tipo de agente dado, definición de procedimiento o tipo de estado puede tener a lo sumo una transición de arranque no etiquetada.

Una definición de señal especializada puede añadir (adjuntando) géneros a la lista de géneros del supertipo.

Una definición de tipo de datos especializada puede añadir literales, campos o selecciones a los constructores de tipo heredados, e igualmente puede añadir operadores, métodos, inicializaciones por defecto o asignación por defecto.

Los parámetros formales de un tipo de estado compuesto especializado son los parámetros formales del tipo de estado compuesto con los parámetros formales añadidos en la especialización.

NOTA – Cuando una puerta en un subtipo es una extensión de una puerta heredada de un supertipo, el <inherited gate symbol> se utiliza en la sintaxis gráfica concreta.

8.3.2 Tipo virtual

Un tipo de agente, tipo de estado o procedimiento puede especificarse como un tipo virtual cuando está definido localmente con una definición de tipo (denotado como tipo *enclosing*). Un tipo virtual puede estar redefinido en especializaciones de su tipo circundante.

Gramática textual concreta

<virtuality> ::=
virtual | redefined | finalized

<virtuality constraint> ::=
atleast <identifier>

<virtuality> y <virtuality constraint> forman parte de la definición de tipo.

Un tipo virtual es un tipo cuyo <virtuality> es **virtual** o **redefined**. Un tipo redefinido es un tipo cuyo <virtuality> es **redefined** o **finalized**. Se deben redefinir únicamente los tipos virtuales.

Cada tipo virtual tiene asociada una constricción de virtualidad que es un <identifier> de la misma clase de entidad que el tipo virtual. Si se especifica la <virtuality constraint>, la constricción de virtualidad es la que está contenida en <identifier>, en otro caso, la constricción de virtualidad se obtiene tal como se describe a continuación.

Un tipo virtual y sus constricciones no pueden tener parámetros de contexto.

Sólo los tipos virtuales pueden tener una <virtuality constraint> especificada.

Si <virtuality> está presente en la referencia y en la definición referenciada, ambas deben ser iguales. Si <procedure preamble> está presente en la referencia de procedimiento y en la definición de procedimiento referenciado, ambos deben ser iguales.

Un tipo de agente virtual debe tener exactamente los mismos parámetros formales y, al menos, las mismas puertas e interfaces con, al menos, las definiciones como las de su constricción. Un tipo de estado virtual debe tener al menos los mismos puntos de conexión de estado que su constricción. Un procedimiento virtual debe tener exactamente los mismos parámetros formales que su constricción. Las restricciones de los argumentos de los operadores y métodos virtuales se presentan en 8.3.4.

Si se utilizan **inherits** y **atleast**, el tipo heredado debe ser idéntico a la constricción o ser un subtipo de la misma.

En caso de una constricción implícita, la redefinición que implique **inherits** debe ser un subtipo de la constricción.

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 ejemplares 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 en un único ítem de trayecto que denote el supertipo en cuestión.

Un tipo virtual o redefinido que no tenga una <specialization> dada explícitamente, puede tener una <specialization> implícita. La constricción de virtualidad y la posible <specialization> implícita se obtienen como se indica a continuación.

Para un tipo virtual V y un tipo redefinido R de V, se aplican las reglas siguientes (todas las reglas se aplican en el orden dado):

a) si el tipo virtual V no tiene <virtuality constraint>, la constricción VC para el tipo V coincide con el tipo virtual V y denota a dicho tipo V, en otro caso, la constricción VC es identificada por la <virtuality constraint> dada con el tipo V;

b) si el tipo virtual V no tiene <specialization> y la constricción VC es del tipo V, el tipo V no tiene una especialización implícita;

c) si el tipo virtual V no tiene <specialization> y la constricción VC no es del tipo V, el tipo de especialización implícita VS es el mismo que la constricción VC;

d) si <specialization> del tipo virtual V está presente, el tipo de especialización VS debe ser el mismo o un subtipo de la constricción VC;

e) si el tipo redefinido R no tiene <virtuality constraint>, la constricción RC para el tipo R es la misma que el tipo R; en otro caso, la constricción RC es identificada por la <virtuality constraint> dada con el tipo R;

f) si el tipo redefinido R no tiene <specialization>, el tipo de especialización implícita RS para R es la misma que la constricción VC del tipo V; en otro caso, el tipo de especialización RS es identificado por la <specialization> explícita con el tipo R;

g) la constricción RC debe ser igual a la constricción VC o ser un subtipo de la misma;

h) el tipo de especialización RS para R debe ser igual a la constricción VC o ser un subtipo de la misma

i) si R es un tipo virtual (redefinido más que finalizado), se aplica la misma regla para R que para V.

Un subtipo de un tipo virtual es un subtipo del tipo virtual original y no de una posible redefinición.

8.3.3 Transición/conservación virtual

Las transiciones o conservaciones de un tipo de proceso, tipo de estado 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 textual concreta

La sintaxis de una transición y conservación virtuales se presentan en 9.4 (arranque de procedimiento virtual), 10.5 (entrada y conservación de procedimiento remoto virtual), 11.1 (arranque de proceso virtual), 11.2.2 (entrada virtual), 11.4 (entrada de prioridad virtual), 11.5 (señal continua virtual), 11.7 (conservación virtual), 11.9 (transición espontánea virtual) y 11.16.3 (manejo virtual).

Las transiciones o conservaciones virtuales no deben aparecer en definiciones (de conjunto de ejemplares) de agente, ni en definiciones de estados compuestos.

Un estado no debe tener más de una transición espontánea virtual.

Una redefinición en una especialización marcada con **redefined** puede definirse de forma diferente en otras especializaciones ulteriores, mientras que una redefinición marcada con **finalized** no debe recibir nuevas definiciones 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 8.3.2).

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 de tratamiento virtual puede redefinirse en una nueva transición de tratamiento.

Una transición continua virtual puede redefinirse en una nueva transición continua. La redefinición es indicada por el mismo estado y prioridad (si está presente) que la transición continua redefinida. Si en un estado existen varias transiciones continuas virtuales, cada una de ellas debe tener una prioridad distinta. Si en un estado sólo existe una transición continua virtual, 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 como una conservación de procedimiento remoto.

La transformación para transición de entrada virtual también se aplica a la transición de entrada de procedimiento remoto virtual.

La transformación de transiciones y conservaciones virtuales en estado asterisco se desarrolla en el anexo F.

8.3.4 Métodos virtuales

Los métodos de un tipo de datos se especifican como virtuales mediante la palabra clave **virtual** en <virtuality>. Los métodos virtuales pueden redefinirse en especializaciones. Ello se indica mediante métodos que tienen el mismo <operation name> y la palabra clave **redefined** o **finalized** en <virtuality>.

Si el tipo derivado contiene solamente una <operation signature> pero no <operation definition>, <textual operation reference>, o <external operation definition> para el método redefinido, solamente se modifica la signatura del método redefinido.

Gramática textual concreta

La sintaxis de los métodos virtuales se presenta en 12.1.4.

Cuando un método se redefine en una especialización, su signatura debe ser compatible en género con la correspondientes signatura del tipo base, y aún más, si el *Result* de la *Operation-signature* denota un género A, el *Result* del método redefinido sólo puede denotar un género B tal que B es compatible con A.

La redefinición de un método virtual no debe modificar la <parameter kind> en ningún <argument> de la <operation signature> heredada.

La redefinición de un método virtual no debe añadir la <argument virtuality> a ningún <argument> de la <operation signature> heredada.

Semántica

Los métodos virtuales no tienen una <virtuality constraint> que, solamente en este caso, no limita la redefinición.

La redefinición de los métodos virtuales se corresponde estrechamente con la redefinición de los tipos virtuales (véase 8.3.2).

8.3.5 Inicialización por defecto virtual

En esta subcláusula se describe la inicialización por defecto virtual, tal como se presenta en 12.3.3.2.

La inicialización por defecto de ejemplares de un tipo de datos se especifican como virtuales mediante la palabra clave **virtual** en <virtuality>. Una inicialización por defecto virtual puede redefinirse en especializaciones. Ello se indica mediante una inicialización por defecto con la palabra clave **redefined** o **finalized** en <virtuality>.

Si el tipo derivado no contiene ninguna <constant expression> en su inicialización por defecto, el tipo derivado carece de una inicialización por defecto.

Gramática textual concreta

La sintaxis de las inicializaciones por defecto se presenta en 12.3.3.2.

Semántica

La redefinición de una inicialización por defecto virtual se corresponde estrechamente con una redefinición de tipos virtuales (véase 8.3.2).

8.4 Asociaciones

Una asociación expresa una relación binaria entre dos tipos de entidades no necesariamente distintas. Con las asociaciones se pretende proporcionar anotaciones estructuradas para indicar propiedades adicionales de los tipos a los que las asociaciones están conectados, en un diagrama o definición que contiene referencias de tipo. El significado de estas propiedades no se define en la presente Recomendación; esto es, el significado puede estar definido por otra Recomendación o norma, o por una especificación común, o por un entendimiento común. Un sistema SDL que contenga una asociación tiene el mismo significado y comportamiento (tal como se define en esta Recomendación) si se suprime la asociación.

Gramática textual concreta

<association> ::=
association [<association name>] <association kind> <end>

<association kind>::=
<association not bound kind>
| <association end bound kind>
| <association two ends bound kind>
| <composition not bound kind>
| <composition part end bound kind>
| <composition composite end bound kind>
| <composition two ends bound kind>
| <aggregation not bound kind>
| <aggregation part end bound kind>
| <aggregation aggregate end bound kind>
| <aggregation two ends bound kind>

<association not bound kind> ::=
from <association end> **from** <association end>

<association end bound kind>::=
from <association end> **to** <association end>

<association two ends bound kind>::=
to <association end> **to** <association end>

<composition not bound kind>::=
from <association end> **composition** <association end>

<composition part end bound kind>::=
to <association end> **composition** <association end>

<composition composite end bound kind> ::=
from <association end> **to composition** <association end>

<composition two ends bound kind>::=
to <association end> **to composition** <association end>

<aggregation not bound kind>::=
from <association end> **aggregation** <association end>

<aggregation part end bound kind>::=
to <association end> **aggregation** <association end>

<aggregation aggregate end bound kind>::=
from <association end> **to aggregation** <association end>

<aggregation two ends bound kind>::=
to <association end> **to aggregation** <association end>

<association end> ::=
[<visibility>] [**as** <role name>] <linked type identifier> [**size** <multiplicity>] [**ordered**]

<linked type identifier> ::=
<agent type identifier>
| <data type identifier>
| <interface identifier>

<multiplicity> ::=
<range condition>

Una <association> puede enlazar tipos de agentes, interfaces o tipos de datos.

Si una <association end> identifica un tipo de agente o una interfaz, la visibilidad **protected** no puede ser utilizada en el otro <association end> de la <association>.

Si dos <association>s distintas identifican el mismo tipo, los <role name>s (si es que existen) de los <association end>s opuestos a dicho tipo común deben ser diferentes.

El género base de la <range condition> en <multiplicity> debe ser el género natural predefinido.

No debe de haber un conjunto de <association>s que contengan una composición tal que un tipo se enlace mediante composición consigo mismo, ya sea directa o indirectamente.

Si un <association end> está precedido por la palabra clave **composition** e identifica un tipo de datos o interfaces, el <linked type identifier> del otro <association end> de la <association> debe identificar un tipo de datos o una interfaz respectivamente.

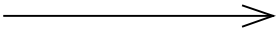
Gramática gráfica concreta

<association area> ::=
<association symbol>
[***is associated with*** <association name>]
is connected to {<association end area> <association end area>} ***set***

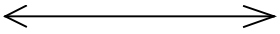
<association symbol> ::=
<association not bound symbol>
| <association end bound symbol>
| <association two ends bound symbol>
| <composition not bound symbol>
| <composition part end bound symbol>
| <composition composite end bound symbol>
| <composition two ends bound symbol>
| <aggregation not bound symbol>
| <aggregation part end bound symbol>
| <aggregation aggregate end bound symbol>
| <aggregation two ends bound symbol>

<association not bound symbol> ::=

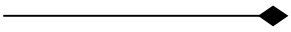
<association end bound symbol>::=



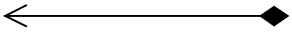
<association two ends bound symbol>::=



<composition not bound symbol> ::=



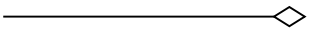
<composition part end bound symbol> ::=



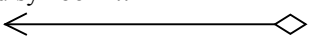
<composition composite end bound symbol> ::=

<composition two ends bound symbol> ::=

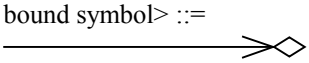
<aggregation not bound symbol> ::=



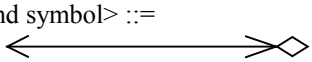
<aggregation part end bound symbol> ::=



<aggregation aggregate end bound symbol> ::=



<aggregation two ends bound symbol> ::=



<association end area> ::=
<linked type reference area> ***is associated with***
{ [<role name>] [<multiplicity>] [<ordering area>] [<symbolic visibility>] }***set***

<ordering area> ::=
ordered

<linked type reference area> ::=
<agent type reference area>
| <data type reference area>
| <interface reference area>

Una <association area> que conecta dos <linked type reference area>s corresponde a una <association> del diagrama circundante.

Si una <association end area> identifica un tipo de agente o una interfaz, la visibilidad **protected** no se utilizará en la otra <association end area> de la <association area>.

Si dos <association area>s distintas identifican el mismo tipo, los <role name>s (si es que existen) de las <association end area>s opuestas a dicho tipo común deben ser diferentes.

No debe de haber un conjunto de <association area>s que contengan una composición tal que un tipo se enlace mediante composición consigo mismo, ya sea directa o indirectamente.

Si el extremo compuesto (el extremo con un diamante) de un <composition not bound symbol>, <composition part end bound symbol>, <composition composite end bound symbol>, o <composition two ends bound symbol> está conectado a una <linked type reference area> que identifica un tipo de datos o una interfaz, la <association end area> opuesta debe estar conectada a una <linked type reference area> que identifique un tipo de datos o una interfaz, respectivamente.

Semántica

Una asociación enlaza los dos tipos de entidades de una forma que no se define en esta Recomendación.

9 Agentes

Una definición de un agente define un conjunto (arbitrariamente grande) de agentes. Un agente se caracteriza por el hecho de tener variables, procedimientos, una máquina de estado (dada por un tipo de estado compuesto explícito o implícito) y conjuntos de agentes contenidos.

Existen dos clases de agentes: *bloques* y *procesos*. Un *sistema* es el bloque más externo. La máquina de estados de un bloque se interpreta *concurrentemente* con los agentes contenidos, mientras que la máquina de estados de un proceso se interpreta *alternativamente* con sus agentes contenidos.

Gramática abstracta

Agent-definition	::	*Agent-name* *Number-of-instances* *Agent-type-identifier*
State-machine-definition	::	*State-name* *Composite-state-type-identifier*
Agent-kind	=	**SYSTEM** | **BLOCK** | **PROCESS**
Number-of-instances	::	*Initial-number* [*Maximum-number*]
Initial-number	=	*Nat*
Maximum-number	=	*Nat*
State-transition-graph	::	[*On-exception*] *State-start-node* *State-node-**set*** *Free-action-**set*** *Exception-handler-node-**set***
Agent-formal-parameter	=	*Parameter*

Gramática textual concreta

```
<agent definition> ::=
                        <system definition> | <block definition> | <process definition>

<agent structure> ::=
                        [<valid input signal set>]
                        {       {       <entity in agent>
                                |       <channel to channel connection>
                                |       <channel definition>
                                |       <gate in definition>
                                |       <agent definition>
                                |       <agent reference>
                                |       <textual typebased agent definition> }*
                                [ <state partitioning>]
                        |       {       <entity in agent>
                                |       <gate in definition> }*
                                <agent body> }
```

La <state partitioning> debe tener el mismo nombre que el agente que la contiene. Define la máquina de estados del agente. Si <agent structure> puede interpretarse como <state partitioning> y como <agent body>, se interpreta como <agent body>.

```
<agent instantiation> ::=
                        [<number of instances>]
                        <agent additional heading>

<agent additional heading> ::=
                        [<specialization>] [<agent formal parameters>]
```

<entity in agent> ::=
<signal definition>
| <signal reference>
| <signal list definition>
| <variable definition>
| <remote procedure definition>
| <remote variable definition>
| <data definition>
| <data type reference>
| <timer definition>
| <interface reference>
| <macro definition>
| <exception definition>
| <procedure reference>
| <procedure definition>
| <composite state type definition>
| <composite state type reference>
| <select definition>
| <agent type definition>
| <agent type reference>
| <association>

<agent reference> ::=
<block reference>
| <process reference>

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

<agent body> ::=
<state machine graph>

<state machine graph>::=
[<on exception>] <start>
{ <state> | <exception handler> | <free action> } *

<agent formal parameters> ::=
(<parameters of sort> {**,** <parameters of sort>}* **)**

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

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

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

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

Lo que sigue es válido para agentes en general. Las propiedades especiales de los sistemas, bloques y procesos se tratan en cláusulas distintas y específicas sobre dichos conceptos.

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

El <number of instances> empleado en la derivación arriba mencionada es el siguiente:

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

b) Si se omiten tanto <number of instances> en <agent definition> como <number of instances> en <agent reference>, entonces se utiliza el <number of instances> en el que se omiten <initial number> y <maximum number>.

c) Si se omite <number of instances> en <agent definition> o <number of instances> en <agent reference>, entonces <number of instances> está presente.

d) Si se especifican tanto <number of instances> en <agent definition> como <number of instances> en <agent reference>, los dos <number of instances> deben ser iguales desde un punto de vista léxico, utilizándose dicho <number of instances>.

Relaciones similares se aplican a las <number of instances> especificadas en <agent diagram> y en <agent reference area> tal como de define a continuación.

El <initial number> de ejemplares debe ser inferior o igual a <maximum number> y <maximum number> debe ser mayor de cero.

Si <agent formal parameters> está presente en <agent instantiation>, también debe estar presente <number of instances>, incluso si <initial number> y <maximum number> se omiten.

La utilización y sintaxis de <valid input signal set> se define en la cláusula 9.

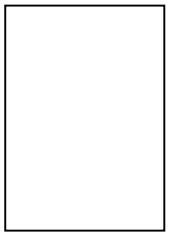
Gramática gráfica concreta

<agent diagram> ::=
<system diagram> | <block diagram> | <process diagram>

<agent diagram content> ::=
{ {<agent text area>}*
{<type in agent area>}*
{ <interaction area> | <agent body area> } }***set***

<agent body area> ::=
<state machine graph area>

<frame symbol> ::=



La <package use area> debe estar ubicada por encima de <frame symbol> del <system diagram>, <block diagram>, o <process diagram>.

<agent text area> ::=
<text symbol>
contains {
[<valid input signal set>]
{ <signal definition>
| <signal reference>
| <signal list definition>
| <variable definition>
| <remote procedure definition>
| <remote variable definition>
| <data definition>
| <data type reference>
| <timer definition>
| <interface reference>
| <macro definition>
| <exception definition>
| <procedure definition>
| <procedure reference>
| <select definition>
| <agent type reference>
| <agent reference> }* }

<state machine graph area>::=
{ [<on exception association area>] <start area>
{ <state area> | <exception handler area> | <in connector area> }* }***set***

<interaction area> ::=
{{ <agent area>
| <create line area>
| <channel definition area>
| <state machine area> }+ }***set***

<agent area> ::=
<agent reference area>
| <agent diagram>
| <graphical typebased agent definition>
| <inherited agent definition>

<agent reference area> ::=
{ <system reference area>
| <block reference area>
| <process reference area> }
[***is connected to*** {<graphical package use area>+ }***set***]

<state machine area> ::=
<state partition area>

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

<create line endpoint area> ::=
<agent area> | <agent type area> | <state machine area>

<agent type area> ::=
<agent type reference area>
| <agent type diagram>

<create line symbol> ::=
<dependency symbol>

Se permite que una <agent text area> contenga una <agent reference> únicamente si el <agent diagram content> directamente circundante contiene una <intersection area>.

En un <agent diagram>, el <gate on diagram> tiene que estar fuera del marco del diagrama.

El *Agent-definition*-**set** de la *Gramática abstracta* del tipo de agente implicado (véase *Modelo*) corresponde a las <agent area>.

El *Channel-definition*-**set** de la *Gramática abstracta* del tipo de agente implicado corresponde a las <channel definition area>.

La punta de flecha del <create line symbol> indica la <agent area> o <agent type area> de un agente sobre el que se realiza la acción de creación. <create line symbol>s son opcionales, pero si se utilizan, debe existir una petición de creación para el agente (o tipo de agente) que se encuentra en la punta de flecha del <create line symbol> del agente o del tipo de agente en el extremo origen del <create line symbol>. La acción de creación puede ser heredada y no necesita ser especificada directamente en el agente o en el tipo de agente. Esta regla se aplica después de la transformación de <option area>.

NOTA 1 – Esta regla puede aplicarse independientemente antes o después de la transformación de <transition option>.

La <state machine area> de <interaction area> identifica la máquina de estados (estado compuesto) del agente que puede darse directamente como un gráfico de agente o referenciado como una definición de estado.

Si existe una <agent reference area> para el agente, tiene que haber una <gate property area> conectada a la <agent reference area> o tiene que haber una <gate> contenida en el símbolo para

cada <gate on diagram> conectada al <agent diagram>. Si el <gate on diagram> es una <graphical gate definition>, la <gate property area> tiene que ser una <graphical gate definition> o <gate>, respectivamente, que menciona el mismo <gate name>. Si el <gate on diagram> es una <graphical interface gate definition>, la <gate property area> tiene que ser una <graphical interface gate definition> que menciona el mismo <interface identifier>. Se aplica una regla correspondiente si hay una <agent reference> para el agente.

Una <graphical package use area> conectada a una <agent reference area> debe ser consistente con la <graphical package use area> del diagrama referenciado.

Semántica

Una *Agent-definition* tiene un nombre que puede utilizarse conjuntamente con **system**, **block** o **process** dependiendo de la clase de agente de que se trate.

Una definición de agente define un conjunto de agentes. Pueden existir al mismo tiempo varios ejemplares del mismo agente y ser interpretados asíncronamente y en paralelo o de forma alternativa uno respecto a otro y con ejemplares de otros conjuntos de agentes en el sistema.

El primer valor de *Number-of-instances* representa el número de ejemplares del conjunto de agentes que existen cuando se crea el sistema o la entidad contenedora (ejemplares iniciales), el segundo valor representa el número máximo de ejemplares simultáneos del conjunto de agentes.

El comportamiento de una *Agent-definition* en un *Agent-definition*-**set** depende de si la *Agent-definition* contenedora es un bloque o un proceso y, por lo tanto, se define para bloque y para proceso de forma separada.

Un ejemplar de agente tiene una máquina de estados finitos ampliada que se comunica y que está definida por su definición explícita o implícita de máquina de estados. Siempre que la máquina se encuentre en un estado, entrada de una señal determinada, realizará una secuencia de actuaciones que se denotan como transición. La realización completa de la transición hace que la máquina de estados del ejemplar del agente quede en espera en otro estado, que no tiene que ser necesariamente distinto del primero.

Cuando se interpreta un agente, se crean los agentes iniciales que contiene. La comunicación de señales entre las máquinas de estados finitos de dichos agentes iniciales, la máquina de estados finitos del agente y su entorno, comienza únicamente cuando todos los agentes iniciales se han creado. El tiempo necesario para crear un agente puede o no ser significativo. Los parámetros formales de los agentes iniciales no tienen ítems de datos asociados (son "indefinidos").

Los ejemplares de agentes existen desde el momento en que el agente contenedor se crea o bien, pueden ser creados mediante acciones de petición de creación de los agentes que han sido interpretados; sus interpretaciones arrancan cuando se interpreta su acción de arranque; pueden dejar de existir mediante acciones de parada.

Cuando la máquina de estados de un agente interpreta una parada, y si dicho agente era un contenedor concurrente, éste continúa tratando las llamadas a procedimientos remotos implícitos que median para el acceso a las variables globales. La máquina de estados de dicho agente permanece en dicha "condición de parada" hasta que todos los agentes contenidos han terminado, tras lo cual termina el agente. Mientras se está en la condición de parada, el agente no aceptará ningún estímulo distinto del conjunto implícito y recibe llamadas a procedimiento remotos que se realizan para cada variable global, si existe alguna. Después de que un agente haya terminado, su pid deja de ser válido.

Las señales que reciben los ejemplares de agentes se denotan como señales de entrada, mientras que las señales enviadas por los ejemplares de agentes se denotan como señales de salida. El <valid input signal set> de un agente define el <valid input signal set> de su máquina de estados.

Las acciones de llamar y atender llamadas a procedimientos remotos y de acceder a variables remotas, corresponden igualmente al intercambio de señales (véanse 10.5 y 10.6 respectivamente).

Las señales sólo pueden ser consumidas por la máquina de estado de un ejemplar de agente cuando éste se encuentra en un estado. El conjunto completo válido de señales de entrada es la unión de lo siguiente:

a) el conjunto de señales de todos los canales o puertas que conducen al conjunto de ejemplares de agentes;

b) el <valid input signal set>;

c) las señales de entrada implícitas introducidas como en 10.5, 10.6, 11.4, 11.5 y 11.6; y

d) las señales del temporizador.

Se asocia exactamente un puerto de entrada a la máquina de estados finitos de cada ejemplar de agente. Las señales enviadas a un agente contenedor se entregan al puerto de entrada del agente, siempre que la señal aparezca en un canal (explícito o implícito) conectado a su máquina de estados. Las señales que sólo ocurran en el <valid input signal set> no deben utilizarse para comunicación externa. Sirven para establecer la comunicación entre ejemplares del mismo conjunto de ejemplares.

La máquina de estados finitos de una agente se encuentra en un estado de espera o activo, realizando una transición. Para cada estado existe un conjunto de señales de conservación (véase también 11.7). Cuando está en estado de espera, la primera señal de entrada cuyo identificador no se encuentra en el conjunto de señales de conservación se toma de la cola y es consumida por el agente. Una transición puede también iniciarse como una transición espontánea independiente de las señales que se encuentren en la cola.

El puerto de entrada puede retener cualquier número de señales de entrada, de forma que en la cola de la máquina de estados finitos del ejemplar de agente puede haber varias señales de entrada. El conjunto de señales retenidas se ordena en la cola de acuerdo al momento de llegada de las señales. Si dos o más señales llegan "simultáneamente" a través de trayectos distintos, se ordenan de forma arbitraria.

Cuando se crea el agente, su máquina de estados finitos recibe un puerto de entrada vacío, creándose variables locales del agente.

Cuando se crea un ejemplar de agente contenedor, se crean los agentes iniciales de los conjuntos de agentes contenidos. Si el contenedor se crea mediante <create request>, el **parent** de los agentes contenidos (véase el *Modelo*) es el pid del contenedor. Los parámetros formales son variables que se crean cuando el sistema es creado (pero a ellos no se pasan parámetros reales, y por lo tanto, son "indefinidos") o cuando el ejemplar del agente se crea dinámicamente.

Si una <agent definition> o una <agent type definition>, utilizada en una <textual typebased agent definition>, contiene <channel definition>s y <textual typebased agent definition>s, entonces, cada puerta de las <agent type definition>s de las <textual typebased agent definition>s contenidas debe conectarse a un canal.

La definición de un agente implica la definición de una interfaz en el mismo ámbito del agente (véase 12.1.2). El genero de pid definido implícitamente por esta interfaz se identifica con *Agent-name* y es visible en la misma unidad de ámbito en la que está definido el agente.

NOTA 2 – Como cada agente tiene una interfaz implícitamente definida con el mismo nombre, el agente debe tener un nombre distinto de cada interfaz explícitamente definida, y cada tipo de agente (éstos también tienen interfaces implícitas) definido en el mismo ámbito; de lo contrario, hay coincidencia de nombres.

Modelo

Una *Agent-definition* tiene implícito un tipo de agente anónimo que define las propiedades del agente.

Un agente con un <agent body> o una <agent body area> es la expresión taquigráfica de un agente que sólo tiene una máquina de estados, pero que no contiene agentes. Esta máquina de estados se obtiene sustituyendo al <agent body> o <agent body area> por una definición de estado compuesto.

Esta definición de estado compuesto tiene el mismo nombre que el agente y su *State-transition-graph* se representa mediante el <agent body> o la <agent body area>.

Un agente que es una especialización es una expresión taquigráfica para definir un tipo de agente implícito y un agente basado en tipo de dicho tipo.

En todos los ejemplares de agente se declaran cuatro variables anónimas del género pid del agente (para agentes no basados en un tipo de agente) o del género pid del tipo de agente (para agentes basados en tipo) y, en lo que sigue, se hará referencia a ellas como **self**, **parent**, **offspring** y **sender**. Proporcionan un resultado para:

a) el ejemplar de agente (**self**);

b) el ejemplar de agente creador (**parent**);

c) el ejemplar de agente creado más recientemente por el ejemplar de agente (**offspring**);

d) el ejemplar de agente del que se ha consumido la última señal de entrada (**sender**) (véase también 11.2.2).

A estas variables anónimas se accede utilizando expresiones pid tal como se explica en 12.3.4.3.

El ejemplar de agente durante la inicialización del sistema, **parent** se inicializa con el valor Null.

Para todos los ejemplares de agente recientemente creadas, **sender** y **offspring** se inicializan con el valor Null.

9.1 Sistema

Un sistema es el agente más externo y tiene el *Agent-kind* **SYSTEM**. Se define por una <system definition> o un <system diagram>. La gramática y la semántica de los agentes se aplican con las adiciones que se proporcionan en esta subcláusula.

Gramática abstracta

Un *Agent* con el *Agent-kind* **SYSTEM** no debe estar contenido en otro *Agent*. Debe contener al menos un *Agent-definition* o una *State-machine-definition* explícita o implícita.

Las definiciones de todas las señales, canales, tipos de datos y sintipos utilizados en la interfaz con el entorno y entre los agentes contenidos del sistema (incluyendo él mismo) están contenidos en la *Agent-definition* del sistema.

El *Initial-number* de ejemplares es 1 y el *Maximum-number* de ejemplares es 1.

NOTA – <number of instances> no puede especificarse.

Gramática textual concreta

<system definition> ::=
{<package use clause>}*
<system heading> <end> <agent structure>
endsystem [<system name>] <end>

<system heading>::=
system <system name> <agent additional heading>

El <agent additional heading> en una <system definition> o <system diagram> no puede incluir <agent formal parameters>.

Gramática gráfica concreta

<system diagram> ::=
<frame symbol> ***contains*** {<system heading> <agent diagram content> }
[***is associated with*** <package use area>]

<system reference area> ::=
<block symbol> ***contains***
{ **system** <system name> }

Los <gate on diagram>s de un <system diagram> puede que no incluya <channel identifier>s.

Una <system reference area> sólo debe utilizarse como parte de una <sdl specification>.

Semántica

Una *Agent-definition* con el *Agent-kind* **SYSTEM** es la representación SDL de una especificación o descripción de un sistema. Un sistema es el bloque más externo. Ello significa que los agentes de un sistema son bloques y procesos que se interpretan de forma concurrente entre sí y con la posible máquina de estados del sistema.

Un sistema está separado de su entorno por la frontera del sistema y contiene a un conjunto de agentes. La comunicación entre el sistema y el entorno o entre agentes dentro del sistema puede efectuarse mediante señales, procedimientos remotos y variables remotas. Dentro de un sistema, estos medios de comunicación se transportan en canales explícitos o implícitos. Los canales conectan entre sí los agentes contenidos o con la frontera del sistema.

Un ejemplar de sistema es una ejemplificación de un tipo de sistema definido por una *Agent-definition* con *Agent-kind* **SYSTEM**. La interpretación de un ejemplar de sistema la efectúa una máquina SDL abstracta que, al hacerlo, da semántica a los conceptos SDL. Interpretar un ejemplar de sistema es:

a) iniciar el tiempo del sistema;

b) interpretar los agentes contenidos y sus canales conectados; y

c) interpretar la máquina de estados opcional del sistema.

Modelo

Para cada <channel definition> de un sistema que menciona **env**, a la definición de agente se añade una puerta con nombre anónimo. La definición de canal se modifica para mencionar esta puerta en el <channel path> dirigido al entorno del sistema.

9.2 Bloque

Un bloque es un agente con *Agent-kind* **BLOCK**. La gramática y la semántica de los agentes se aplica, por tanto, con las adiciones que proporciona esta cláusula. Un bloque se define por una <block definition> o un <block diagram>.

Los ejemplares contenidos en un ejemplar de bloque se interpretan de forma concurrente y asíncrona entre sí y con la máquina de estados del ejemplar de bloque contenedor. Toda la comunicación entre los diferentes ejemplares contenidos dentro de un bloque se efectúa de forma asíncrona mediante el intercambio de señales, ya sea explícita o implícitamente utilizando, por ejemplo, llamadas a procedimientos remotos.

Gramática textual concreta

<block definition> ::=
{<package use clause>}*
<block heading> <end> <agent structure>
endblock [[<qualifier>] <block name>] <end>

<block heading> ::=
block [<qualifier>] <block name> <agent instantiation>

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

Gramática gráfica concreta

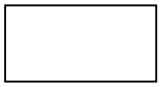
<block diagram> ::=

<frame symbol> ***contains*** {<block heading> <agent diagram content> }
is connected to { {<gate on diagram>}* <external channel identifiers> }***set***
[***is associated with*** <package use area>]

<block reference area> ::=

<block symbol> ***contains***
{ { <block name> [<number of instances>] } {<gate>*}***set*** }
is connected to { <gate property area>* }***set***

<block symbol> ::=



Las <gate>s se colocan cerca del borde de los símbolos y se asocian con los puntos de conexión a canales.

Los <external channel identifiers> identifican canales externo conectados a canales en el <block diagram>. Se coloca fuera del <frame symbol>, cerca del punto extremo de los canales en el <frame symbol>.

Semántica

Una definición de bloque es una definición de agente que define contenedores para uno o más procesos o definiciones de bloque.

Un ejemplar de bloque es una ejemplificación de un tipo de bloque definido por una *Agent-definition* con *Agent-kind* **BLOCK**. Interpretar un ejemplar de bloque es:

a) interpretar los agentes contenidos y sus canales conectados;

b) interpretar la máquina de estado opcional del bloque (si está presente).

En un bloque con una máquina de estados finita, la máquina de estados finita se crea como parte de la creación del bloque (y de los agentes que contiene), y se interpreta de forma concurrente con los agentes del bloque.

Un bloque con una definición de variable pero sin máquina de estados finita tiene una máquina de estados finita implícita asociada que se interpreta de forma concurrente con agentes en el bloque.

El acceso desde los agentes contenidos en el bloque a una variable del bloque se efectúa por dos procedimientos remotos definidos implícitamente para establecer y obtener los elementos de datos asociados a la variable. La máquina de estados del bloque proporciona dichos procedimientos.

Modelo

Un bloque b con una máquina de estados y con variables se modela manteniendo el bloque b (sin las variables) y transformando la entidad de estado y las variables en una máquina de estados diferenciada (sm) en el bloque b. Para cada variable v de b, esta máquina de estados tendrá una variable v y dos procedimientos exportados, set_v (con una parámetro de IN del género de v) y get_v (con un tipo de retorno del género de v). Cada asignación a v de las definiciones incluidas se transforma en una llamada remota de set_v. Cada ocurrencia de v en expresiones de definiciones incluidas se transforma en una llamada remota de get_v. Estas ocurrencias se aplican asimismo a las ocurrencias en procedimientos definidos en el bloque b, ya que éstas se transforman en procedimientos que son locales para los agentes llamantes.

Un bloque b que sólo tenga variables y/o procedimientos se transforma como se ha mencionado antes, teniendo el gráfico de la máquina de estados generada un único estado, para el que los procedimientos set y get constituyen entradas.

Los canales conectados a la máquina de estados se transforman de forma que se conectan a la máquina de estados (sm).

Esta transformación sucede después de que los tipos y parámetros de contexto se han transformado.

9.3 Proceso

Un proceso es un agente con *Agent-kind* **PROCESS**. La semántica de los agentes se aplica, por tanto, con las adiciones que se proporcionan en esta subcláusula. Un proceso se define por una <process definition> o un <process diagram>.

Un proceso se utiliza para introducir datos compartidos en una especificación, permitiendo que se utilicen variables del proceso contenedor o utilizando objetos. Todos los ejemplares de un proceso pueden acceder a las variables locales del proceso.

Para conseguir comunicaciones seguras, a pesar de la compartición de datos en un proceso, todos los ejemplares se interpretan utilizando una semántica alternativa. Ello implica que para cualesquiera dos ejemplares dentro de un proceso, no se interpretan dos transiciones en paralelo e, igualmente, que la interpretación de una transición en un ejemplar no es interrumpido por otro ejemplar. Cuando por ejemplo, un ejemplar espera la devolución de una llamada a procedimiento remoto, se encuentra en un estado y, por tanto, puede interpretarse un ejemplar alternativo.

Gramática abstracta

Una *Agent-definition* con la *Agent-kind* **PROCESS** tiene que, o bien contener al menos una *Agent-definition*, o bien tener una *State-machine-definition* explícita o implícita.

Las *Agent-definitions* contenidas de una *Agent-definition* con la *Agent-kind* **PROCESS** tendrá la *Agent-kind* **PROCESS**.

Gramática textual concreta

<process definition> ::=
{<package use clause>}*
<process heading> <end> <agent structure>
endprocess [[<qualifier>] <process name>] <end>

<process heading> ::=
process [<qualifier>] <process name> <agent instantiation>

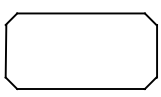
<process reference> ::=
process <process name> [<number of instances>] **referenced** <end>

Gramática gráfica concreta

<process diagram> ::=
<frame symbol> ***contains*** {<process heading> <agent diagram content> }
is connected to { {<gate on diagram>}* <external channel identifiers> }***set***
[***is associated with*** <package use area>]

<process reference area> ::=
<process symbol> ***contains***
{ { <process name> [<number of instances>] } {<gate>*}***set*** }
is connected to { {<gate property area>*}***set*** }

<process symbol> ::=



Las <puertas> se colocan cerca del borde de los símbolos y se asocian con el punto de conexión a canales.

<external channel identifiers> identifica canales externos conectados a canales en el <process diagram>. Se coloca fuera del <frame symbol>, cerca del punto extremo de los canales en el <frame symbol>.

Semántica

La interpretación del ejemplar de un proceso que contenga conjuntos de ejemplares de procesos se realiza interpretando los ejemplares de los conjuntos de ejemplares de procesos contenidos de forma alternativa, entre sí y con la máquina de estados del ejemplar de proceso contenedor, si lo hubiera. La interpretación alterna implica que, en cada instante, sólo uno de los ejemplares del contexto alternativo puede interpretar una transición y, asimismo, que una vez ha arrancado la interpretación de una transición de un ejemplar de proceso implicado, éste continúa hasta que se haya alcanzado un determinado estado (explícito o implícito) o se termine el ejemplar del proceso. El estado puede ser un estado implícito que ha sido introducido por transformaciones (por ejemplo, debido a una llamada a un procedimiento remoto).

Un proceso con definiciones de variables y procesos contenidos, pero sin una máquina de estados, tiene una máquina de estados implícita asociada que se interpreta alternativamente con los procesos contenidos.

NOTA – La agregación de estado también tiene una interpretación alternativa. No obstante, cada uno de los procesos que se alternan en un proceso tiene su propio puerto de entrada y su propio **self**, **parent**, **offspring** y **sender**. En caso de agregación de estado, sólo existe un puerto de entrada y un conjunto de **self**, **parent**, **offspring** y **sender** que pertenecen al agente contenedor.

9.4 Procedimiento

Se definen procedimientos por medio de definiciones de procedimiento. El procedimiento se invoca mediante una llamada de procedimiento que identifica la definición de procedimiento. Se asocian parámetros a la llamada de procedimiento. Mediante el mecanismo de transferencia de parámetros se controla qué variables son afectadas por la interpretación de un procedimiento. Las llamadas a procedimientos pueden ser acciones o expresiones (de procedimientos que retornan valor, exclusivamente).

Gramática abstracta

Procedure-definition	::	*Procedure-name* *Procedure-formal-parameter** [*Result*] *Procedure-identifier* *Data-type-definition-**set*** *Syntype-definition-**set*** *Variable-definition-**set*** *Composite-state-type-definition-**set*** *Procedure-definition-**set*** *Procedure-graph*
Procedure-name	=	*Name*
Procedure-formal-parameter	=	*In-parameter*
	|	*Inout-parameter*
	|	*Out-parameter*
In-parameter	::	*Parameter*
Inout-parameter	::	*Parameter*
Out-parameter	::	*Parameter*
Parameter	::	*Variable-name* *Sort-reference-identifier*
Result	::	*Sort-reference-identifier*
Procedure-graph	::	[*On-exception*] *Procedure-start-node* *State-node-**set*** *Free-action-**set*** *Exception-handler-node-**set***

Procedure-start-node :: [*On-exception*]
Transition
Procedure-identifier = *Identifier*

Si una *Procedure-definition* contiene *Result*, corresponde a un procedimiento que retorna valor.

Gramática textual concreta

```
<procedure definition> ::=
                        <external procedure definition>
                |       {<package use clause>}*
                        <procedure heading> <end>
                            <entity in procedure>*
                            [<procedure body>]
                        endprocedure [ [<qualifier>] <procedure name> ] <end>
                |       {<package use clause>}*
                        <procedure heading>
                            [ <end> <entity in procedure>+ ]
                            [<virtuality>] <left curly bracket>
                                    <statement list>
                            <right curly bracket>

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

<procedure heading> ::=
                        <procedure preamble>
                        procedure [<qualifier>] <procedure name>
                        [<formal context parameters>] [<virtuality constraint>]
                        [<specialization>]
                        [<procedure formal parameters>]
                        [<procedure result>] [<raises>]

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

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

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

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

<raises> ::=
                        raise <exception identifier> {, <exception identifier>}*

<entity in procedure> ::=
                        <variable definition>
                |       <data definition>
                |       <data type reference>
                |       <procedure reference>
                |       <procedure definition>
                |       <composite state type definition>
                |       <composite state type reference>
                |       <exception definition>
                |       <select definition>
                |       <macro definition>

<procedure signature> ::=
                        [ (<formal parameter> { , <formal parameter> }* ) ] [<result>] [<raises>]

<procedure body> ::=
                        [<on exception>] [<start>] { <state> | <exception handler> | <free action> }*

<external procedure definition> ::=
                        procedure <procedure name> <procedure signature> external <end>
```

<procedure reference> ::=
<type preamble>
procedure <procedure identifier> <type reference properties>

La <virtuality> opcional antes de <left curly bracket> <statement list> en <procedure definition> se aplica en la transición de arranque del procedimiento, que en este caso es la lista de enunciados.

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

<variable definition> en una <procedure definition> no puede contener **exported**, <variable name>s (véase 12.3.1).

Si está presente, **exported** es heredado por cualquier subtipo de un procedimiento. Un procedimiento virtual exportado debe contener **exported** en todas las redefiniciones. Los tipos virtuales incluidos los procedimientos virtuales se describen en 8.3.2. La cláusula optativa **as** en una redefinición debe denotar el mismo <remote procedure identifier> que en el supertipo. Si se omite en una redefinición, el <remote procedure identifier> del supertipo es implícito.

Dos procedimientos exportados en un agente no pueden mencionar el mismo <remote procedure identifier>.

Un procedimiento externo no puede mencionarse en una <type expression>, en un <formal context parameter> ni en una <procedure constraint>.

No existe una sintaxis gráfica correspondiente para <external procedure definition>.

Si una excepción puede generarse (*raise*) en un procedimiento cuando no está activo ningún manejador de excepciones con la correspondiente cláusula de manejador (es decir, no es manejada), el <raises> debe mencionar esa excepción. Se considera que en un procedimiento no puede manejarse una excepción si existe un potencial flujo de control dentro del procedimiento que produce dicha excepción y ninguno de los manejadores de excepción activados en dicho control de flujo manejan la excepción.

Si una <procedure definition>aparece dentro de un <text symbol>, no podrá contener <state>.

Un **endprocedure** dentro de una <procedure definition> interior en una <entity in procedure> de una <procedure definition> exterior pertenece a la <procedure definition> interior.

Gramática gráfica concreta

<procedure diagram> ::=
<frame symbol> ***contains*** {
<procedure heading>
{ <procedure text area>*
<procedure area>*
<procedure graph area> }***set*** }
[***is associated with*** <package use area>]

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

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

<procedure reference area> ::=
<type reference area>

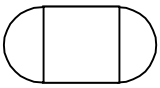
La <type reference area> que forma parte de una <procedure reference area> debe tener un <graphical type reference heading> que contenga un <procedure name>.

<procedure symbol> ::=

<procedure graph area> ::=
[<on exception association area>] [<procedure start area>]
{<state area> | <exception handler area> | <in connector area> }*

<procedure start area> ::=
<procedure start symbol>
contains { [<virtuality>] }
[***is connected to*** <on exception association area>]
is followed by <transition area>

<procedure start symbol> ::=



La <package use area> debe ubicarse encima del <frame symbol>.

La <on exception association area> de una <procedure graph area> identifica al manejador de excepciones asociado a todo el gráfico. El extremo origen no debe estar conectado a ningún símbolo.

Semántica

Un procedimiento es un medio de dar un nombre a un grupo de ítems y representar este grupo por una sola referencia. Las reglas para los procedimientos prescriben la forma de elegir el grupo de ítems, 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 10.5.

Una variable de procedimiento es una variable local dentro del procedimiento, que no puede ser exportada. 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 un *Call-node* (representado por una <procedure call>, véase 11.13.3), de un *Value-returning-call-node* (representado por una <value returning procedure call>, véase 12.3.5), o de una *Operation-application* (representado por una <operation application>, véase 12.2.7) causa la creación de un ejemplar de procedimiento y la interpretación comienza de la forma siguiente:

a) Para cada *In-parameter* se crea una variable local que tiene el *Name* y el *Sort* del *In-parameter*. La variable se asocia con el resultado de la expresión interpretando una asignación entre la variable y la expresión dada por el correspondiente parámetro real, si está presente. Si no es así, la variable no recibe ningún ítem de datos asociado, es decir que se vuelve "indefinida".

b) Para cada *Out-parameter* que tenga el *Name* y el *Sort* de la *Out-parameter* se crea una variable local. La variable no tiene ítem de datos, es decir, se vuelve "indefinida".

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

d) Cada *Inout-parameter* denota una variable que se da por la expresión del parámetro efectivo en 11.13.3. El *Variable name* contenido se utiliza durante toda la interpretación del *Procedure-graph* cuando se hace referencia al ítem de datos asociado a la variable o cuando se asigna un nuevo valor a la variable.

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

f) Antes de la interpretación de un *Return-node* contenido en el *Procedure-graph*, los *Out-parameters* reciben los ítems de datos de la correspondiente variable local.

Los nodos del gráfico de procedimiento se interpretan de la misma manera que los nodos equivalentes de un agente; es decir, el procedimiento tiene el mismo conjunto completo de señales de entrada válidas que el agente circundante, y el mismo puerto de entrada que el ejemplar del agente circundante que lo ha llamado, ya sea directa o indirectamente.

Un procedimiento externo es un procedimiento cuyo <procedure body> no está incluido en la descripción SDL. Conceptualmente, se supone que a un procedimiento externo se le da un <procedure body> y que se transformará en una <procedure definition> como parte de la transformación de sistema genérica (véase el anexo F).

Modelo

Un parámetro formal sin <parameter kind> explícito tiene el <parameter kind> implícito **in**.

Si un <variable name> está presente en <procedure result>, entonces todos los <return>s o <return area>s dentro del gráfico de procedimiento que no tengan una <expression> se sustituyen por un <return> o <return area> respectivamente, conteniendo <variable name> como la <expression>.

Un <procedure result> con <variable name> es una sintaxis derivada para una <variable definition> con <variable name> y <sort> en <variables of sort>. Si existe una <variable definition> que incluye un <variable name> no se añaden ninguna <variable definition> adicional.

Una <procedure start area> que contiene <virtuality>, un procedimiento <start> que contiene <virtuality>, o una <statement list> en una <procedure definition> que sigue a <virtuality> se denomina un arranque de procedimiento virtual. El arranque de procedimiento virtual se describe más detalladamente en 8.3.3.

La segunda forma de <procedure definition> es sintaxis derivada para la siguiente <procedure definition>:

<procedure preamble>
<procedure heading>**;**
  start <virtuality>**;**
  task { <statement list>***-transform*** **};**
  return ;
endprocedure ;

Esta transformación tiene lugar después de la transformación de <compound statement>.

10 Comunicación

10.1 Canal

Gramática abstracta

Channel-definition	::	*Channel-name* [**NODELAY**] *Channel-path-**set***
Channel-path	::	*Originating-gate* *Destination-gate* *Signal-identifier-**set***
Originating-gate	=	*Gate-identifier*
Destination-gate	=	*Gate-identifier*
Gate-identifier	=	*Identifier*
Agent-identifier	=	*Identifier*
Channel-name	=	*Name*

El *Channel-path-**set*** contiene al menos un *Channel-path* y no más de dos. Cuando hay dos trayectos, el canal es bidireccional y la *Originating-gate* de cada *Channel-path* tiene que ser la misma que la *Destination-gate* del otro *Channel-path*.

Si la *Originating-gate* y la *Destination-gate* son el mismo agente, el canal debe ser unidireccional (tiene que haber un solo elemento en el *Channel-path-**set***).

La *Originating-gate* o *Destination-gate* deben estar definidas en la misma unidad de ámbito de la sintaxis abstracta en la que se define el canal.

NODELAY denota que el canal no tiene retardo.

Se permite que un canal conecte los dos sentidos de una puerta bidireccional.

Gramática textual concreta

<channel definition> ::=
channel [<channel name>] [**nodelay**]
<channel path> [<channel path>]
endchannel [<channel name>] <end>

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

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

<via gate> ::=
via <gate>

El <channel name> de finalización sólo se especifica si, a su vez, se especifica el <channel name> de arranque. Si el <channel name> de arranque no se especifica, no puede hacerse referencia al canal por un nombre.

El <channel endpoint> **this** denota la máquina de estados del agente que circunda directamente a la definición del canal.

<gate> sólo debe especificarse si:

a) <channel endpoint> denota una conexión a una <textual typebased agent definition> o a una <textual typebased state partition definition>, en cuyo caso, la <gate> debe estar definida directamente en el tipo de agente o en el tipo de agente o tipo de estado para ese agente o estado respectivamente; o

b) **env** está especificado y el canal está definido en un tipo de agente, en cuyo caso <gate> debe estar definida en ese tipo de agente respectivamente.

Si <gate> está especificada, el canal está conectado a esa puerta. La puerta y el canal deben tener al menos un elemento común en sus listas de señales en el mismo sentido. Si no se especifica <gate> alguna, se aplican las reglas siguientes:

a) si el punto extremo de un canal es un agente o máquina de estados y ese agente/estado contiene una <channel to channel connection> para el canal, el canal está conectado a la puerta implícita que introduce la <channel to channel connection>;

b) si el punto extremo de un canal es un estado, el canal está conectado a la puerta implícita de esa máquina de estados (véase el anexo F),

en cualquier otro caso, el canal introduce una puerta implícita en el agente o servicio mencionado en <channel endpoint>. Esta puerta obtiene la <signal list> de los <channel path>es respectivos como sus correspondientes limitaciones de puerta. El canal está conectado a esa puerta.

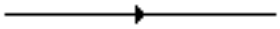
Gramática gráfica concreta

<channel definition area> ::=
<channel symbol>
is associated with
{ [<channel name>] { [<gate>] [<signal list area>] [<signal list area>] }***set*** }
is connected to {
{ <agent area> | <state machine area>
| <gate property area> | <gate on diagram> }
{ <agent area> | <state machine area>
| <gate property area> | <gate on diagram> } }***set***

Cuando un <channel symbol> está conectado a una <state machine area>, denota la máquina de estados del agente que circunda directamente la definición del canal.

<channel symbol> ::=
<delaying channel symbol 1>
| <delaying channel symbol 2>
| <nondelaying channel symbol 1>
| <nondelaying channel symbol 2>

<delaying channel symbol 1> ::=



<delaying channel symbol 2> ::=

<nondelaying channel symbol 1> ::=

<nondelaying channel symbol 2> ::=

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

Las puntas de flecha para <nondelaying channel symbol 1> y <nondelaying channel symbol 2> están situadas en los extremos del canal e indican que dicho canal no tiene retardo.

Semántica

Una *Channel-definition* representa un trayecto de transporte para señales (incluidas las señales implicadas por procedimientos remotos y variables remotas, véanse 10.5 y 10.6). Un canal puede considerarse como uno o dos trayectos de canal unidireccionales e independientes entre dos agentes, o entre un agente y su entorno. Un canal también puede estar conectado a la máquina de estados (estado compuesto) de un agente con el entorno y con agentes contenidos.

El *Signal-identifier*-**set** de 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 su origen. Si dos o más señales son presentadas simultáneamente al canal, se ordenan de forma arbitraria.

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, el primer ejemplar de señal de la cola es liberado y se aplica a uno de los puntos extremos que está conectado al canal.

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

Cuando se envía un ejemplar de señal a un ejemplar del mismo conjunto de ejemplares de agente, la interpretación del *Output-node* implica que la señal se aplica directamente al puerto de entrada del agente de destino o que la señal se envía a través de un canal sin retardo que conecta el conjunto de ejemplares del agente con él mismo.

Un procedimiento remoto o una variable remota en un canal se menciona como salida desde un importador y como entrada hacia un exportador.

Modelo

Si se omite el <channel name> en una <channel definition> o <channel definition area>, el canal recibe un nombre de forma implícita e inequívoca.

Si un agente o un tipo de agente contiene puertas explícitas o implícitas que no están conectadas por medio de canales explícitos, se derivan canales implícitos sin listas de señales. A partir de entonces, los canales sin listas de señales explícitas se llenan con señales, procedimientos remotos y variables remotas. Los detalles se describen en el anexo F.

Los <channel path>s definidos explícitamente deben tener, después de estas transformaciones, listas de señales no vacías.

Un canal en el que los dos puntos extremos son puertas de una <textual typebased agent definition> representa canales individuales de cada uno de los agentes de dicho conjunto con todos los agentes del mismo, incluyendo el agente de origen. Cualquier canal bidireccional resultante que conecte un agente del conjunto con el propio agente se divide en dos canales unidireccionales.

10.2 Conexión

Gramática textual concreta

<channel to channel connection> ::=
connect <external channel identifiers>
and <channel identifiers> <end>

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

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

Después de la palabra clave **and** no puede mencionarse canal alguno en más de una <channel to channel connection> de una unidad de ámbito dada.

Para cualquier pareja de <channel to channel connection>s de una unidad de ámbito, los <external channel identifiers>s mencionarán el mismo conjunto de canales o bien, no tendrán canales en común. Si dos o más <channel to channel connection>s de una unidad de ámbito dada tienen el mismo conjunto de canales externos, ello es sintaxis derivada de una única <channel to channel connection> para la que uno de los <external channel identifiers> es su <external channel identifiers>, y sus <channels identifiers> están constituidos por la lista de todos los <channel identifier>s internos.

Gramática gráfica concreta

Gráficamente, el constructivo connect está representado por la conexión gráfica de un <channel symbol> en una <channel definition area> a <external channel identifiers> en una <gate on diagram>.

En la conexión de un diagrama no se mencionará ningún <channel identifier> más de una vez.

NOTA – Debido al constructivo **connect**, un canal (externo) que puede ser anónimo en la versión gráfica de una especificación puede necesitar tener un nombre en la correspondiente versión textual. Esto es completamente análogo al caso de <merge area>s en gráficos de procesos o procedimientos. Una herramienta

que convierta la versión gráfica de una especificación en una versión textual, debe ser capaz de generar nombres de canales implícitos.

Semántica

Los <channel identifier>s de un <external channel identifiers> que pertenezca a una <channel to channel connection> deben denotar canales conectados al agente o servicio circundante. Cada canal conectado al agente circundante debe mencionarse en el <external channel identifiers> que forme parte de al menos una <channel to channel connection>.

Cada canal identificado por un <channel identifier> en un <channel identifiers> que forme parte de una <channel to channel connection> debe definirse en el mismo agente en el que se define la <channel to channel connection> y debe tener en la frontera de dicho agente uno de sus puntos extremos. Cada canal definido en el agente circundante y cuyo entorno sea uno de sus puntos extremos, debe ser mencionado en el <channel identifiers> que forme parte de una <channel to channel connection>.

Modelo

Las conexiones son constructivos taquigráficos y se transforman en puertas.

Cada una de las <channel to channel connection> de una unidad de ámbito dada define una puerta implícita en la unidad de ámbito. Todos los canales de la <channel to channel connection> están conectados a dicha puerta en sus respectivas unidades de ámbito. Las limitaciones de puerta de la puerta implícita se derivan de los canales conectados a la puerta.

El nombre de la puerta es un nombre derivado singular e inequívoco. En la unidad de ámbito circundante, la <channel definition> que identifica el <channel identifier> se amplía con una parte <via gate>. La parte <via gate> se añade al <channel endpoint> que hace referencia a la unidad de ámbito actual y menciona la puerta implícita. Dentro de la unidad de ámbito, los canales asociados con el canal externo mediante la <channel to channel connection> se modifican ampliando el <channel endpoint> que menciona **env** con una parte <via gate> para la puerta implícita.

10.3 Señal

Gramática abstracta

Signal-definition	::	*Signal-name* *Sort-reference-identifier**
Signal-identifier	=	*Identifier*
Signal-name	=	*Name*

Gramática textual concreta

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

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

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

<signal reference> ::=
<type preamble>
signal <signal identifier> <type reference properties>

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

Una señal abstracta sólo puede ser utilizada en especialización y en limitaciones de señal.

Gramática gráfica concreta

<signal reference area> ::=
<type reference area>

La <type reference area> que forma parte de una <signal reference area> debe tener un <graphical type reference heading> que contenga un <signal name>.

Semántica

Un ejemplar de señal es un flujo de información entre agentes y es una ejemplificación de un tipo de señal definido por una definición de señal. Un ejemplar de señal puede ser enviado por el entorno o por un agente y siempre está dirigido a un agente o al entorno. Un ejemplar de señal se crea cuando se interpreta un *Output-node* y deja de existir cuando se interpreta un *Input-node*.

La semántica de <virtuality> se define en 8.3.2.

10.4 Definición de lista de señales

Un <signal list identifier> puede utilizarse en <signal list> como una notación taquigráfica para una lista de identificadores de señales, procedimientos remotos, señales de temporizador e interfaces.

Gramática textual concreta

<signal list definition> ::=
signallist <signal list name> <equals sign> <signal list> <end>

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

<signal list item> ::=
<signal identifier>
| **(** <signal list identifier> **)**
| <timer identifier>
| [**procedure**] <remote procedure identifier>
| [**interface**] <interface identifier>
| [**remote**] <remote variable identifier>

La <signal list> que se construye sustituyendo todos los <signal list identifier>s de la lista por la lista de los <signal identifier>s o <timer identifier>s que éstos denotan y sustituyendo todos los <remote procedure identifier>s y todos los <remote variable identifier>s por una de las señales implícitas que cada uno de ellos denota (véanse 10.5 y 10.6), corresponde a un *Signal-identifier*-**set** en la *Gramática abstracta*.

Un <signal list item> que es un <identifier> denota un <signal identifier> o <timer identifier> si lo permiten las reglas de visibilidad, o bien, un <remote procedure identifier> si lo permiten las reglas de visibilidad, o bien, un <remote variable identifier>. Para que un <signal list item> denote obligatoriamente a un <remote procedure identifier>, <interface identifier> o <remote variable identifier> puede utilizarse la palabra clave **procedure**, **interface** o **remote** respectivamente.

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>).

Sólo está permitido utilizar **this** en <signal list>s que sean parte de las <gate constraint>s (véase 8.1.6).

Gramática gráfica concreta

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

<signal list symbol> ::=

10.5 Procedimientos remotos

Un agente cliente puede llamar a un procedimiento definido en otro agente, mediante una petición al agente servidor a través de una llamada a procedimiento remoto de un procedimiento en el agente servidor.

Gramática textual concreta

<remote procedure definition> ::=
remote procedure <remote procedure name> [**nodelay**]
<procedure signature> <end>

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

<remote procedure call body> ::=
<remote procedure identifier> [<actual parameters>]
<communication constraints>

<communication constraints> ::=
{**to** <destination> | **timer** <timer identifier> | <via path>}*****

En 10.1 se describe la utilización de **nodelay**.

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**, está implícito el nombre del procedimiento exportado así como la <remote procedure definition> del ámbito más próximo circundante con el mismo nombre.

Un procedimiento remoto mencionado en una <remote procedure call> debe estar en el conjunto de salida completo (véase el anexo F) de un tipo de agente o conjunto de agentes circundante.

Si <destination> de un <remote procedure call body> es <pid expression> con un género distinto a pid (véase 12.1.6), el <remote procedure identifier> debe representar un procedimiento remoto contenido en la interfaz que define el género pid.

Cuando se omiten el <destination> y el <via path>, existe una ambigüedad sintáctica entre <remote procedure call body> y <procedure call>. En este caso, el <identifier> contenido denota un <procedure identifier> si las reglas de visibilidad lo permiten y, en cualquier otro caso, un <remote procedure identifier>.

El <timer identifier> de <communication constraints> no podrá tener el mismo <identifier> que un <exception identifier>.

En un <remote procedure call body> una lista de <communication constraints> se asocia al último <remote procedure identifier>. Por ejemplo, en

call p **to call** q **timer** t **via** g

el **timer** t y la **gate** g se aplica a la **call** de q.

Un <communication constraints> contendrá no más de un <destination> y no más de un <timer identifier>.

Gramática gráfica concreta

<remote procedure call area> ::=
<procedure call symbol> ***contains*** <remote procedure call body>
[***is connected to*** <on exception association area>]

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 un procedimiento importado y un procedimiento exportado se establece haciendo ambos referencia a la misma <remote procedure definition>.

Una llamada a procedimiento remoto por un agente solicitante hace que dicho agente solicitante espere hasta que el agente servidor haya interpretado el procedimiento. Se conservan las señales enviadas al agente solicitante mientras espera. El agente servidor interpretará 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 para un estado no se especifica <save part> ni <input part>, se añade una transición implícita que consiste en la llamada a procedimiento únicamente y que conduce al mismo estado. Si para un estado está especificada una <input part>, se añade una transición implícita que consiste en la llamada a procedimiento seguida de <transition>. Si para un estado dado se especifica <save part>, se añade una conservación implícita de la señal para el procedimiento solicitado.

Modelo

Una llamada a procedimiento remoto

call Proc(apar) **to** destination **timer** timerlist **via** viapath

se modela mediante un intercambio de señales definidas implícitamente. Si las cláusulas **to** o **via** se omiten de la llamada a procedimiento remoto, también se omiten en las transformaciones siguientes. Los canales son explícitos si se ha mencionado el procedimiento remoto en la <signal list> (la salida para el importador y la entrada para el exportador) de, al menos, una puerta o canal conectado al importador o exportador. Cuando un procedimiento remoto se transporta en canales explícitos, se ignora la palabra clave **nodelay** de la <remote procedure definition>. El agente solicitante envía al agente servidor una señal que contiene los parámetros actuales de la llamada a procedimiento, excepto los parámetros reales que corresponden a parámetros **out**, y espera una respuesta. En respuesta a dicha señal, el agente servidor interpreta el correspondiente procedimiento remoto, devuelve una señal al agente solicitante con los resultados de todos los parámetros **in**/**out** y parámetros **out**, e interpreta la transición.

Existen dos <signal definition>s implícitas para cada <remote procedure definition>s en una <system definition>. Los <signal name>s en dichas <signal definition>s se denotan mediante pCALL y pREPLY respectivamente, donde p se determina de forma inequívoca. Las señales se definen en la misma unidad de ámbito que la <remote procedure definition>. pCALL y pREPLY tienen un primer parámetro de género Integer predefinido.

En cada canal que menciona el procedimiento remoto, éste es sustituido por *pCALL*. Para cada uno de dichos canales, se añade un nuevo canal en sentido opuesto; este canal transporta la señal *pREPLY*. El nuevo canal tiene la misma propiedad de retardo que el original.

a) Para cada procedimiento importado, se definen dos nuevas variables Integer implícitas, n y newn, donde n se inicializa a 0.

NOTA 1 – El parámetro n se introduce para reconocer y descartar señales de respuesta de llamadas a procedimientos remotos que tuvieron entrada a través de la expiración del temporizador asociado.

La <remote procedure call> se transforma tal como se indica a continuación.

```
task n:= n + 1;
output pCALL(apar,n) to destination via viapath;
wait in state pWAIT, saving all other signals;
input pREPLY(aINOUTpar,newn);
decision newn = n;
(true):
(false): nextstate pWAIT;
enddecision;
return;
```

donde apar es la lista de parámetros reales, a excepción de los parámetros reales correspondientes a parámetros out, y aINOUTpar es la lista modificada de parámetros in/out y out reales, incluido un parámetro adicional en caso de que se transforme un valor que devuelve una llamada a procedimiento remoto.

NOTA 2 – El enunciado retorno termina el procedimiento implícito introducido conforme a 11.12.1.

Para cada excepción contenida en el <raises> de un procedimiento remoto p y todas las excepciones predefinidas e, se define una señal eRAISE que puede transportar todos los parámetros de excepción de e. Lo siguiente se inserta en el estado pWAIT:

```
state pWAIT;
input eRAISE(params,newn);
decision newn = n;
(true): raise e(params);
(false): nextstate pWAIT;
enddecision;
```

Para un temporizador t incluido en <communication constraints> se inserta implícitamente una excepción adicional con el mismo nombre y los mismos parámetros en el mismo ámbito que la definición del temporizador, no debiendo existir una excepción definida explícitamente con el mismo nombre que el temporizador en la misma unidad de ámbito en la que se define el temporizador.

Adicionalmente, se inserta lo siguiente para un temporizador t incluido en <communication constraints>

```
state pWait;
input t(aParams);
raise t(aParams);
```

donde aParams significa variables implícitamente definidas con el género de los parámetros contenido en la definición del temporizador.

En todos los estados del agente excepto pWAIT se inserta

```
input pReply, eRAISE;
nextstate actual state;
```

b) En el agente servidor se definen un manejador de excepciones implícitas pEXC y una variable entera implícita n para cada <input part> explícita o implícita que sea una entrada de procedimiento remoto. Además, existe una variable ivar para cada una de dichas <input part> definidas en el ámbito en el que aparece la entrada de procedimiento explícito o implícito. Si se transforma un valor que devuelve una llamada a procedimiento remoto, se define una variable implícita res con el mismo género que <sort> en <procedure result>.

Para todos los <state>s con una transición de entrada de procedimiento remoto, se añade la siguiente <input part>:

```
input pCALL(fpar,n);
task ivar:= sender;
call Proc(fpar); onexception pEXC;
output pREPLY(INOUTpar,n) to ivar;
transition;
```

o bien,

```
input pCALL(fpar,n);
task ivar:= sender;
task res := call Proc(fpar); onexception pEXC;
output pREPLY(INOUTpar,res,n) to ivar;
transition;
```

si se ha transformado un valor que devuelve una llamada a procedimiento remoto.

Para todos los <state>s con conservación de procedimiento remoto, se añade la siguiente <save part>:

```
save pCall;
```

Para todos los <state>s con un <remote procedure reject>, se añade la siguiente <input part>:

```
input pCALL;
output eRAISE(params,n) to sender;
transition;
```

Para todos los restantes <state>s excluidos los estados implícitos derivados de entradas, se añaden las siguientes <input part>:

```
input pCALL(fpar,n);
task ivar:= sender;
call Proc(fpar); onexception pEXC;
output pREPLY(INOUTpar,n) to ivar;
/* next state the same */
```

Para cada excepción e contenida en el <raises> del procedimiento remoto y para cada excepción predefinida se inserta lo siguiente:

```
exceptionhandler pEXC;
handle e(params);
output eRAISE(params,n) to ivar;
raise e(params);
```

Si se ha asociado un manejador de excepciones a una entrada de procedimiento remoto, el manejador de excepciones se asocia a la entrada de la señal resultante (no se muestra en el modelo anterior).

NOTA 3 – Existe la posibilidad de un bloqueo empleando el constructivo de procedimiento remoto, especialmente si no se indica <destination> o si <destination> no denota una <pid expression> de un agente cuya existencia está garantizada por la especificación en el momento de recibirse la señal pCALL. Los temporizadores asociados permiten evitar el bloqueo.

10.6 Variables remotas

En SDL, una variable siempre pertenece a un ejemplar de agente, para la cual es local. Normalmente, la variable sólo es visible para el ejemplar del agente que la posee y para los agentes contenidos. Si un ejemplar de agente en otro agente necesita acceder a los ítems de datos asociados a una variable, es necesario un intercambio de señales con el ejemplar del agente que posee la variable.

Esto puede conseguirse mediante la siguiente notación taquigráfica, denominada variables importadas y exportadas. La notación taquigráfica puede utilizarse también para exportar ítems de datos a otras instancias de agente dentro del mismo agente.

Gramática textual concreta

<remote variable definition> ::=

remote <remote variable name> {**,**<remote variable name>}* <sort> [**nodelay**]
{**,** <remote variable name> {**,** <remote variable name>}* <sort> [**nodelay**]}*
<end>

<import expression> ::=

import (<remote variable identifier> <communication constraints> **)**

<export> ::=

export (<variable identifier> { **,** <variable identifier> }* **)**

En el anexo F se describe la utilización de **nodelay**.

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.

Una variable remota mencionada en una <import expression> debe estar en el conjunto completo de salidas (véase el anexo F) de un tipo de agente o conjunto de agente circundante.

El <variable identifier> en <export> debe denotar una variable definida con **exported**.

Si <destination> en una <import expression> es <pid expression> con un género distinto de pid (véase 12.1.6), el <remote variable identifier> debe representar una variable remota contenida en la interfaz que define el género pid.

No hay una sintaxis gráfica correspondiente para <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 variable importada y una variable exportada se establece refiriéndolas a la misma <remote variable definition>.

Las variables importadas se especifican como parte del conjunto de salida de la entidad activa circundante. Las variables exportadas se especifican como parte del conjunto completo de entrada de la entidad activa circundante.

El ejemplar de agente que posee una variable cuyos ítems de datos se exportan a otros ejemplares de agentes se denomina el exportador de la variable. Otros ejemplares de agente que utilizan dichos ítems de datos se denominan importadores de la variable. La variable se denomina variable exportada.

Un ejemplar de agente puede ser exportador e importador de la misma variable remota.

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 interpretación de <export> por la cual un exportador revela el resultado vigente de una variable exportada. Una operación de exportación causa el almacenamiento del resultado vigente de la variable exportada en su copia implícita.

b) *Operación de importación*

Una operación de importación es la interpretación de una <import expression> por la cual un importador accede al resultado de una variable exportada. El resultado se almacena en una variable implícita denotada por el <remote variable identifier> en la <import expression>. El exportador que contiene la variable exportada es especificado por <destination> en <import expression>. Si no se especifica ningún <destination>, la importación procede de un ejemplar de agente arbitrario 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 la variable de exportación y en la <import expression>.

Modelo

Una operación de importación se modela mediante el intercambio de señales definidas implícitamente. Cuando se transporta una variable remota por canales explícitos, se ignora la palabra clave **nodelay** de la <remote variable definition>. 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 resultado 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 se define, la se inicializa también la copia implícita con el mismo resultado que la variable de exportación.

Hay dos <signal definition>s implícitas para cada <remote variable definition>s 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 señal xQUERY tiene un argumento del género predefinido Integer y xREPLY tiene argumentos del género de la variable y de Integer. La copia implícita de la variable exportada se denota por imcx.

En cada canal que menciona la variable remota, ésta es sustituida por xQUERY. Para cada uno de dichos canales, se añade un nuevo canal en sentido opuesto; este canal transporta la señal xREPLY. En el caso de un canal, el nuevo canal tiene la misma característica de retardo que el original.

Para cada excepción predefinida (denotada como predefExc), se define una señal anónima adicional (denotada como predefExcRAISE).

a) *Importador*

Para cada variable importada, se definen dos variables enteras implícitas n y newn, siendo n inicializada con el valor 0. Además, se define una variable implícita cuyo género es el de la variable remota.

La <import expression>

import (x **to** destination **via** via-path)

se transforma en lo siguiente, donde la cláusula **to** se omite si no existe el destino y la cláusula **via** se omite si no existe en la expresión original:

```
task n:= n + 1;
output xQUERY(n) to destination via via-path;
wait in state xWAIT, saving all other signals;
input xREPLY(x,newn);
decision newn = n;
(true):
(false): nextstate xWAIT;
enddecision;
return;
state xWAIT
input predefExcRAISE;
raise predefExc;
```

En todos los demás estados, se conserva xREPLY.

NOTA 1 – El enunciado retorno termina el procedimiento implícito introducido conforme a 11.12.1.

Para cada temporizador t incluido en <communication constraints> se inserta implícitamente en el mismo ámbito que la definición del temporizador una excepción adicional con el mismo nombre y los mismos parámetros. En ese caso no debe haber una excepción con el mismo nombre en la unidad de ámbito de la definición de temporizador.

Además, para cada temporizador t que se incluye en <communication constraints> se inserta lo siguiente:

state xWait;
input t(aParams);
raise t(aParams);

donde aParams representa variables definidas implícitamente con el género de los parámetros contenidos en la definición del temporizador.

El resultado de la transformación se encapsula en un procedimiento implícito, descrito en 11.12.1. Cada <on exception> asociado a la acción de importación se adjunta a una llamada del procedimiento implícito.

b) *Exportador*

A todas los <state>s del exportador, excluidos los estados implícitos derivados de la importación, se añade la <input part> siguiente:

input xQUERY(n);
task ivar := sender;
output xREPLY(imcx,n) **to** ivar; **onexception** xEXC;
nextstate *the state containing this input*;
exceptionhandler xEXC;
handle predefExc;
output predefExcRAISE **to** ivar;
raise predefExc;

Para cada uno de dichos estados, ivar se define como variable de género pid, y n como una variable de tipo Integer.

La <export>

export x

se transforma en lo siguiente:

task imcx := x;

NOTA 2 – 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 agente cuya existencia está garantizada por la especificación en el momento de recibir la señal xQUERY. La especificación de un temporizador de conjunto en la <import expression> evita dicho bloqueo.

11 Comportamiento

11.1 Arranque

Gramática abstracta

State-start-node :: [*On-exception*]
[*State-entry-point-name*]
Transition

Gramática textual concreta

<start> ::=
start [<virtuality>] [<state entry point name>] <end> [<on exception>] <transition>

Si en un <start> existe un <state entry point name>, el <start> debe serlo de un <composite state>.

Gramática gráfica concreta

<start area> ::=

<start symbol> ***contains*** { [<virtuality>] [<state entry point name>]}
[***is connected to*** <on exception association area>]
is followed by <transition area>

<start symbol> ::=

Si en un <start> existe un <state entry point name>, la <start area> debe serlo de un <composite state>.

Semántica

Se interpreta la *Transition* del *State-start-node*.

Modelo

Un <start> o <start area> que contenga <virtuality> se denomina arranque virtual. El arranque virtual se describe en 8.3.3.

Un <start> o <start area> contenida respectivamente en un <composite state> o <composite state diagram> se define en 11.11.

11.2 Estado

Gramática abstracta

State-node	::	*State-name* [*On-exception*] *Save-signalset* *Input-node-**set*** *Spontaneous-transition-**set*** *Continuous-signal-**set*** [*Composite-state-type-identifier*]
State-name	=	*Name*

*State-node*s dentro de un *State-transition-graph* o *Procedure-graph* deben tener distintos *State-name*s.

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

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

Un *State-node* con *Composite-state-type-identifier* representa una <composite state application>.

Gramática textual concreta

<state> ::=

<basic state> | <composite state application>

Una <composite state application> representa un estado con un *Composite-state-type-identifier*.

Gramática gráfica concreta

<state area> ::=

<basic state area> | <composite state application area>

Una <composite state area> representa un estado con un *Composite-state-type-identifier*.

11.2.1 Estado básico

Gramática textual concreta

<basic state> ::=

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

<state list> ::=

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

<asterisk state list> ::=

<asterisk> [**(** <state name> { **,** <state name>}* **)**]

Un estado puede tener asociado como máximo un manejador de excepciones.

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 eventuales señales de entrada implícitas. Para cada *State-node*, una *Spontaneous-transition* está representada por una <spontaneous transition>.

Un <state name> puede aparecer en más de un <state> de un cuerpo.

El <state name>s en una <asterisk state list> debe ser distinto y debe estar contenido en otro <state list>s en el cuerpo circundante o en el cuerpo de un supertipo.

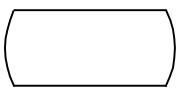
El <state name> opcional que termina un <state> sólo puede especificarse si la <state list> en el <state> consiste en un solo <state name>, en cuyo caso debe ser ese mismo <state name>.

Gramática gráfica concreta

<basic state area> ::=

<state symbol> ***contains*** <state list>
[***is connected to*** <on exception association area>]
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>

Una <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.

Semántica

Un estado representa una condición particular en la que la máquina de estados de un agente puede consumir un ejemplar de señal. Si se consume un ejemplar de señal, se interpreta la transición asociada. Una transición puede también interpretarse como el resultado de una señal continua o de una transición espontánea.

Para cada estado, los *Save-signal*s, *Input-node*s, *Spontaneous-signal*s, y *Continuous-signal*s se interpretan en el orden siguiente:

a) si el puerto de entrada contiene una señal que concuerda con una entrada de prioridad del estado actual, se consume la primera de dichas señales (véase 11.4); en cualquier otro caso

b) en el orden de las señales en el puerto de entrada:

   1) las *Provided-expression*s del *Input-node* correspondientes a la señal actual se interpretan en orden arbitrario, si es que existen;

   2) si la señal actual está habilitada, se consume esa señal (véase 11.6); en cualquier otro caso

   3) se selecciona la señal siguiente en el puerto de entrada;

c) si no se ha encontrado ninguna señal habilitada, en el orden de prioridad de las *Continuous-signal*s, si las hubiera, considerando las *Continuous-signal*s de la misma prioridad en un orden arbitrario y considerando la no prioridad como la menor de las prioridades:

   1) se interpreta la *Continuous-expression* contenida en la *Continuous-signal* actual;

   2) si la señal continua actual está habilitada, se consume esa señal (véase 11.5); en cualquier otro caso

   3) se selecciona la siguiente señal continua;

d) si no se ha encontrado ninguna señal habilitada, la máquina de estados espera en el estado hasta que se recibe otro ejemplar de señal. Si el estado tiene condiciones habilitantes o señales continuas, los pasos se repiten aunque no se reciba señal alguna.

En cualquier momento, en un estado que contenga *Spontaneous-transition*s, la máquina de estados puede interpretar la *Provided-expression* de una *Spontaneous-transition* y subsiguientemente, si la *Spontaneous-transition* estuviera habilitada, la *Transition* de una de las *Spontaneous-transition*s (véase 11.9), o la *Transition* de una de las *Spontaneous-transition*s, si no existiera *Provided-expression*.

Modelo

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

Cuando varios <state>s contienen el mismo <state name>, dichos <state>s se concatenan en un <state> con dicho <state name>.

Un <state> con una <asterisk state list> se transforma en una lista de <state>s, uno para cada <state name> del cuerpo en cuestión, excepto para aquéllos <state name>s contenidos en la <asterisk state list>.

11.2.2 Aplicación de estado compuesto

Una <composite state application> prescribe que la máquina de estados tiene una estado compuesto. Las propiedades del estado compuesto se definen como parte de la <composite state application>, por un estado compuesto referenciado, o por un tipo de estado compuesto.

Gramática textual concreta

```
<composite state application> ::=
                        state <composite state list> <end> [<on exception>]
                                {       <input part>
                                |       <priority input>
                                |       <save part>
                                |       <spontaneous transition>
                                |       <continuous signal>
                                |       <connect part> }*
                        [ endstate [<state name>] <end> ]
```

Una <composite state reference> al mismo estado compuesto sólo debe aparecer en una de las <composite state application>s en la máquina de estados circundante.

Un *State-node* con *Composite-state-type identifier* representa una <composite state application>.

Gramática gráfica concreta

```
<composite state application area> ::=
                        <state symbol> contains <composite state list>
                        [ is connected to <on exception association area> ]
                        is associated with
                                {       <input association area>
                                |       <priority input association area>
                                |       <continuous signal association area>
                                |       <spontaneous transition association area>
                                |       <save association area>
                                |       <connect association area> }*

<connect association area> ::=
                        <solid association symbol> is connected to <connect area>

<composite state list> ::=
                                {<state list token> { , <state list token> }* }
                        |       {<asterisk state list>}

<state list token> ::=
                                <state name> [<actual parameters>]
                        |       <typebased composite state>
```

Un <state list token> sólo debe contener <actual parameters> si la <state area> coincide con una <nextstate area>. En este caso, la <state area> sólo debe contener un <state name> y, opcionalmente, <actual parameters>.

Semántica

La semántica de <composite state application> se define en 11.11.

11.3 Entrada

Gramática abstracta

```
Input-node              ::      [PRIORITY]
                                Signal-identifier
                                [Variable-identifier]*
                                [Provided-expression]
                                [On-exception]
                                Transition
Variable-identifier     =       Identifier
```

La longitud de la lista de *Variable-identifier* opcionales debe ser igual al número de *Sort-reference-identifier*s en la *Signal-definition* denotada por el *Signal-identifier*.

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

Gramática textual concreta

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

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

<stimulus> ::=
<signal list item>
[**(** [<variable>] { **,** [<variable>] }* **)** | <remote procedure reject>]

<remote procedure reject> ::=
raise <exception raise>

<asterisk input list> ::=
<asterisk>

Un <state> puede contener como máximo una <asterisk input list>. Un <state> no debe contener una <asterisk input list> y una <asterisk save list>.

Un <remote procedure reject> únicamente puede especificarse si el <signal list item> denota un <remote procedure identifier>. El <exception identifier> en el <remote procedure reject> debe ser mencionado en la <remote procedure definition>.

Un <signal list item> no debe denotar un <remote variable identifier> y si denota un <remote procedure identifier> o un <signal list identifier>, se deben omitir los parámetros <stimulus> (incluidos los paréntesis).

Cuando la <input list> contiene un <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 11.15.

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

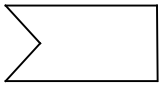
En la *Gramática abstracta*, los <remote procedure identifier>s se representan también como *Signal-identifiers*.

Gramática gráfica concreta

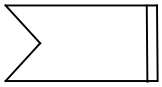
<input area> ::=
<input symbol> ***contains*** { [<virtuality>] <input list> }
[***is connected to*** <on exception association area>]
[***is associated with*** <solid association symbol> ***is connected to*** <enabling condition area>]
is followed by <transition area>

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

<plain input symbol> ::=



<internal input symbol> ::=



Un <input area> cuya <input list> contiene un <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.

NOTA – No existe diferencia entre un <plain input symbol> y un <internal input symbol>.

Semántica

Una entrada permite el consumo del ejemplar de señal de entrada especificada. El consumo de la señal de entrada pone a la disposición del agente la información transportada por la señal. A las variables asociadas con la entrada se asignan los ítems de datos transportados por la señal consumida.

Los ítems de datos 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 correspondiente ítem de datos. Si no hay un ítem de datos asociado con un género especificado en la señal, la variable correspondiente se convierte en "indefinida".

El emisor del agente consumidor (véase la cláusula 9, *Modelo*) recibe el valor pid del agente originador transportado por el ejemplar de señal.

Los ejemplares de señal que pasan del entorno a un ejemplar de agente dentro del sistema transportarán siempre un valor pid diferente de cualquiera de los del sistema.

Modelo

Un <stimulus> cuyo <signal list item> sea un <signal list identifier> es una sintaxis derivada para una lista de <stimulus>s sin parámetros e insertados en la <input list> o <priority input list> circundante. En esta lista existe una correspondencia biunívoca entre los <stimulus>s y los miembros de la lista de señales.

Cuando la lista de <stimulus>s de una <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 <textual task body> una <assignment > para cada una de las <variable>s, asignando a la <variable> el resultado de la nueva variable correspondiente. Los resultados se asignarán siguiendo el orden de izquierda a derecha de la lista de <variable>s. Esta <task> se convierte en la primera <action statement > en la <transition>.

Una <asterisk input list> se transforma en una lista de <input part>s, una para cada miembro del conjunto completo de señales de entrada válidas de la <agent definition> circundante, excepto para <signal identifier>s de las señales de entrada implícitas introducidas por los conceptos descritos en 10.5, 10.6, 11.4, 11.5 y 11.6 y para <signal identifier>s contenidos en las otras <input list>s y <save list>s del <state>.

Una <input part> o <input area> que contiene <virtuality> se denomina transición de entrada virtual. La transición de entrada virtual se describe en 8.3.3.

11.4 Entrada prioritaria

En algunos casos es conveniente expresar que la recepción de una señal tiene prioridad sobre la recepción de otras señales. Esto puede expresarse mediante una entrada de prioridad.

Gramática textual concreta

<priority input> ::=
priority input [<virtuality>]
<priority input list> <end> [<on exception>]<transition>

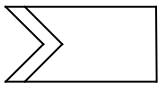
<priority input list> ::=
<stimulus> {**,** <stimulus>}*

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 connected to*** <on exception association area>]
is followed by <transition area>

<priority input symbol> ::=



Una <priority input> o <priority input association area> representa un *Input-node* con **PRIORITY**.

Semántica

Si un *Input-node* de un estado tiene **PRIORITY**, la señal es una señal con prioridad y se consumirá antes que otras señales, siempre que tenga habilitada una transición.

Modelo

Una <priority input> o <priority input area> con <virtuality> se denomina entrada de prioridad virtual. Ésta se describe en 8.3.3.

11.5 Señal continua

Cuando se describen sistemas, puede surgir una situación en la que una transición deba ser interpretada al cumplirse cierta condición. Un señal continua interpreta una expresión booleana y la transición asociada se interpreta cuando la expresión retorna el valor booleano predefinido verdadero.

Gramática abstracta

Continuous-signal	::	*Continuous-expression* [*Priority-name*] *Transition*
Continuous-expression	=	*Boolean-expression*
Priority-name	=	*Nat*

Gramática textual concreta

<continuous signal> ::=
provided [<virtuality>]
<continuous expression> <end>
[**priority** <priority name> <end>] [<on exception>] <transition>

<continuous expression> ::=
<Boolean expression>

<priority name> ::=
<Natural literal name>

Gramática gráfica concreta

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

<continuous signal area> ::=
<enabling condition symbol>
contains {
[<virtuality>] <continuous expression>
[[<end>] **priority** <priority name>] }
[***is connected to*** <on exception association area>]
is followed by <transition area>

Semántica

La *Continuous-expression* se interpreta cuando se pasa al estado al que está asociado su *Continuous-signal*, y mientras se está esperando en el estado, siempre que no se encuentre en el puerto de entrada ningún <stimulus> de una <input list> asociada. Si la *Continuous-expression* retorna el valor booleano predefinido verdadero, se habilita la señal continua.

La señal continua que tienen el valor más bajo de *Priority-name* tiene la mayor prioridad.

Modelo

Una <continuous signal> o <continuous signal area> con <virtuality> se denomina señal continua virtual. La transición continua virtual se describe en 8.3.3.

11.6 Condición habilitadora

Una condición habilitadora permite imponer una condición adicional al consumo de una señal, más allá de su recepción, así como sobre una transición espontánea.

Gramática abstracta

Provided-expression = *Boolean-expression*

Gramática textual concreta

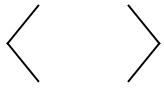
<enabling condition> ::=
provided <provided expression> <end>

<provided expression> ::=
<Boolean expression>

Gramática gráfica concreta

<enabling condition area> ::=
<enabling condition symbol> ***contains*** <provided expression>

<enabling condition symbol> ::=



Cuando la <provided expression> contiene una <imperative expression>, la *Provided-expression* es un *Value-returning-call-node* que sólo contiene el *Procedure-identifier* del procedimiento implícito definido por el *modelo* descrito a continuación. En otro caso, *Provided-expression* se representa por <provided expression>.

Semántica

La *Provided-expression* de un *Input-node* se interpreta cuando se pasa al estado al que está asociado el *Input-node*, y en cualquier momento en el que se vuelva a pasar al estado a través de la llegada de un <stimulus>. En caso de que se den múltiples condiciones de habilitación, éstas se interpretan secuencialmente en un orden arbitrario cuando se pasa el estado.

Una señal en el puerto de entrada se habilita si todas las *Provided-expression*s de un *Input-node* retornan el valor booleano predefinido verdadero, o si el *Input-node* no tiene una *Provided-expression*. La *Provided-expression* de una *Spontaneous-transition* puede ser interpretada en cualquier momento mientras el agente se encuentre en el estado.

Modelo

Cuando la <provided expression> contiene una <imperative expression> se define implícitamente el procedimiento siguiente con un nombre anónimo al que se hace referencia como *isEnabled*.

procedure *isEnabled* -> Boolean;
    start ;
    return <provided expression>;
endprocedure ;

NOTA – La <Boolean expression> puede ser transformada ulteriormente al modelo de <import expression>.

11.7 Conservación (save)

Una conservación especifica un conjunto de identificadores de señales y de identificadores de procedimientos remotos cuyos ejemplares no son relevantes para el agente 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> ::=
    save [<virtuality>] <save list> <end>

<save list> ::=
    <signal list>
  | <asterisk save list>

<asterisk save list> ::=
    <asterisk>

Un <save list> representa el *Signal-identifier*-**set**.

Un <state> puede contener como máximo una <asterisk save list>. Un <state> no debe contener simultáneamente la <asterisk input list> y la <asterisk save list>.

Gramática gráfica concreta

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

<save symbol> ::=

Semántica

No se habilita una señal en el Save-signalset.

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

El efecto de la conservación es válido solamente para el estado al cual está asociada la conservación. En el estado siguiente, los ejemplares de señal que han sido "conservados" se tratan como ejemplares de señal normales.

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 <agent definition> excepto para los <signal identifier>s de las señales de entrada implícitas introducidas por los conceptos que se describen en 10.5, 10.6, 11.4, 11.5 y 11.6 y para las <signal identifier>s contenidas en las otras <input list>s y <save list>s del <state>.

Una <save part> o <save area> que contenga <virtuality> se denomina una conservación virtual, que se describe en 8.3.3.

11.8 Transición implícita

Gramática textual concreta

Un <signal identifier> contenido en el conjunto completo de señales de entrada válidas de una <agent definition> pueden omitirse en el conjunto de <signal identifier>s contenidos en las <input list>s, <priority input list>s y en <save list> de un <state>.

Modelo

Para cada <state> existe una <input part> implícita que contiene una <transition> que sólo contiene un <nextstate> que conduce de nuevo al mismo <state>.

11.9 Transición espontánea

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

Gramática abstracta

Spontaneous-transition :: [*On-exception*]
[*Provided-expression*]
Transition

Gramática textual concreta

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

<spontaneous designator> ::=
none

Gramática gráfica concreta

<spontaneous transition area> ::=
<input symbol> ***contains*** { [<virtuality>] <spontaneous designator> }
[***is connected to*** <on exception association area>]
[***is associated with*** <solid association symbol> ***is connected to*** <enabling condition area>]
is followed by <transition area>

Semántica

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

Después de la activación de una transición espontánea la expresión **sender** del agente retorna **self**.

Modelo

Una <spontaneous transition> o <spontaneous transition area> que contiene <virtuality> se llama transición espontánea virtual. La transición espontánea virtual se describe en 8.3.3.

11.10 Etiqueta

Gramática abstracta

Free-action	::	*Connector-name* *Transition*
Connector-name	=	*Name*

Gramática textual concreta

<label> ::=

<connector name> :

<free action> ::=

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

El término "cuerpo" ("body") se utiliza para hacer referencia a un gráfico de máquina de estados, posiblemente después de la transformación desde una <statement list> y después de la transformación desde un tipo. Un cuerpo hace referencia por tanto a <agent body>, <procedure body>, <operation body>, <state machine graph area>, <procedure graph area>, <operation graph area>, o <composite state body>.

Todos los <connector name>s definidos en un cuerpo 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 cuerpo.

Sólo se permite transferencia de control a etiquetas dentro del mismo cuerpo. Está permitido tener una unión del cuerpo de la especialización con un conector definido en el supertipo.

Si la <transition string> de la <transition> en <free action> es no-vacía, el primer <action statement> debe tener una <label>, en caso contrario el <terminator statement> debe tener una <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 la correspondiente <out connector area>, con el mismo <connector name> en la misma <state machine graph area> <type state machine graph> o <procedure graph area>.

Semántica

Una *Free-action* define el objetivo de un *Join-node*. En la gramática abstracta, sólo las acciones libres tienen etiquetas; las etiquetas dentro de una transición se transforman en acciones libres independientes.

Modelo

Si una <label> no es la primera etiqueta de una <transition string>, la <transition string> se divide en dos partes. Todos los <action statement>s previos a la <label> se preservan en la transición original, que se termina con un <join> a la <label>. Todos los enunciados de acción que siguen a <label> se copian en una nueva <free action>, que comienza con la <label>.

11.11 Máquina de estados y estado compuesto

Un estado compuesto es un estado que puede constar, o bien de subestados (con transiciones asociadas) interpretados secuencialmente, o bien en una agregación de subestados interpretados en un modo entrelazado. Un subestado es un estado, de tal forma que un subestado puede a su vez ser un estado compuesto.

Las propiedades de un estado compuesto (subestados, transiciones, variables, y procedimientos) son definidas por un <composite state> o una <composite state type definition>, y la especificación de un <state> con <composite state name> dentro de una máquina de estados o un estado compuesto. Las transiciones asociadas con un estado compuesto se aplican a todos los subestados del estado compuesto.

Gramática abstracta

Composite-state-formal-parameter	=	*Agent-formal-parameter*
State-entry-point-definition	=	*Name*
State-exit-point-definition	=	*Name*
Entry-procedure-definition	=	*Procedure-definition*
Exit-procedure-definition	=	*Procedure-definition*
Named-start-node	::	*State-entry-point-name* [*On-exception*] *Transition*
State-entry-point-name	=	*Name*

Entry-procedure-definition representa un procedimiento con el nombre entrada. *Exit-procedure-definition* representa un procedimiento con el nombre salida. Estos procedimientos no pueden tener parámetros y sólo pueden contener una transición individual.

Gramática textual concreta

<composite state> ::=
<composite state graph> | <state aggregation>

Gramática gráfica concreta

<composite state area> ::=
<composite state graph area> | <state aggregation area>

Semántica

Un estado compuesto se crea cuando se crea la entidad circundante y se suprime cuando se suprime la entidad circundante.

Las variables locales se crean y se suprimen cuando se crea y se suprime, respectivamente, el estado compuesto. Si una <variable definition> contiene una <constant expression>, a la <variable definition> se asigna el resultado de la <constant expression> en el momento de la creación. Si no existe <constant expression>, el resultado de la <variable definition> es indefinido.

Composite-state-formal-parameters son variables locales que se crean cuando se crea el estado compuesto. Una variable obtiene el resultado de la expresión dada por el correspondiente parámetro real si está presente en el *Nextstate-node*. Si no es así, el resultado de la variable es indefinido.

Una transición que emane de un subestado tiene una prioridad superior que una transición conflictiva que emana de cualquiera de los estados que la contienen. Las transiciones conflictivas son transiciones activadas por la misma entrada, entrada prioritaria, conservación o señal continua.

Entry-procedure-definition y *Exit-procedure-definition*, si están definidos, se llaman de forma implícita cuando se entra y cuando se sale del estado, respectivamente. No es obligatorio definir ambos procedimientos. El procedimiento de entrada es llamado antes de invocar la transición de arranque, o si se entra de nuevo en el estado como consecuencia de la interpretación de

Nextstate-node con **HISTORY**. El procedimiento de salida se invoca después de que se interprete un *Return-node* del *Composite-state-graph* , o cuando se interpreta una transición asociada directamente al *State-node*. Cuando en un estado compuesto se genera una excepción, no se invoca el procedimiento de salida.

11.11.1 Gráfico de estado compuesto

En un gráfico de estado compuesto, las transiciones se interpretan secuencialmente.

Gramática abstracta

Composite-state-graph :: *State-transition-graph*
[*Entry-procedure-definition*]
[*Exit-procedure-definition*]
*Named-start-node**-set***

Gramática textual concreta

<composite state graph> ::=
{<package use clause>}*
<composite state heading> <end> **substructure**
[<valid input signal set>]
{<gate in definition>}*
<state connection points>*
<entity in composite state>*
<composite state body>
endsubstructure [[<qualifier>] <composite state name>] <end>

<composite state heading> ::=
state [<qualifier>] <composite state name>
[<agent formal parameters>]

<entity in composite state> ::=
<variable definition>
| <data definition>
| <select definition>
| <data type reference>
| <macro definition>
| <procedure definition>
| <exception definition>
| <composite state type definition>
| <composite state type reference>

<composite state body> ::=
[<on exception>] <start>* { <state> | <exception handler> | <free action> }*

<composite state reference> ::=
state substructure <composite state name> **referenced** <end>

Composite state graph representa <composite state body>.

Si un <composite state body> contiene al menos un <start> pero no contiene <state>, el <composite state body> se interpretará como una parte encapsulada de una transición.

Exactamente uno de los <start>s será no etiquetado. Cada punto de entrada y punto de salida etiquetado adicional tiene que ser definido por <state connection points> correspondientes.

Si un <composite state body> contiene al menos un <state> diferente del estado asterisco, tiene que estar presente un <start>.

<variable definition> en un <composite state> no puede contener **exported** <variable name>s, si el <composite state> está circundado por una <procedure definition>.

Gramática gráfica concreta

```
<composite state graph area> ::=
                        <frame symbol> contains {
                            <composite state heading>
                            {   <composite state text area>*
                                <type in composite state area>*
                                <composite state body area> } set }
                        is associated with {<graphical state connection point>* } set
                        is connected to { {<gate on diagram>}* }set
                        [ is associated with <package use area> ]

<composite state text area> ::=
                        <text symbol> contains
                        {   <valid input signal set>
                        |   <variable definition>
                        |   <data definition>
                        |   <data type reference>
                        |   <procedure definition>
                        |   <procedure reference>
                        |   <select definition>
                        |   <macro definition>}*

<type in composite state area> ::=
                            <procedure area>
                        |   <data type reference area>
                        |   <composite state type diagram>
                        |   <composite state type reference area>

<composite state body area> ::=
                        [<on exception association area>]
                        <start area>* { <state area> | <exception handler area> | <in connector area> }*
```

Tiene que haber al menos un <valid input signal set> en las <composite state text area> de una <composite state graph area>.

El <package use area> debe ser ubicado en la parte superior del <frame symbol>.

Semántica

El *State-start-node* no etiquetado del *Composite-state-graph* se interpreta como el punto de entrada por defecto del estado compuesto. Se interpreta cuando el *Nextstate-node* no tiene *State-entry-point*. Los *Named-start-nodes* se interpretan como puntos de entrada adicionales del estado compuesto. El *State-entry-point* de un *Nextstate-node* define qué transición de arranque nombrada se interpreta.

Un *Action-return-node* en un estado compuesto (sin <state exit point name>) se interpreta como el punto de salida por defecto del estado compuesto. La interpretación de un *Action-return-node* activará el *Connect-node* sin un *Name* en la entidad circundante. *Named-return-nodes* adicionales, es decir, <return> con <state exit point name>, se interpretarán como puntos de salida adicionales del estado compuesto. La interpretación de un *Named-return-node* activará una transición de salida en la entidad circundante contenida en un *Connect-node* con el mismo *Name*.

Los nodos del gráfico de estados se interpretarán de la misma manera que los nodos equivalentes de un gráfico de agente o de procedimiento. Es decir, el gráfico de estado tiene el mismo conjunto completo de señales de entrada válidas que el agente circundante, y el mismo puerto de entrada que el ejemplar del agente circundante.

Es posible especificar un <composite state> que sólo conste de transiciones recibidas en el estado asterisco, sin <start> ni subestado alguno. Dicho <composite state> se interpreta como un estado básico que contiene transiciones encapsuladas. Las transiciones encapsuladas pueden ser terminadas por <dash nextstate> o por <return>.

Modelo

Si el <composite state> no consta de ningún <state>s con <state name>s sino solamente de <state> con <asterisk>, transforma el estado asterisco en un <state> con <state name> anónimo y con un <start> que conduce a dicho <state>.

11.11.2 Agregación de estado

Una agregación de estado es una partición de un estado compuesto. Consiste en múltiples estados compuestos que tienen una interpretación entrelazada en el nivel de transición. En un instante dado cualquiera, cada partición de una agregación de estado está en uno de los estados de esa partición, o (en el caso de una sola de las particiones) en una transición, o ha completado una partición y está en espera de que se completen otras particiones. Toda transición culmina en la compleción. Pueden utilizarse agregaciones de estados para dividir el gráfico de un estado.

Gramática abstracta

State-aggregation-node	::	*State-partition-**set***
		[*Entry-procedure-definition*]
		[*Exit-procedure-definition*]
State-partition	::	*Name*
		Composite-state-type-identifier
		*Connection-definition-**set***
Connection-definition	::	*Entry-connection-definition* | *Exit-connection-definition*
Entry-connection-definition	::	*Outer-entry-point Inner-entry-point*
Outer-entry-point	::	*State-entry-point-name* | **DEFAULT**
Inner-entry-point	::	*State-entry-point-name* | **DEFAULT**
Exit-connection-definition	::	*Outer-exit-point Inner-exit-point*
Outer-exit-point	::	*State-exit-point-name* | **DEFAULT**
Inner-exit-point	::	*State-exit-point-name* | **DEFAULT**

El *State-entry-point-name* en el *Outer-entry-point* denotará una *State-entry-point-definition* de la *Composite-state-type-definition* en que aparece el *State-aggregation-node*. El *State-entry-point-name* del *Inner-entry-point* denotará una *State-entry-point-definition* del estado compuesto en la *State-partition*. Asimismo, los *State-exit-points* denotarán puntos de salida en los estados compuestos interior y exterior, respectivamente. **DEFAULT** indica los puntos de entrada y salida no etiquetados.

Todos los puntos de entrada y salida del estado contenedor y de las particiones de estados aparecerán en exactamente una *Connection-definition*.

Los conjuntos de señales de entrada de las *State-partition-**set*** dentro de un estado compuesto tienen que estar disjuntos. El conjunto de señales de entrada de una *State-partition* se define como la unión lógica de todas las señales que aparecen en un *Input-node* o el *Save-signalset* dentro del tipo de estado compuesto, incluidos los estados anidados, y los procedimientos mencionados en *Call-node*s.

Gramática textual concreta

```
<state aggregation> ::=
                        {<package use clause>}*
                        <state aggregation heading> <end> substructure
                            <state connection points>*
                            <entity in composite state>*
                            <state aggregation body>
                        endsubstructure [ [<qualifier>] <composite state name> ] <end>

<state aggregation heading> ::=
                        state aggregation [<qualifier>] <composite state name>
                        [<agent formal parameters>]

<state aggregation body> ::=
                    {   <state partitioning>
                    |   <state partition connection> }+
```

<state partitioning> ::=
<textual typebased state partition definition>
| <composite state reference>
| <composite state>

<state partition connection> ::=
connect <outer entry point> **and** <inner entry point> <end>

<outer entry point> ::=
<state partition identifier> **via** <point>

<inner entry point> ::=
<state partition identifier> **via** <point>

<point> ::=
<state entry point> | <state exit point> | **default**

Gramática gráfica concreta

<state aggregation area> ::=
<frame symbol> ***contains*** {
<state aggregation heading>
{ <composite state text area>*
<type in composite state area>*
<state aggregation body area> } ***set*** }
is associated with {<graphical state connection point>* } ***set***
is connected to { {<gate on diagram>}* }***set***
[***is associated with*** <package use area>]

<state aggregation body area> ::=
{ { <state partition area> | <connection definition area>}+ }***set***

<state partition area> ::=
<state partition reference area>
| <composite state area>
| <graphical typebased state partition definition>
| <inherited state partition definition>

<state partition reference area> ::=
<state symbol> ***contains*** <state name>

<graphical typebased state partition definition> ::=
<state symbol> ***contains*** <typebased state partition heading>

<typebased state partition heading> ::=
<state name> <colon> <composite state type expression>

<inherited state partition definition> ::=
<dashed state symbol> ***contains*** <state partition identifier>

<dashed state symbol> ::=

<connection definition area> ::=
<solid association symbol> ***is connected to*** <graphical point>

<graphical point> ::=
<graphical state connection point>
| { <state entry points> | <state exit points> } ***is associated with*** <state partition area>
| <state connection point symbol> ***is connected to*** <frame symbol>

El mismo punto de salida de estado de una partición de estado no podrá estar conectado a más de un punto de salida de estado del estado circundante.

Un punto de entrada de estado del estado circundante no podrá estar conectado a dos puntos de entrada de estado diferentes, de la misma partición de estado.

Semántica

Si una *Composite-state-type-definition* contiene un *State aggregation-node*, los estados compuestos de cada *State-partition* se interpretan de una manera entrelazada en el nivel de transición. Toda transición culmina en la compleción antes de que se interprete otra transición. La creación de un estado compuesto con partición de estado implica la creación de las *State-partition-**set*** contenidas y sus conexiones. Si la *Composite-state-type-definition* de una *State-partition* tiene *Composite-state-formal-parameter*s, estos parámetros formales están *undefined* cuando se pasa a ese estado.

Los *State-start-nodes* no etiquetados de las particiones se interpretan en cualquier orden como el punto de entrada por defecto del estado compuesto. Se interpretan cuando el *Nextstate-node* no tiene *State-entry-point*. *Named-start-nodes* se interpretan como puntos de entrada adicionales del estado compuesto. Si se pasa al estado compuesto a través del *Outer-entry-point* de *Entry-connection-definition*s, se interpreta la transición de arranque de la partición con el correspondiente *Inner-entry-point*. Se entra en estas particiones en un orden indeterminado, una vez concluido el procedimiento de entrada de la agregación de estado.

Una vez que todas y cada una de las particiones han interpretado (en cualquier orden) un *Action-Return-node* o *Named-return-node*, las particiones salen del estado compuesto. Las *Exit-connection-definitions* asocian los puntos de salida desde las particiones con los puntos de entrada del estado compuesto. Si particiones diferentes salen del estado compuesto a través de puntos de salida diferentes, el punto de salida del estado compuesto se elige de una manera no determinística. El procedimiento de salida de la agregación de estado se interpreta después de que todas las particiones de estados hayan sido completadas. Las señales pertenecientes al conjunto de señales de entrada de una partición que completaron su nodo de retorno se conservan hasta que todas las particiones hayan sido completadas.

Los nodos de los gráficos de partición de estado se interpretan de la misma manera que los nodos equivalentes de un gráfico de agente o de procedimiento, con la diferencia de que sus conjuntos de señales de entrada están disjuntos. Las particiones de estados comparten el mismo puerto de entrada que el agente circundante.

Una transición de entrada asociada con una aplicación de estado compuesto que contiene un *State-aggregation-node* se aplica a todos los estados de todas las particiones de estados. Si tal transición termina por un *Nextstate-node* con **HISTORY**, todas las particiones vuelven a pasar a sus subestados respectivos.

Modelo

Si un punto de entrada de la agregación de estado no está conectado a ningún punto de entrada de una partición de estado, se añade una conexión implícita a la entrada no etiquetada. Asimismo, si un punto de salida de una partición no está conectado a ningún punto de salida de la agregación de estado, se añade una conexión a la salida no etiquetada.

Si en el conjunto completo de entradas válidas de un agente hay señales que no son consumidas por ninguna partición de estado de un determinado estado compuesto, se añade una partición de estado implícita adicional a ese estado compuesto. Esta partición implícita tiene una sola transición de arranque no etiquetada y un solo estado que contiene todas las transiciones implícitas (incluidas las transiciones implícitas para los procedimientos exportados y para las variables exportadas). Cuando finaliza una de las otras particiones, se envía al agente una señal implícita, que es consumida por la partición implícita. Una vez que la partición implícita ha consumido todas las señales implícitas, finaliza (sale) a través de un *State-return-node*.

11.11.3 Punto de conexión de estado

Los puntos de conexión de estados se definen en estados compuestos, en estados compuestos y en tipos de estado directamente especificados, y representan puntos de conexión para la entrada y la salida de un estado compuesto.

Gramática textual concreta

<state connection points> ::=
{ **in** <state entry points> | **out** <state exit points> } <end>

<state entry points> ::=
<state entry point>
| **(** <state entry point> { **,** <state entry point> }***** **)**

<state exit points> ::=
<state exit point>
| **(** <state exit point> { **,** <state exit point> }***** **)**

<state entry point> ::=
<state entry point name>

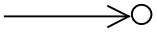
<state exit point> ::=
<state exit point name>

Gramática gráfica concreta

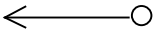
<graphical state connection point> ::=
<state connection point symbol>
is associated with { <state entry points> | <state exit points> }
is connected to <frame symbol>

<state connection point symbol> ::=
<state connection point symbol 1> | <state connection point symbol 2>

<state connection point symbol 1> ::=



<state connection point symbol 2> ::=



Para <state connection point symbol 1>, el <graphical state connection point> debe contener <state entry points>; si no es así, el <graphical state connection point> debe contener <state exit points>.

En <state connection point symbol 1> y <state connection point symbol 2>, el centro del círculo se colocará en el borde del <frame symbol> a que está conectado.

Semántica

Un <state entry point> define un punto de entrada en un <composite state>. Un <state exit point> define un punto de salida en un <composite state>.

Cada <composite state> tiene implícitamente definidos dos <state connection points>s anónimos. Son los puntos de entrada y de salida por defecto que corresponden a un <start> y a un <return> no etiquetados, respectivamente.

Un <composite state> sólo puede referirse a sus propios <state entry point>s en <start>s etiquetados, y a sus propios <state exit point>s en <return>s etiquetados.

11.11.4 Connect

Gramática abstracta

Connect-node	::	[*State-exit-point-name*] [*On-exception*] *Transition*
State-exit-point-name	=	*Name*

Gramática textual concreta

<connect part> ::=
connect [<virtuality>] [<connect list>] <end>
[<on exception>] <exit transition>

<connect list> ::=
<state exit point name> { **,** <state exit point name>}*****
| <asterisk connect list>

<exit transition> ::=
<transition>

<asterisk connect list> ::=
<asterisk>

Una <connect part> con un <state exit point> como máximo, representa un *Connect-node*. Si no se da ninguna <connect list>, se omite el *State-exit-point-name*.

La <connect list> sólo debe hacer referencia a <state exit point>s que sean visibles.

Gramática gráfica concreta

<connect area> ::=
[<virtuality>] [<connect list>]
[***is connected to*** <on exception association area>]
is followed by <exit transition area>

<exit transition area> ::=
<transition area>

Una <connect part> con un <state exit point> como máximo, representa un *Connect-node*. Si no se da ninguna <connect list>, se omite el *State-exit-point-name*.

La <connect list> sólo debe hacer referencia a <state exit point>s que sean visibles.

Semántica

Una <connect part> representa un punto de salida en un <composite state>. La interpretación se inicia de nuevo en este punto si en el <composite state> se interpreta un <return> que señala a un <state exit point> que se ha encontrado en la <connect list>.

Una <connect part> con una <connect list> vacía corresponde a un <return> no etiquetado, es decir un <return> sin <expression>, en un <composite state>.

Modelo

Cuando la <connect list> de una determinada <connect part> contiene más de un <state exit point>, se crea una copia de la <connect part> para cada uno de los <state exit point>. A continuación, la <connect part> es sustituida por dichas copias.

Una <connect list> que contiene una <asterisk connect list> se transforma en una lista de <state exit point>s, con uno para cada uno de los <state exit point> del <composite state> en cuestión. A continuación, la lista de <state exit point>s se transforma tal como se ha descrito anteriormente.

11.12 Transición

11.12.1 Cuerpo de transición

Gramática abstracta

Transition :: *Graph-node**
(*Terminator* | *Decision-node*)

Graph-node :: (*Task-node*
| *Output-node*
| *Create-request-node*
| *Call-node*
| *Compound-node*
| *Set-node*
| *Reset-node*) [*On-exception*]

Terminator :: (*Nextstate-node*
| *Stop-node*
| *Return-node*
| *Join-node*
| *Continue-node*
| *Break-node*
| *Raise-node*) [*On-exception*]

Gramática textual concreta

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

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

<action statement> ::=
[<label>]
{ <action 1> <end> [<on exception>] | <action 2> <end> }

<action 1> ::=
<task>
| <output>
| <create request>
| <decision>
| <set>
| <reset>
| <export>
| <procedure call>
| <remote procedure call>

<action 2> ::=
<transition option>

<terminator statement> ::=
[<label>]
{ <terminator 1> <end> [<on exception>] | <terminator 2> <end> }

<terminator 1> ::=
<return>
| <raise>

<terminator 2> ::=
<nextstate>
| <join>
| <stop>

Si se omite el <terminator statement> de una <transition>, la última acción de la <transition> debe contener un <decision> de terminación (véase 11.13.5) o una <transition option> de terminación, excepto 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>
| <raise area> }

<transition string area> ::=
{ <task area>
| <output 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 deberá ejecutar el agente.

La <transition area> representa *Transition* y la <transition string area> representa la lista de *Graph-node*s.

Semántica

Una transición efectúa una secuencia de acciones. Durante una transición los datos de un agente pueden manipularse y las señales pueden ser de salida. La transición terminará por la entrada de la máquina de estados del agente pasando a un estado, por una parada o por un retorno, o por la transferencia de control a otra transición.

Una transición en un proceso de un bloque puede interpretarse al mismo tiempo que una transición en otro proceso del mismo bloque (siempre que ambos no estén circundados por un proceso) o de otro bloque. Las transiciones de procesos contenidos en un proceso se interpretan por entrelazado, es decir, en cada momento sólo un proceso contenido interpreta una transición hasta que éste alcanza un estado siguiente (ejecución-hasta-la-compleción). Un modelo válido de la interpretación de un sistema SDL es un entrelazado completo de distintos procesos al nivel de todas las acciones que no pueden transformarse (por la reglas del anexo F) en otras acciones y que no están excluidos porque están en una transición en forma alternativa con una transición que se interpreta (véase 9.3).

Puede transcurrir un tiempo indefinido hasta que se interpreta una acción. Es válido que dicho tiempo varíe cada vez que se interpreta la acción. También es válido que el tiempo sea el mismo en cada interpretación o que sea cero (es decir, no se modifica el resultado de **now**, véase 12.3.4.1).

Modelo

Una acción de transición puede ser transformada en una lista de acciones (que posiblemente contenga estados implícitos) de conformidad con las reglas de transformación para <import expression> y <remote procedure call>. Para preservar un manejador de excepciones asociado con la acción, terminador o decisión original, la lista de acciones se encapsula en un procedimiento nuevo e implícitamente definido con un nombre anónimo al que se hace referencia como a *Actions*, tal como se indica a continuación, en el cual list-of-actions hace referencia a la lista resultante de acciones:

procedure *Actions*;
start;
list-of-actions;
return;
endprocedure;

La acción antigua se sustituye por una llamada a *Actions*. Si con la acción original se asoció un manejador de excepciones, éste también se asocia con la llamada a *Actions*.

Si el constructivo transformado tuvo lugar en un terminador o en una decisión, el terminador o la decisión original se sustituyen por una llamada a *Actions*, seguida del nuevo terminador o decisión. Si con la decisión o terminador original se asoció un manejador de excepciones, éste se asocia con la llamada a *Actions* y con el nuevo terminador o decisión.

Ningún manejador de excepciones se asocia con el cuerpo de *Actions* o con parte alguna de dicho cuerpo.

11.12.2 Terminador de transición

11.12.2.1 Estado siguiente

Gramática abstracta

Nextstate-node	::	*State-name* [*Nextstate-parameters*]
Nextstate-parameters	::	[*Expression*]* [*State-entry-point-name*] [**HISTORY**]

Nextstate-parameters sólo pueden estar presentes si *State-name* denota un estado compuesto.

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

Gramática textual concreta

<nextstate> ::=
    nextstate <nextstate body>

<nextstate body>::=
    <state name> [<actual parameters>] [**via** <state entry point name>]
  | <dash nextstate>

<dash nextstate> ::=
    <hyphen>
  | <history dash nextstate>

<history dash nextstate> ::=
    <history dash sign>

Un *Nextstate-node* con **HISTORY** representa un <history dash nextstate>.

Si una transición es terminada por un <history dash nextstate>, el <state> tiene que ser un <composite state>.

La <transition> contenida en un <start> no debe conducir directa o indirectamente a un <dash nextstate>. La <transition> contenida en un <start> o un <handle> no debe conducir directa o indirectamente a un <history dash nextstate>.

Una <on exception> dentro de un <start> o asociado a un cuerpo completo no debe conducir directa o indirectamente (a través de <on exception>s dentro de <exception handler>s) a un <exception handler> que contenga <dash nextstate>s.

Gramática gráfica concreta

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

Si hay <state entry point name>, el <nextstate> debe hacer referencia a un estado compuesto con punto de entrada de estado.

Si hay <actual parameters>, el <nextstate> debe hacer referencia a un estado compuesto con <formal parameters>.

Semántica

Un estado siguiente representa el terminador de una transición. Especifica el estado del agente, procedimiento o estado compuesto cuando termina la transición.

Un estado siguiente indicado por guión para un estado compuesto implica que el estado siguiente es el estado compuesto.

Si hay un *State-entry-point-name*, el estado siguiente es un estado compuesto y la interpretación continua con el *State-start-node* que tiene el mismo nombre en el *Composite-state-graph*.

Cuando se interpreta un *Nextstate-node* con **HISTORY**, el siguiente estado es uno en el cual la transición actual ha sido activada. Si la interpretación pasa de nuevo a un estado compuesto, se invoca su procedimiento de entrada.

Modelo

En cada <nextstate> de un <state>, el <dash nextstate> es sustituido por el <state name> del <state>. Este modelo se aplica tras la transformación de <state>s y las demás transaformaciones excepto las de comas finales, sinónimos, entradas prioritarias, señales continuas, condiciones habilitadoras, tareas implícitas para acciones imperativas y variables o procedimientos remotos.

En el resto de esta sección Modelo se describe cómo se determina el significado de <dash nextstate> en los manejadores de excepción.

Un manejador de excepciones se denomina alcanzable desde un estado o desde un manejador de excepciones si está asociado al estado o al manejador de excepciones, al estímulo asociado al estado o al manejador de excepciones o si está asociado a las acciones de transición que siguen al estímulo. Todos los manejadores de excepciones que son alcanzables un manejador de excepciones que, a su vez, es alcanzable desde un estado, se dice que es alcanzable desde dicho estado.

NOTA – La alcanzabilidad es transitiva.

Para cada <state>, se aplica la regla siguiente: todos los manejadores de excepciones que son alcanzables se distinguen del estado copiando cada manejador de excepciones en un <exception handler> con un nombre nuevo. Las <on exception>s se modifican utilizando este nuevo nombre. A continuación, se eliminan todos los manejadores de excepción que no son alcanzables desde ningún estado.

Después de esta sustitución, un <exception handler> que contenga <dash nextstate>s puede ser alcanzado, directa o indirectamente, exactamente desde un <state>. Los <dash nextstate>s de dentro de cada uno de los <exception handler> se sustituyen por el <state name> de dicho <state>.

11.12.2.2 Unión (Join)

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

Gramática abstracta

Join-node :: *Connector-name*

Gramática textual concreta

<join> ::=
join <connector name>

Tiene que haber exactamente un <connector name> correspondiente a un <join> dentro del mismo cuerpo. En 8.3.1 se prescribe la regla para <agent type body>.

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 toda <out connector area> en un <state machine graph area> debe haber una <in connector area> en esa <state machine graph area>.

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

Cuando se interpreta un *Join-node*, la interpretación continua con la *Free-action* denominada con *Connector-name*.

11.12.2.3 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 que el agente que la interpreta realice una parada.

Esto significa que las señales retenidas en el puerto de entrada se descartan y que la máquina de estados del agente pasa a un estado de parada. Cuando todos los agentes han dejado de existir, el propio agente deja de existir.

11.12.2.4 Retorno

Gramática abstracta

Return-node	=	*Action-return-node*
	|	*Value-return-node*
	|	*Named-return-node*
Action-return-node	::	( )
Value-return-node	::	*Expression*
Named-return-node	::	*State-exit-point-name*

Un *Return-node* debe estar contenido en un *Procedure-graph* o *Composite-state-graph*. Un *Action-return-node* sólo debe estar contenido en el *Procedure-Graph* o en un *Procedure-definition* sin *Result* o *Composite-state-graph*. Un *Value-return-node* sólo debe estar contenido en el *Procedure-Graph* de un *Procedure-definition* que contiene *Result*. Un *Named-return-node* sólo debe estar contenido en un *Composite-state-graph*.

Gramática textual concreta

<return> ::=

return [<expression> | {**via** <state exit point>}]

Gramática gráfica concreta

<return area> ::=

<return symbol>
[***is connected to*** <on exception association area>]
[***is associated with*** {<expression> | <state exit point>}]

<return symbol> ::=

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

<state exit point> se permite solamente si el ámbito circundante es un estado compuesto que contiene el <state exit point> especificado.

Semántica

Un *Return-node* de un procedimiento 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 el ejemplar de procedimiento deja de existir.

c) Si se interpreta un *Value-return-node*, el resultado de *Expression* retorna al contexto llamante.

d) En lo sucesivo, la interpretación del contexto llamante continúa en el nodo que sigue a la llamada.

Un *Return-node* en un estado compuesto produce la activación de un *Connect-node*. Para un *Named-return-node*, la interpretación continúa en el *Connect-node* con el mismo nombre. Para un *Action-return-node*, la interpretación continúa en el *Connect-node* sin un nombre.

11.12.2.5 Generación (de ejemplar de excepción)

Gramática abstracta

Raise-node :: *Exception-identifier*
[*Expression*]*

La longitud de la lista de *Expression*s opcionales debe ser la misma que el número de *Sort-reference-identifiers* de la Exception-definition denotada por el *Exception-identifier*.

Cada *Expression* debe tener un género compatible con el *Sort-reference-identifier* correspondiente (por posición) en la *Exception-definition*.

Gramática textual concreta

<raise> ::=
raise <raise body>

<raise body> ::=
<exception raise>

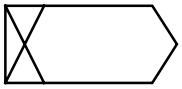
<exception raise> ::=
<exception identifier> [<actual parameters>]

Un <raise> representa un *Raise-node*.

Gramática gráfica concreta

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

<raise symbol> ::=



Semántica

La interpretación de un *Raise-node* crea un ejemplar de excepción (para la interpretación de un ejemplar de excepción, véase 11.16). Los ítems de datos transportados por el ejemplar de excepción son el resultado de los parámetros reales de <raise>. Si se omite una *Expression* de la lista de *Expression*s opcionales (es decir, si se omite la correspondiente <expression> en <actual parameters>), no se transporta ningún ítem de datos con el correspondiente lugar del ejemplar de excepción, es decir, el lugar correspondiente es "indefinido".

Si en la definición de excepción se especifica un sintipo y en el <raise> se especifica una excepción, la verificación de intervalo definida en 12.1.9.5 se aplica a la expresión.

Modelo

Una generación puede transformarse en una lista de acciones (conteniendo posiblemente estados implícitos) más una nueva generación conforme con el modelo (de llamadas de procedimiento remoto, por ejemplo). Se aplica entonces el modelo para terminadores de transición descrito en 11.12.1.

11.13 Acción

11.13.1 Tarea

Gramática abstracta

Task-node	=	*Assignment*
	|	*Assignment-attempt*
	|	*Informal-text*

Gramática textual concreta

<task> ::=
task <textual task body>

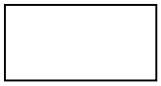
<textual task body> ::=
<assignment>
| <informal text>
| <compound statement>

Gramática gráfica concreta

<task area> ::=
<task symbol> ***contains*** <graphical task body>
[***is connected to*** <on exception association area>]

<graphical task body> ::=
<statement list>
| <informal text>

<task symbol> ::=



Puede omitirse el <end> de terminación en una <statement list> de un <graphical task body>.

Semántica

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

Un área de tarea crea su propio ámbito.

Modelo

Si la <statement list> en el <compound statement> de <textual task body> está vacía, se suprime la <task>. Si la <statement list> en un <graphical task body> está vacía, se suprime la <task area>. Todo ítem sintáctico que conduzca a la <task> o a la <task area> conducirá directamente al ítem que siga a la <task> o a la <task area>, respectivamente.

Una <task> que contenga un <compound statement> se transforma como se muestra en 11.14.1. El resultado de esta transformación se inserta en lugar de <task>.

Una <task area> que contiene una <statement list> se transforma como <compound statement> de la misma forma que el <compound statement> en una <task> (véase el párrafo anterior), excepto en que el resultado de transformar el <compound statement> en 11.14.1 se transforma, a su vez, en su gráfica equivalente que pasa a sustituir a la <task area>.

NOTA – La transformación de una <task> o <task area> que contenga un <statement list> no se corresponde necesariamente con un *Task-node* en la gramática abstracta.

11.13.2 Creación

Gramática abstracta

Create-request-node :: [*Variable-identifier*]
Agent-identifier
[*Expression*]*

La longitud de la lista de *Expression*s opcionales debe coincidir con el número de *Agent-formal-parameter*s en la *Agent-definition* del *Agent-identifier*.

Toda *Expression* que por su posición corresponda a un *Agent-formal-parameter* debe tener un género compatible con el género del *Agent-formal-parameter* en la *Agent-definition* denotado por *Agent-identifier*.

Gramática textual concreta

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

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

<actual parameters> ::=

(<actual parameter list> **)**

<actual parameter list> ::=

[<expression>] { **,** [<expression>] }*

Las comas después de la última <expression> en <actual parameter list> pueden omitirse.

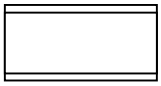
this sólo puede especificarse en una <agent type definition> y en ámbitos circundados por una <agent type definition>.

Gramática gráfica concreta

<create request area> ::=

<create request symbol> ***contains*** <create body>
[***is connected to*** <on exception association area>]

<create request symbol> ::=



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

Semántica

La acción crear causa la creación de un ejemplar de agente ya sea dentro del agente que realiza la creación o en el agente que contenga al mismo. El parent del proceso creado (véase la cláusula 9, *Modelo*) tiene el mismo pid que el que retorna en **self** del agente creador. Las expresiones **self** de los agentes creados (véase la cláusula 9, *Modelo*) y offspring del agente creador (véase la cláusula 9, *Modelo*) tienen ambas el mismo valor nuevo y único de pid.

Cuando se crea un ejemplar de agente se le da un puerto de entrada vacío, se crean variables y las expresiones de parámetros reales se interpretan en el orden dado y se asignan (como se define en 12.3.3) a los parámetros formales correspondientes. Si el agente creado tiene conjuntos de agentes contenidos, se crean los ejemplares iniciales de dichos conjuntos. Los agentes comienzan interpretando el nodo de arranque en el gráfico de agente, y se interpretan en algún orden los nodos de arranque de los agentes contenidos en un inicio antes de que se interpreten las transiciones causadas por las señales.

El agente creado se interpreta asíncrona y concurrentemente, o de forma alternativa con otros agentes dependiendo de la clase de clase de agente contenedor (sistema, bloque, proceso).

Si se intenta crear un número mayor de ejemplares de agente que el especificado por el número máximo de ejemplares en la definición de agente, no se crea ningún nuevo ejemplar, la expresión offspring del agente creador (véase la cláusula 9, *Modelo*) tiene el valor Null, y la interpretación continúa.

Si se omite una <expression> en <actual parameters>, el correspondiente parámetro formal no tiene ningún ítem de datos asociado, es decir, es "indefinido".

Si se utiliza el <agent type identifier> en una <create request>, el correspondiente tipo de agente puede no estar definido como <abstract> o contener parámetros de contexto formales.

Si en una unidad de ámbito se definen un conjunto de ejemplares y un tipo de agente con el mismo nombre, y un enunciado crear de dicha unidad de ámbito utiliza ese nombre, se crea un ejemplar en el conjunto de ejemplares que no se basa en el tipo de agente. Nótese que es posible crear un ejemplar del tipo de agente definiendo un conjunto de ejemplares basados en el mismo y creando un ejemplar en ese conjunto.

Modelo

La indicación **this** es una sintaxis derivada para el <process identifier> implícito que identifica el conjunto de ejemplares del agente en el cual se interpreta la creación.

Si se utiliza un <agent type identifier> en una <create request> se aplican los siguientes modelos:

a) Si existe un conjunto de ejemplares del tipo de agente indicado en el agente contenedor del ejemplar que realiza la creación, el <agent type identifier> es una sintaxis derivada que denota el conjunto de ejemplares.

b) Si existe más de un conjunto de ejemplares, éste se determina en el momento de la interpretación en el que el se crea el conjunto de ejemplares. En este caso, la <create request> es sustituida por una decisión no determinista utilizando una expresión **any** seguida de una rama para cada conjunto de ejemplares. En cada una de las ramas se inserta una petición de creación para el correspondiente conjunto de ejemplares.

c) Si en el agente contenedor no existe ningún conjunto de ejemplares del tipo de agente indicado, entonces:

   i) se crea en el agente contenedor un conjunto de ejemplares implícito del tipo dado con un nombre único; y

   ii) el <agent identifier> de la <create request> es una sintaxis derivada para este conjunto de ejemplares implícito.

11.13.3 Llamada a procedimiento

Gramática abstracta

Call-node	::	*Procedure-identifier* [*Expression*]*
Value-returning-call-node	::	*Procedure-identifier* [*Expression*]*

La longitud de la lista de *Expression* opcionales 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 *In-parameter* tiene que tener el mismo género que el *Procedure-formal-parameter*.

Cada *Expression* correspondiente por posición a un *Inout-parameter* o *Out-parameter* tiene que ser un *Variable-identifier* compatible en género con el género identificado por el *Sort-reference-identifier* del *Procedure-formal-parameter*.

Gramática textual concreta

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

<procedure call body> ::=
[**this**] { <procedure identifier> | <procedure type expression> } [<actual parameters>]

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

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

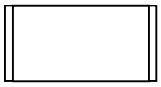
Si se utiliza **this**, <procedure identifier> debe denotar un procedimiento circundante.

La <procedure call> representa un *Call-node*. Una <value returning procedure call> (véase 12.3.5) representa un *Value-returning-call-node*.

Gramática gráfica concreta

<procedure call area> ::=
<procedure call symbol> ***contains*** <procedure call body>
[***is connected to*** <on exception association area>]

<procedure call symbol> ::=



El <procedure call area> representa un *Call-node*.

Semántica

La interpretación de un *Call-node* o *Value-returning-call-node* de procedimiento interpreta las expresiones de parámetros reales en el orden en que se presentaron y después transfiere la interpretación a la definición de procedimiento referenciada por el *Procedure-identifier*, y se interpreta dicho gráfico de procedimiento (véase la explicación en 9.4).

La interpretación de la transición que contiene una *Call-node* continúa cuando ha terminado la interpretación del procedimiento llamado.

La interpretación de la transición que contiene una *Value-returning-call-node* continúa cuando ha terminado la interpretación del procedimiento llamado. El resultado del procedimiento llamado retorna por medio del *Value-returning-call-node*.

Si se omite <expression> en <actual parameters>, el parámetro formal correspondiente no tiene ítem de datos asociado, es decir, es "indefinido".

Modelo

Si el <procedure identifier> no está definido en el servicio o agente 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.

11.13.4 Salida

Gramática abstracta

Output-node	::	*Signal-identifier* [*Expression*]* [*Signal-destination*] *Direct-via*
Signal-destination	=	*Expression* | *Agent-identifier*
Direct-via	=	(*Channel-identifier* | *Gate-identifier*)***-set***
Channel-identifier	=	*Identifier*

La longitud de la lista de *Expression* opcionales 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 compatible en género con la *Sort-identifier-reference* correspondiente (por posición) en la *Signal-definition*.

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

Para cada *Gate-identifier* in *Direct-via* tienen que existir cero o más canales tales que la puerta por conducto de este trayecto es alcanzable con el *Signal-identifier* del agente y el *Out-signal-identifier-**set*** de la puerta incluirá el *Signal-identifier*.

Gramática textual concreta

<output> ::=
output <output body>

<output body> ::=
<signal identifier> [<actual parameters>] {**,** <signal identifier> [<actual parameters>] }*
<communication constraints>

<destination> ::=
<pid expression> | <agent identifier> | **this**

<via path> ::=
via { <channel identifier> | <gate identifier> }

La <pid expression> o el <agent identifier> en <destination> representa la *Signal-destination*. Hay una ambigüedad sintáctica entre <pid expression> y <agent 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 <agent identifier>. <agent identifier> debe denotar un agente, que puede alcanzarse desde el agente originador.

<communication constraints> (véase 10.5) en una <output> no contendrá una cláusula **timer** <timer identifier>. Contiene a lo sumo una cláusula **to** <destination> y cero o más <via path>s.

Cada <via path> de <communication constraints> representa un *Channel-identifier* o *Gate-identifier* en el *Direct-via*.

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

Si <destination> es una <pid expression> con un género estático distinto del pid (véase 12.1.6) el <signal identifier> debe representar una señal definida o utilizada por la interfaz que ha definido el género del pid.

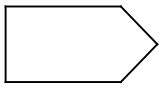
El <gate identifier> en <via path> puede utilizarse para identificar una puerta que se ha definido utilizando <textual interface gate definition> o <graphical interface gate definition>.

Gramática gráfica concreta

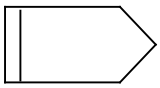
<output area> ::=
<output symbol> ***contains*** <output body>
[***is connected to*** <on exception association area>]

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

<plain output symbol> ::=



<internal output symbol>::=



NOTA 1 – No hay diferencia de significado entre un <plain output symbol> y un <internal output symbol>.

Semántica

Agent-identifier en *Signal-destination* indica que *Signal-destination* es cualquier ejemplar existente del conjunto de ejemplares de agente indicados por *Process-identifier*. Si no hay ejemplares, se descarta la señal.

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

Si hay un ejemplar de proceso que contiene el transmisor y el receptor, los ítems de datos transportados por el ejemplar de señal son el resultado de los parámetros reales de la salida. En caso contrario, los ítems de datos transportados por el ejemplar de señal son réplicas recién creadas de los resultados de los parámetros reales de la salida y no comparten referencias con los resultados de los parámetros reales de la salida. Cuando se producen ciclos de referencias en el resultado de los parámetros reales, los ítem de datos transportados también contienen dichos ciclos. Cada ítem de datos transportado es igual al correspondiente parámetro real de la salida.

Si se omite una <expression> en <actual parameters>, no se transporta ítems de datos con el correspondiente lugar del ejemplar de señal, es decir, el lugar correspondiente está "indefinido".

El pid del agente originador es también transportado por el ejemplar 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 12.1.9.5.

Si <destination> es una <pid expression> y el género estático de la expresión pid es pid, se realiza la verificación de compatibilidad para el género dinámico de la expresión pid (véase 12.1.6) para la señal que denota el *Signal-identifier*.

El ejemplar de señal pasa entonces a un trayecto de comunicación capaz de transportarlo. El conjunto de trayectos de comunicación capaces de transportar el ejemplar de señal puede limitarse, mediante la cláusula <via path>s para que incluya al menos uno de los trayectos mencionados en *Direct-via*.

Si *Signal-destination* es una *Expression*, el ejemplar de señal se entrega al ejemplar de agente denotado por *Expression*. Si este ejemplar no existe o no puede alcanzarse desde el agente originador, se descarta el ejemplar de señal.

Si *Signal-destination* es un *Agent-identifier*, el ejemplar de señal se entrega a un ejemplar arbitrario del conjunto de ejemplares de agente denotado por *Agent-identifier*. Si no existe ese ejemplar, se descarta el ejemplar de señal.

NOTA 2 – Si *Signal-destination* es Null en un *Output-node*, el ejemplar de señal se descarta cuando se interpreta 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 ejemplares de agente, al que se puede llegar por los trayectos de comunicación que pueden transportar el ejemplar de señal. Este conjunto de ejemplares de agente se elige arbitrariamente. En el segundo paso, cuando el ejemplar de señal llega al final del trayecto de comunicación, es entregado a un ejemplar del conjunto de ejemplares de agente. El ejemplar se selecciona arbitrariamente. Si no se puede seleccionar ningún ejemplar, se descarta el ejemplar de señal.

Obsérvese que el hecho de que se especifique el mismo *Channel-identifier* o *Gate-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.

Modelo

Si se especifican varios pares de <signal identifier> y <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> y <actual parameters>. La cláusula **to** <destination> y los <via path>s se repiten en cada una de las <output>s o <output area>s.

La indicación **this** en <destination es sintaxis derivada para el <agent identifier> implícito que identifica el conjunto de ejemplares para el agente en el que se interpreta la salida.

11.13.5 Decisión

Gramática abstracta

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

Las *Range-condition*s de las *Decision-answer*s debe ser mutuamente excluyentes, y las *Constant-expression*s de las *Range-condition*s deben ser de un género compatible. Si la *Decision-question* es una *Expression*, la *Range-condition* de la *Decision-answer*s debe ser compatibles en género con el género de la *Decision-question*.

Gramática textual concreta

<decision> ::=
decision <question> <end> [<on exception>]
<decision body>
enddecision

<decision body> ::=
<answer part>+ [<else part>]

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

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

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

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

Una <answer part> o <else part> en una decisión o en una opción de transición es, respectivamente, una <answer part> o <else part> de terminación si contiene una <transition> en la que se especifique un <terminator statement>, o bien, contiene una <transition string> cuyo último <action statement> contiene un decisión u opción de terminación. Una <decision> o <transition option> es, respectivamente, una decisión de terminación y una opción de transición de terminación si cada <answer part> y <else part> de su <decision body> son, respectivamente, una <answer part> o <else part> de terminación.

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

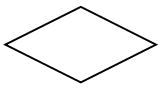
Hay una ambigüedad sintáctica entre <informal text> y <character string> en <question> y <answer>. Si la <question> y todas las <answer>s son <character string>, todas ellas se interpretan como <informal text>. Si la <question> o alguna <answer> es una <character string> que no encaja en el contexto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

Gramática gráfica concreta

<decision area> ::=
<decision symbol> ***contains*** <question>
[***is connected to*** <on exception association area>]
is followed by <graphical decision body>

<decision symbol> ::=



<graphical decision body> ::=
{ <graphical answer part>+ [<graphical else part>] } ***set***

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

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

<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>.

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*.

La <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 resultado 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 resultado de la expresión sobre la cual se plantea la pregunta no está cubierto por los resultados especificados en las otras respuestas.

En todos aquellos casos en que no se especifica la *Else-answer* y el resultado de la evaluación de expresión de pregunta no se corresponde con una de las respuestas, se genera la excepción predefinida NoMatchingAnswer.

Modelo

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

La utilización exclusiva de **any** en una <decision> es una notación taquigráfica para utilizar <any expression> en la decisión. Suponiendo que <decision body> va seguido de N <answer part>s, **any** en <decision> es 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 las <range condition>s en N <answer>s.

11.14 Lista de enunciados

Una lista de enunciados puede utilizarse en una <task>, <task area>, <procedure definition>, u <operation definition> para definir variables locales de la lista de enunciados y una serie de acciones que deben interpretarse. El objeto de una lista de enunciados es permitir descripciones textuales concisas de algoritmos que se combinan con SDL/GR o con la forma más concisa SDL/PR. La semántica de una lista de enunciados se determina por transformación de los enunciados conforme a los modelos que siguen, de tal modo que los enunciados se interpretan efectivamente de izquierda a derecha.

Un enunciado de definición de variable puede introducir variables al comienzo de una <statement list>. A diferencia de <variable definition> en 12.3.1, no se requiere que la inicialización de variables en este contexto sea una <constant expression>.

Gramática textual concreta

<statement list> ::=
<variable definitions> <statements>

<variable definitions> ::=
{ <variable definition statement> }*

<statements> ::=
<statement>*

<statement> ::=
<empty statement>
| <compound statement>
| <assignment statement>
| <algorithm action statement>
| <call statement>
| <expression statement>
| <if statement>
| <decision statement>
| <loop statement>
| <terminating statement>
| <labelled statement>
| <exception statement>

<terminating statement> ::=
<return statement>
| <break statement>
| <loop break statement>
| <loop continue statement>
| <raise statement>

Un <loop break statement> y <loop continue statement> solamente puede ocurrir dentro de un <loop statement>.

Un <terminating statement> sólo puede aparecer como el último <statement> en <statements>. Si el último <statement> en <statement list> es una <terminating statement>, la <statement list> es de terminación.

<variable definition statement> ::=
dcl <local variables of sort> { **,** <local variables of sort> }* <end>

<local variables of sort> ::=
<variable name> { **,** <variable name>}* <sort> [<is assigned sign> <expression>]

11.14.1 Enunciado compuesto

Es posible agrupar múltiples enunciados en un solo enunciado.

Gramática abstracta

Compound-node	::	*Connector-name* *Variable-definition-**set*** *Init-graph-node** *Transition* *Step-graph-node**
Init-graph-node	=	*Graph-node*
Step-graph-node	=	*Graph-node*
Continue-node	::	*Connector-name*
Break-node	::	*Connector-name*

Gramática textual concreta

<compound statement> ::=
<left curly bracket> <statement list> <right curly bracket>

El <compound statement> representa un *Compound-node*. El *Connector-name* es representado por un nombre anónimo de nueva creación. El *Variable-definition-**set*** es representado por la lista de todas las <variable definition>s en <statement list>. La *Transition* se representa por la transformación de <statements> en <statement list>, o por la transformación de <statements> en <statement list> seguida por un *Break-node* con *Connector-name*, si la <statement list> no es de terminación.

Semántica

Una <compound statement> crea su propio ámbito.

La interpretación de un *Compound-node* se efectúa como sigue:

a) Se crea una variable local para cada Variable-definition del *Variable-definition*-**set**.

b) Se interpreta la lista de Init-graph-nodes.

c) Se interpreta la Transition.

d) Cuando se interpreta un Continue-node con un Connector-name que concuerda con Connector-name, se interpreta la lista de Step-graph-nodes y la interpretación ulterior continúa en el paso c).

e) Cuando termina la interpretación del Compound-node, todas las variables creadas por la interpretación del Compound-node dejarán de existir. La interpretación de un Compound-node termina:

   i) cuando se interpreta un *Break-node*; o

   ii) cuando se interpreta un *Continue-node* con un *Connector-name* diferente del *Connector-name* en *Compound-node*; o

   iii) cuando se interpreta un *Return-node*; o

   iv) cuando se crea un ejemplar de excepción que no es tratado dentro de la *Transition* del *Compound-node*.

f) De allí en adelante, la interpretación continúa como sigue:

i) si la interpretación del *Compound-node* terminó debido a la interpretación de un *Break-node* con un *Connector-name* que concuerda con *Connector-name*, la interpretación continúa en el nodo que sigue al *Compound-node*; en otro caso

ii) si la interpretación del *Compound-node* terminó debido a la interpretación de un *Break-node*, *Continue-node* o *Return-node*, la interpretación continúa con la interpretación del *Break-node* o *Return-node*, respectivamente, en el punto de invocación del *Compound-node*; en otro caso

iii) si la interpretación del *Compound-node* terminó debido a la creación de un ejemplar de excepción, la interpretación continúa como se describe en 11.16.

Modelo

Si la <statement list> contiene <variable definitions>, se efectúa lo siguiente para cada <variable definition statement>. Se crea un nuevo <variable name> para cada <variable name> en el <variable definition statement>. Cada aparición de <variable name> en los <variable definition statement>s siguientes y dentro de <statements> se sustituye por el correspondiente <variable name> de nueva creación.

Para cada <variable definition statement> se forma una <variable definition> a partir del <variable definition statement> omitiendo la <expression> de inicialización (si está presente) y se inserta como un <variable definition statement> en lugar del <variable definition statement> original. Si está presente una <expression> de inicialización, se construye un <assignment statement> para cada <variable name> mencionado en las <local variables of sort> en su orden de aparición, donde a <variable name> se le da el resultado de <expression>. Estos <assignment statement>s se insertan al principio de <statement>s en su orden de aparición.

La <statement list> es equivalente a la concatenación de la transformación de cada <variable definition statement> y la transformación de cada <statement> en <statements> (véanse 11.14.1 a 11.14.7).

NOTA – La <statement list> no vacía transformada se convierte en una lista de <action statement>s y <terminator statement>s separados por caracteres punto y coma y se termina por punto y coma por lo que pueden tratarse como una <transition>.

Si la <statement list> está vacía, el resultado de su transformación es el texto vacío.

11.14.2 Acciones de transición y terminadores utilizados como enunciados

En una lista de enunciados, un enunciado de asignación no está precedido por la palabra clave **task**, y una llamada a procedimiento no necesita la palabra **call**. La mayoría de los demás <action statement>s que pueden utilizarse en una transición (véase 11.12.1) y el terminador <return> de una transición (véase 11.12.1), puede utilizarse como <statement>s en una <statement list>.

Gramática textual concreta

```
<assignment statement> ::=
                        <assignment> <end>

<algorithm action statement> ::=
                        <output> <end>
                    |   <create request> <end>
                    |   <set> <end>
                    |   <reset> <end>
                    |   <export> <end>

<return statement> ::=
                        <return> <end>
```

<raise statement> ::=
<raise> <end>

<call statement> ::=
[**call**] <procedure call body> <end>

Un <return statement> sólo está permitido dentro de una <procedure definition> o dentro de una <operation definition>.

La palabra clave **call** no puede omitirse si el <call statement> es sintácticamente ambiguo con una aplicación de operación o variable que tenga el mismo nombre.

NOTA – Esta ambigüedad no se resuelve por contexto.

Modelo

<assignment statement> se transforma en la <task>.

task <assignment> **;**

Un <call statement> es una sintaxis derivada para <procedure call> y se transforma en una <procedure call> con el mismo <procedure call body>:

call <procedure call body> **;**

La transformación de un <algorithm action statement>, un <return statement> y un <raise statement> se consigue dejando fuera el <end> final.

11.14.3 Expresiones como enunciados

Las expresiones que son aplicaciones de operación pueden utilizarse como enunciados, en cuyo caso, se interpreta la <operation application> y se ignora el resultado.

Gramática textual concreta

<expression statement> ::=
<operation application> <end>

Modelo

Se crea un nuevo <variable name>. Al final de la *Decl-list* se añade una <variable definition> que declara que el recién creado <variable name> es del mismo género que el resultado de la <operation application>. Finalmente, el enunciado expresión se transforma en:

task <variable name> **:=** <operation application> **;**

11.14.4 Enunciado If

Se interpreta la <Boolean expression> y si devuelve el valor booleano predefinido verdadero, se interpreta el **<**consequence statement**>**; si no es así, se interpreta la <alternative statement> si existe.

Gramática textual concreta

<if statement> ::=
if (<Boolean expression> **)** <consequence statement>
[**else** <alternative statement>]

<consequence statement> ::=
<statement>

<alternative statement> ::=
<statement>

Un <alternative statement> se asocia con el <consequence statement> precedente más próximo.

Modelo

El <if statement> es equivalente al siguiente <action statement> que implica una <decision>:

```
decision <Boolean expression> ;
   ( true ) : task { <consequence statement>-transform };
   ( false ) : task { <alternative statement>-transform };
enddecision ;
```

La transformación de <alternative statement> sólo se inserta si existe <alternative statement>.

11.14.5 Enunciado de decisión

El enunciado de decisión es una forma concisa de decisión. Se evalúa la <expression> y se interpreta la <algorithm answer part> cuya <range condition> contiene el resultado de la expresión. No están permitidas las condiciones de superposición de intervalo. A diferencia de una <decision> (véase 11.13.5), no es necesario que la expresión se ajuste a una de las condiciones de intervalo. Si no hay ajuste y si existe un <alternative statement>, éste se interpreta. Si no hay ajuste y tampoco existe <alternative statement>, la interpretación continua después del <decision statement>.

Gramática textual concreta

<decision statement> ::=
decision (<expression> **)** <left curly bracket>
<decision statement body>
<right curly bracket>

<decision statement body> ::=
<algorithm answer part>+ [<algorithm else part>]

<algorithm answer part> ::=
(<range condition> **)** <colon> <statement>

<algorithm else part> ::=
else <colon> <alternative statement>

Modelo

Una <algorithm answer part> se transforma en la siguiente <answer part>.

```
( <range condition> ) : task { <statement>-transform };
```

Se forma entonces un <decision body> tomando la transformación de cada <algorithm answer part> en orden y asociando a la misma la <else part> siguiente.

```
else : task { <alternative statement>-transform };
```

donde la transformación de <alternative statement> únicamente se inserta si hay <alternative statement>. A la <decision body> resultante se hace referencia como *Body*. La <decision statement> es equivalente a la siguiente <action statement>:

```
decision ( <expression> ) ;
   Body
enddecision ;
```

11.14.6 Enunciado de bucle

<loop statement> proporciona una facilidad generalizada para la iteración vinculada o no vinculada de un <loop body statement>, con un número arbitrario de variables de bucle. Estas variables pueden definirse dentro del <loop statement> y se estructuran en etapas tal como se especifica en <loop step>. Pueden utilizarse para generar resultados sucesivos y para acumular resultados. Cuando termina el <loop statement>, puede interpretarse una <finalization statement> en el contexto de las variables de bucle.

El <loop body statement> se interpreta repetidamente. La interpretación del bucle se controla mediante la presencia de una <loop clause>. Una <loop clause> debe comenzar con una <loop variable indication> que proporciona una forma conveniente de declarar e iniciar variables de bucle

locales. El ámbito y vida útil de cualquier variable definida en una <loop variable indication> son, de hecho, las del <loop statement>. Si la inicialización está presente como una <expression> en una <loop variable definition>, la expresión sólo se evalúa una vez antes de la primera interpretación del cuerpo del bucle. Como una alternativa, cualquier variable visible puede definirse como una variable de bucle y puede tener asignado un ítem de datos. Antes de cada iteración, se evalúan todos los elementos <Boolean expression>. La interpretación de <loop statement> se da por terminada si cualquier elemento de <Boolean expression> devuelve el valor falso. Consecuentemente, si no existe una <Boolean expression>, la interpretación del <loop statement> continuará hasta que éste alcance una salida no local. Si en dicha <loop clause> existe una <loop variable indication>, la <loop step> de cada cláusula de bucle calcula y asigna al final de cada iteración el resultado de la respectiva variable de bucle. Si en una <loop clause> no existe una <loop variable indication>, o si no hay ninguna <loop step>, no se realiza ningún enunciado de asignación a la variable de bucle. La <loop variable indication>, <Boolean expression> y <loop step> son opcionales. Una variable de bucle es visible pero no debe ser asignada en el <loop body statement>.

La interpretación del cuerpo del bucle puede también terminar cuando se alcance un **break**. El alcance de un enunciado **continue** hace que la interpretación pase inmediatamente a la siguiente iteración. (Véase también <break statement> en 11.14.7.)

Si una <loop statement> se termina "normalmente" (es decir, por una <Boolean expression> cuyo resultado de evaluación es el valor booleano predefinido falso), se interpreta el <finalization statement>. Una variable de bucle es visible y retiene su resultado cuando se interpreta la <finalization statement>. Un enunciado **break** o **continue** dentro de la <finalization statement> da por terminada la siguiente <loop statement> externa.

Gramática textual concreta

```
<loop statement> ::=
                        for ( [ <loop clause> { ; <loop clause> }* ] )
                            <loop body statement> [ then <finalization statement> ]

<loop body statement> ::=
                        <statement>

<finalization statement> ::=
                        <statement>

<loop clause> ::=
                        [<loop variable indication>]
                        , [<Boolean expression>]
                        <loop step>

<loop step> ::=
                        [ , [ { <expression> | [ call ] <procedure call body> } ] ]

<loop variable indication> ::=
                        <loop variable definition>
                    |   <variable identifier> [ <is assigned sign> <expression> ]

<loop variable definition> ::=
                        dcl <variable name> <sort> <is assigned sign> <expression>

<loop break statement> ::=
                        break <end>

<loop continue statement> ::=
                        continue <end>
```

La palabra clave **call** no puede omitirse en un <loop step> si esto condujera a una ambigüedad con una aplicación de operación o variable con el mismo nombre.

El <procedure identifier> en el <procedure call body> de un <loop step> no podrá hacer referencia a una llamada a procedimiento que retorna un valor.

Un <finalization statement> se asocia con el <loop body statement> precedente más próximo.

Un <loop statement> representa un *Compound-node.* El *Connector-name* se representa por un nombre anónimo de nueva creación, designado por *Label*.

El *Variable-definition-**set*** se representa por la lista de <variable definition statement>s construidos a partir del <variable name> y <sort> mencionados en cada <loop variable definition>.

La lista de *Init-graph-node*s se representa por la transformación de la <statement list> construida a partir de <assignment statement>s formados de cada <loop variable indication> en su orden de aparición.

Si tanto <loop variable indication> como <expression> estuvieran presentes, se construye un <assignment statement> a partir de cada <loop clause> entre el <variable name> o <variable identifier> y la <expression> en <loop step>. Se construyen <statements> tomando estos elementos <assignment statement> en secuencia o, si no se hubiera construido ningún <assignment statement>, tomando la <expression>, el <procedure call> o el <procedure call body> en <loop step>. <statements> representa la lista de *Step-graph-nodes*.

La *Transition* se representa por la siguiente <decision>, donde *Limit* hace referencia a la <expression> construida combinando todos los ítems de <Boolean expression> mediante el operador predefinido "and" de tipo booleano:

```
decision Limit ;
( True ) : task { <loop body statement>-transform }; Continue ;
( False ) : task { <finalization statement>-transform };
enddecision ;
```

La transformación de <finalization statement> se inserta únicamente si se hubiera presentado originalmente un <finalization statement>.

Un <loop continue statement> representa un *Continue-node*. El *Connector-name* se representa por *Label* del enunciado de bucle circundante más interno.

Modelo

Toda aparición de un <loop break statement> dentro de una <loop clause> o el <loop body statement> o un <finalization statement> de otra <loop statement> contenido en este <loop statement>, ninguno de los cuales aparece dentro de otro <loop statement> interior se sustituye por:

```
break Label ;
```

Si una <Boolean expression> no está presente en una <loop clause>, el valor booleano predefinido verdadero se inserta como la <Boolean expression>.

Después, el <loop statement> se sustituye por el <loop statement> así modificado seguido de un <labelled statement> con <connector name> *Break*.

11.14.7 Enunciados ruptura y etiquetado

Un <break statement> es una forma más restrictiva de una <join>.

Un <break statement> hace que la interpretación se transfiera inmediatamente al enunciado siguiente al que tiene la correspondiente <label>.

Gramática textual concreta

<break statement> ::=
break <connector name> <end>

<labelled statement> ::=
<label> <statement>

Un <break statement> tiene que estar contenido en un enunciado que haya sido etiquetado con el <connector name> dado.

Un <break statement> representa un *Break-node* con el *Connector-name* representado por <connector name>.

Un <labelled statement> representa un *Compound-node* con el *Connector-name* representado por el <connector name> en <label>.

11.14.8 Enunciado vacío

Un enunciado puede estar vacío, lo cual se indica mediante un signo punto y coma. Un <empty statement> no tiene efectos.

Gramática textual concreta

<empty statement> ::=
<end>

Modelo

La transformación del <empty statement> es texto vacío.

11.14.9 Enunciado excepción

Un enunciado puede encapsularse dentro de un manejador de excepciones.

Gramática textual concreta

<exception statement> ::=
try <try statement> <handle statement>+

<try statement> ::=
<statement>

<handle statement> ::=
handle (<exception stimulus list> **)** <statement>

Un <handle statement> se asocia con el <try statement> precedente más próximo. <try statement> no debe ser un <break statement>.

El manejador de excepciones construido en *Modelo* representa la *On-exception* del *Local-scope-node* representado por el <compound statement> que se obtiene a partir del <try statement> (véase 11.14.1).

Semántica

Un <exception statement> crea su propio ámbito con un manejador de excepciones.

Modelo

Si el <try statement> no era un <compound statement>, el <try statement> se transforma en primer lugar en un <compound statement>:

```
{ <try statement> }
```

A continuación, se transforman el <try statement> (transformado) y todos los <handle statement>s. Para cada <handle statement>, se construye el <handle> siguiente:

```
handle <exception stimulus list> ;
       task { <handle statement>-transform };
```

Los <handle>s construidos se agrupan en una lista a la que se hace referencia como *HandleParts*. Se construye un manejador de excepciones con un nombre anónimo. Se hace referencia al nombre del manejador de excepciones como *handler*.

```
exceptionhandler handler;
   HandleParts
endexceptionhandler;
```

La transformación del <exception statement> es la transformación del <try statement>.

11.15 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 lista de *Expression*s en el *Set-node* y *Reset-node* deben corresponder en posición con la lista del *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>] [<timer default initialization>]

<timer default initialization> ::=
<is assigned sign> <Duration constant expression>

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

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

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

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

Una <set clause> puede omitir <Time expression>, si <timer identifier> denota un temporizador que tiene <timer default initialization> en su definición.

Una <reset clause> representa un *Reset-node*; un <set clause> representa un *Set-node*.

No hay sintaxis gráfica correspondiente para <reset> y <set>.

Semántica

Un ejemplar 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 al mismo ejemplar de temporizador únicamente si la expresión de igualdad (véase 12.2.7) aplicada a todas las expresiones correspondientes en las dos listas produce el valor booleano predefinido verdadero (es decir, las dos listas de expresiones tienen el mismo resultado).

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 del sistema llegue a dicho valor Tiempo, al puerto de entrada del agente se aplica 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** produce el mismo resultado que la expresión **self**. Si se da una lista de expresiones cuando el temporizador está inicializado, los resultados de la expresión o expresiones 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 12.1.9.5 aplicada a la expresión correspondiente en una inicialización o reinicialización debe ser el valor booleano predefinido verdadero; si no es así, se genera la expresión predefinida OutOfRange.

Cuando un temporizador inactivo es reinicializado, sigue estando inactivo.

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

Un ejemplar de temporizador está inactivo antes de ser inicializado por primera vez.

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

Modelo

Una <set clause> sin <Time expression> es sintaxis derivada para una <set clause> en la cual <Time expression> es:

now + <Duration constant expression>

donde <Duration constant expression> se deriva de la <timer default initialization> en la definición del temporizador.

Un <reset> o un <set> puede contener varias <reset clause>s o <set clause>s respectivamente. Ello constituye sintaxis derivada para especificar una secuencia de <reset>s o <set>s, uno para cada <reset clause> o <set clause>, 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.

11.16 Excepción

Un ejemplar de excepción transfiere el control a un manejador de excepciones.

Gramática abstracta

Exception-definition	::	*Exception-name* *Sort-reference-identifier**
Exception-name	=	*Name*
Exception-identifier	=	*Identifier*

Gramática textual concreta

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

<exception definition item> ::=
<exception name> [<sort list>]

Semántica

Un ejemplar de excepción denota que ha ocurrido una situación excepcional (típicamente una situación de error) mientras se está interpretando un sistema. Un ejemplar de excepción se crea implícitamente por el sistema subyacente o de forma explícita por un *Raise-node*, dejando de existir si es capturada por un *Handle-node* o *Else-handle-node*.

La creación de un ejemplar de excepción rompe el flujo normal de control dentro de un agente, servicio, operación o procedimiento. Si un ejemplar de excepción se crea dentro de un procedimiento llamado, operación, o enunciado compuesto y no se captura allí, el procedimiento, operación, o enunciado compuesto, respectivamente, terminan y el ejemplar de excepción se propaga (dinámicamente) hacia fuera al llamante y se trata como si se hubiese creado en el mismo lugar que la llamada a procedimiento, aplicación de operación, o invocación del enunciado compuesto. Esta regla también se aplica para llamadas a procedimientos remotos; en este caso, el ejemplar de excepción se propaga hacia atrás al ejemplar del proceso llamante, además de propagarse dentro del ejemplar de agente llamado.

En el lote Predefined se predefinen una serie de tipos de excepciones. Dichos tipos de excepciones son los que el sistema subyacente puede crear de forma implícita. También está permitido que el especificador cree ejemplares de dichos tipos de excepción de forma explícita.

Si un ejemplar de excepción se crea dentro de un ejemplar de agente y no se captura en el mismo, el comportamiento ulterior del sistema es indefinido.

11.16.1 Manejador de excepciones

Gramática abstracta

Exception-handler-node	::	*Exception-handler-name* [*On-exception*] *Handle-node-**set*** [*Else-handle-node*]
Exception-handler-name	=	*Name*

Los Exception-handler-nodes dentro de un *State-transition-graph* o *Procedure-graph* dado deben tener todos distintos Exception-handler-names.

NOTA – Un *Exception-handler-name* puede tener el mismo nombre que un *State-name*. No obstante, son diferentes.

Los *Exception-identifier*s en el *Handler-node-**set*** deben ser distintos.

Gramática textual concreta

<exception handler> ::=
exceptionhandler <exception handler list> <end>
[<on exception>]
<handle>*
[**endexceptionhandler** [<exception handler name>] <end>]

<exception handler list> ::=
<exception handler name> { **,** <exception handler name> }*
| <asterisk exception handler list>

<asterisk exception handler list> ::=
<asterisk> [**(** <exception handler name> { **,** <exception handler name>}* **)**]

Cuando la <exception handler list> contiene un <exception handler name>, el <exception handler name> representa un *Exception-handler-node*. Para cada *Exception-handler-node*, el *Handle-node-**set*** se representa por las <handle part>s que contienen <exception identifier>s en sus <exception stimulus list>s. Para cada *Exception-handler-node*, el *Else-handle-node* se representa mediante un <handle> explícita o implícita en la que la <exception stimulus list> es una <asterisk exception stimulus list>.

Los <exception handler name>s de una <asterisk exception handler list> deben ser distintos y estar contenidos en otras <exception handler list>s en el cuerpo circundante o en el cuerpo de un supertipo.

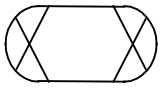
Un <exception handler> contiene como máximo una <asterisk exception stimulus list> (véase 11.16.3).

Un <exception handler> contiene como máximo un <exception handler> asociado.

Gramática gráfica concreta

<exception handler area> ::=
<exception handler symbol> ***contains*** <exception handler list>
[***is connected to*** <on exception association area>]
is associated with <exception handler body area>

<exception handler symbol> ::=



<exception handler body area> ::=
<handle association area>*

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

Una <exception handler area> representa uno o más *Exception-handler-node*s. Los <solid association symbol>s que tienen su origen en un <exception handler symbol> pueden tener un trayecto de origen común. En caso de coincidir con una <on exception area>, una <exception handler area> debe contener un <state name> (no una <asterisk state list>).

Semántica

Un manejador de excepciones representa una condición particular en la que un agente, servicio, operación o procedimiento puede manejar un ejemplar de excepción que él ha creado. El manejo de un ejemplar de excepción da lugar a una transición. No cambia el estado del proceso o procedimiento.

Si el *Exception-handler-node* no tiene un *Handle-node* con el mismo *Exception-identifier* que el ejemplar de excepción, éste es capturado por el *Else-handle-node*. Si no hay *Else-handle-node*, el ejemplar de excepción no se maneja en dicho manejador de excepciones.

Modelo

Cuando la <exception handler list> de un <exception handler> contiene más de un <exception handler name>, se crea una copia del <exception handler> para cada uno de dichos <exception handler name>. Entonces, el <exception handler> es sustituido por dichas copias.

Un <exception handler> con una <asterisk exception handler list> se transforma en una lista de <exception handler>s, uno por cada <exception handler name> del cuerpo en cuestión, excepto para aquellos <exception handler name>s contenidos en la <asterisk exception handler list>.

11.16.2 Excepción activa (On-Exception)

Gramática abstracta

On-exception :: *Exception-handler-name*

El *Exception-handler-name* que se especifica en *On-exception* debe tener el nombre de un <exception handler> dentro del mismo *State-transition-graph* o *Procedure-graph*.

Gramática textual concreta

<on exception> ::=
onexception <exception handler name> <end>

Un <exception handler name> puede aparecer en más de un <exception handler> de un cuerpo.

Gramática gráfica concreta

<on exception association area> ::=
<solid on exception association symbol> ***is connected to***
{ <on exception area> | <exception handler area> }

<solid on exception association symbol> ::=

<on exception area> ::=
<exception handler symbol> ***contains*** <exception handler name>

Un <solid on exception association symbol> pueden constar de varios segmentos de líneas verticales y horizontales. La punta de flecha debe estar asociada a la <on exception area> o <exception handler area>.

Semántica

Una *On-exception* indica cual es el manejador de excepciones al que un agente, servicio, operación o procedimiento accede si el agente, servicio o procedimiento crea un ejemplar de excepción. Por medio de una <on exception> o de una <on exception association area> se asocia un manejador de excepciones a otra entidad. Se dice que un manejador de excepciones está activo siempre que es capaz de reaccionar a la creación de un ejemplar de excepción.

Puede haber simultáneamente activos varios manejadores de excepciones; para cada ejemplar de agente, procedimiento u operación, existen varios ámbitos de excepción que pueden contener una manejador de excepciones activo. Los ámbitos de excepción, en orden creciente de carácter local, son los siguientes:

a) el gráfico completo del ejemplar,

b) los estados compuestos (si se interpreta un estado compuesto),

c) el gráfico de los estados compuestos (si los hubiera),

d) el estado actual,

e) la transición para el estímulo en el estado vigente o la transición de arranque,

f) el estado de excepción vigente,

g) la transición para la cláusula de manejo vigente, y

h) la acción vigente.

Debido al anidamiento de estados compuestos, puede haber activos simultáneamente más de un manejador de excepciones para un determinado estado compuesto o gráfico de estado compuesto.

Cuando se crea un ejemplar de excepción, los manejadores de excepciones activos se examinan en orden decreciente de carácter local. Cuando se examina un estado de excepción, el manejador de excepciones es del ámbito de excepción vigente desactivado. Si no hay activo ningún manejador de excepciones para un ámbito de excepción determinado o si el estado de excepción maneja la excepción, se examina el siguiente ámbito de excepción.

Durante la interpretación de una <constant expression> no hay activo ningún manejador de excepciones.

Un manejador de excepciones puede asociarse al gráfico completo de agente/servicio/procedimiento/operación, a un arranque de transición, a un estado, a un manejador de excepciones, a un activador de estados (por ejemplo, entrada o manejo) con su transición asociada o a un terminador de transición (a algunos tipos del mismo). El texto siguiente describe, para cada caso, cuando se activa o desactiva el manejador de excepciones.

a) *Gráfico completo de agente/procedimiento/operación*

El manejador de excepciones se activa cuando arranca la interpretación del gráfico de ejemplar de agente, operación o procedimiento; el manejador de excepciones se desactiva

cuando el ejemplar de agente, operación o procedimiento se encuentra en posición de parada o cuando deja de existir.

b) *Transición de arranque*

El manejador de excepciones se activa cuando comienza la interpretación de la transición de arranque en el agente, operación o procedimiento; el manejador de excepciones se desactiva cuando el agente, servicio o procedimiento interpreta un nodo de estado siguiente, está en situación de parada o deja de existir.

c) *Estado compuesto*

El manejador de excepciones se activa cuando se pasa al estado compuesto; está activo durante el estado compuesto, incluidos cualesquiera *Connect-nodes* o transiciones asociadas al mismo. Se desactiva cuando la interpretación pasa a otro estado.

d) *Gráfico de estado compuesto*

El manejador de excepciones se activa antes de que se invoque el procedimiento de entrada de un estado compuesto. Es desactivado después de que se completa el procedimiento de salida del estado compuesto.

e) *Estado*

El manejador de excepciones se activa siempre que el agente o procedimiento pasa a un estado dado. El manejador de excepciones se desactiva cuando el agente, o procedimiento interpreta un nodo del estado siguiente, pasa a la condición de parada o deja de existir.

f) *Manejador de excepciones*

El manejador de excepciones se activa siempre que el agente o procedimiento accede a dicho manejador de excepciones; el manejador de excepciones se desactiva cuando el agente o procedimiento pasa al nodo del estado siguiente, pasa a la condición de parada o deja de existir.

g) *Entrada*

El manejador de excepciones para el estímulo se activa siempre que en el agente o procedimiento comienza la interpretación del nodo de entrada. El manejador de excepciones se desactiva cuando el agente o procedimiento accede a un *Nextstate-node*, pasa a la condición de parada o deja de existir.

h) *Manejo*

El manejador de excepciones del *Handle-node* actual se activa siempre que en el agente o procedimiento comienza la *Transition* del *Handle-node*. El manejador de excepciones se desactiva cuando el agente o procedimiento accede a un *Nextstate-node*.

i) *Decisión*

El manejador de excepciones se activa siempre que en el agente o procedimiento arranca la interpretación de una decisión. El manejador de excepciones se desactiva cuando el agente o procedimiento accede a la transición de una rama de decisión (es decir, el manejador de excepciones abarca la expresión de la decisión y si la expresión se ajusta a cualquiera de los intervalos de las ramas de decisión).

j) *Acción de la transición (excepto decisión)*

El manejador de excepciones se activa siempre que en el agente o procedimiento comienza la interpretación de una acción. El manejador de excepciones se desactiva cuando el agente o procedimiento completan la interpretación de una acción.

k) *Terminador de transición (con expresiones)*

El manejador de excepciones se activa siempre que en el agente o procedimiento accede al terminador dado. El manejador de excepciones se desactiva cuando se completa la interpretación del terminador.

Cualquier manejador de excepciones se desactiva cuando maneja una excepción y crea un ejemplar de excepción. Los manejadores de excepciones para acciones y terminadores también abarcan las acciones que resultan del modelo para <transition>, por ejemplo <import expression>.

NOTA – Las reglas anteriores implican que en algunos casos, varios manejadores de excepciones pueden desactivarse simultáneamente. Por ejemplo, si un manejador de excepciones para un estado y otro para una transición de estado asociada estuviesen simultáneamente activos, ambos manejadores de excepciones se desactivan cuando la transición de entrada accede a un nodo del estado siguiente. Los manejadores de excepciones del contexto sintáctico abarcan los estados implícitos o estímulos: es decir, <on exception>s o <on exception association area>s se copian en el modelo.

Modelo

Cuando varios <exception handler>s contienen el mismo <exception handler name>, los <exception handler>s están concatenados en un <exception handler> que tiene dicho <exception handler name>.

En una especialización, la asociación con el manejador de excepciones se considera parte del gráfico o de la transición. Si se redefine una transición virtual, la nueva transición reemplaza una <on exception> de la transición original. Si en una especialización se hereda un gráfico o un estado, también se hereda cualquier manejador de excepciones asociado.

11.16.3 Manejo (handle)

Gramática abstracta

Handle-node	::	*Exception-identifier* [*Variable-identifier*]* [*On-exception*] *Transition*
Else-handle-node	::	[*On-exception*] *Transition*

La longitud de la lista de *Variable-identifier*s opcionales en *Handle-node* debe coincidir con el número de *Sort-reference-identifier*s en la *Exception-definition* denotada por el *Exception-identifier*.

Los géneros de las variables deben corresponder, por posición, a los géneros de los ítems de datos que pueden ser transportados por la excepción.

Gramática textual concreta

<handle> ::=
handle [<virtuality>] <exception stimulus list> <end>
[<on exception>] <transition>

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

<exception stimulus> ::=
<exception identifier> [**(** [<variable>] { **,** [<variable>] }* **)**]

<asterisk exception stimulus list> ::=
<asterisk>

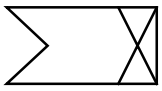
Cuando la <exception stimulus list> contiene un <exception stimulus>, la <handle> representa un *Handle-node*. Un <handle> con <asterisk exception stimulus list> representa un *Else-handle-node*.

Después de la última <variable> en <exception stimulus> pueden omitirse las comas

Gramática gráfica concreta

<handle area> ::=
<handle symbol> ***contains*** { [<virtuality>] <exception stimulus list> }
[***is connected to*** <on exception association area>]
is followed by <transition area>

<handle symbol> ::=



El trayecto a la <transition area> en <handle area> debe tener su origen en <handle symbol>.

Una <handle area> cuya <exception stimulus list> contenga un <exception stimulus>, corresponde a un *Handle-node*. Cada uno de los <exception identifier> contenidos en un <handle symbol> da el nombre a uno de los *Handle-node*s que dicho <handle symbol> representa. Una <handle area> con <asterisk exception stimulus list> representa un *Else-handle-node*.

Semántica

Un *Handle-node* consume un ejemplar del tipo de excepción especificado. El consumo del ejemplar de excepción permite que la información transportada por ésta esté disponible al agente o procedimiento. Las variables mencionadas en el *Handle-node* se asignan a los ítems de datos transportados por el ejemplar de excepción consumido.

Los ítems de datos se asignan a las variables de izquierda a derecha. Si no se menciona variable alguna para una posición de parámetro en la excepción, se descartan los ítems de datos en dicha posición. Si no se asocia ningún ítem de datos con una posición de parámetro dada, la variable correspondiente queda "indefinida".

La expresión **sender** recibe el mismo resultado que la expresión **self**.

NOTA – La expresión **state** no cambia para tomar el nombre manejador de excepciones.

Modelo

Cuando la <exception stimulus list> de un determinado <handle> contiene más de un <exception stimulus>, se crea una copia del <handle> para cada <exception stimulus>. El <handle> se sustituye entonces por dichas copias.

Cuando una o más de las <variable>s de una determinada <exception stimulus> son <indexed variable>s o <field variable>s, todas las <variable>s se sustituyen por <variable identifier>s únicas, nuevas e implícitamente declaradas. Inmediatamente antes de la <transition> del <handle>, se inserta una <task> que en su <textual task body> contiene una <assignment> para cada una de las <variable>s, asignando el resultado de la correspondiente variable nueva a la <variable>. Los resultados se asignan en orden de izquierda a derecha de la lista de <variable>s. Esta <task> pasa a ser el primer <action statement> en la <transition>.

Un <handle> o <handle area> que contiene <virtuality> se denomina una transición de manejo virtual. Las transiciones de manejo virtual se describen en 8.3.3.

12 Datos

En esta cláusula se define el concepto de datos en SDL. Ello incluye la terminología de datos, los conceptos para definir los nuevos tipos de datos y los datos predefinidos.

Cuando se habla de datos en SDL se hace referencia principalmente a tipos de datos. Un tipo de datos define un conjunto de elementos o ítems de datos a los que se hace referencia como género, y un conjunto de operaciones que pueden aplicarse a esos ítems de datos. Los géneros y operaciones definen las propiedades de los tipos de datos. Estas propiedades se definen mediante definiciones de tipos de datos.

Un tipo de datos consiste en un conjunto que constituye el *sort* (género) del tipo de datos y una o más *operations*. A título de ejemplo, considérese el tipo de datos predefinido booleano. El género booleano de los tipos de datos booleanos consta de elementos verdadero y falso. Entre las

operaciones de los tipos de datos booleanos se encuentran "=" (igualdad), "/=" (desigualdad), "not", "and", "or", "xor", and "=>" (implicación). Como un ejemplo más, considérese el tipo de datos predefinido Natural. Tiene el género Natural consistente en los elementos 0, 1, 2, etc., y las operaciones "=", "/=", "+", "-", "*", "/", "mod", "rem", "<", ">", "<=", ">=", y potencia.

SDL proporciona varios tipos de datos predefinidos que son familiares por su comportamiento y sintaxis. Los tipos de datos predefinidos se describen en el anexo D.

Las variables son objetos que pueden asociarse con un elemento de un género por asignación. Cuando se accede a una variable, se retorna el ítem de datos asociado.

Los elementos del género de un tipo de datos son *values*, u *objects* que constituyen referencias a valores, o bien, *pids*, que constituyen referencias a agentes. El género de un tipo de datos puede definirse de cualquiera de las formas siguientes:

a) Enumerando explícitamente los elementos del género.

b) Formando el producto cartesiano de los géneros S_1, S_2, … S_n; el género es igual al conjunto que consiste en todas las tuplas formadas tomando el primer elemento del género S_1, tomando el segundo elemento del género S_2, …, y finalmente, el último elemento del género S_n.

c) Los géneros de los pids se definen mediante la definición de una interfaz (véase 12.1.2).

d) Se predefinen varios géneros que forman la base de los tipos de datos predefinidos descritos en el anexo D. Los géneros predefinidos Any y pid se describen en 12.1.5 y 12.1.6.

Si los elementos de un género son objetos, el género es un género objeto. Si los elementos de un género son pids, el género es un género pid. Los elementos de un género valor son valores.

Las operaciones se definen desde y hacia elementos de géneros. Por ejemplo, la aplicación de la operación de suma ("+") desde y hacia elementos del género Entero es válida, mientras que no lo es para elementos de género booleano.

Cada ítem de datos pertenece exactamente a un género. Es decir, los géneros nunca tienen ítems de datos comunes.

Para la mayoría de los géneros hay formas literales para denotar elementos del género: por ejemplo, para Enteros se utiliza más frecuentemente "2" que "1 + 1". Puede haber más de un literal para denotar el mismo ítem de datos: por ejemplo, 12 y 012 pueden utilizarse para denotar el mismo ítem de datos Entero. La misma designación literal puede utilizarse para más de un género; por ejemplo 'A' es tanto un "carácter" como una "cadena de caracteres" de longitud uno. Algunos géneros pueden no tener formas literales para denotar elementos del género; por ejemplo, los géneros pueden también formarse como el producto cartesiano de otros géneros. En ese caso, los elementos de dichos géneros se denotan mediante operaciones que construyen los ítems de datos a partir de los elementos del género o los géneros componentes.

Una expresión denota un ítem de datos. Si una expresión no contiene una variable o una expresión imperativa, por ejemplo, si es un literal de un género dado, cada ocurrencia de la expresión denotará siempre al mismo ítem de datos. Estas expresiones "pasivas" corresponden a la utilización funcional del lenguaje.

Una expresión que contenga variables o expresiones imperativas puede interpretarse como que tiene distintos resultados durante la interpretación de un sistema SDL dependiendo del ítem de datos asociados con las variables. La utilización activa de datos incluye la asignación a variables, la utilización de variables y la inicialización de variables. La diferencia entre expresión activa y pasiva es que el resultado de una expresión pasiva es independiente de cuándo se interpreta, mientras que una expresión activa puede tener distintos resultados dependiendo de los valores, objetos o pids vigentes asociados con las variables o el estado actual del sistema.

12.1 Definiciones de datos

Las definiciones de datos se utilizan para definir tipos de datos. Los mecanismos básicos para definir datos son las definiciones de tipos de datos (véase 12.1.1) y las interfaces (véase 12.1.2). La especialización (véase 12.1.3) permite que la definición de un tipo de datos se base en otro tipo de datos, a los que se hace referencia como su supertipo. La definición del género del tipo de datos así como las operaciones implicadas para el género se dan mediante constructivos de tipos de datos (véase 12.1.7). Pueden definirse operaciones adicionales tal como se describe en 12.1.4. Por su parte, en 12.1.8 se muestra como se define el comportamiento de las operaciones de un tipo de datos.

Debido a que los datos predefinidos se definen en lotes de utilización predefinidos e implícitos (véase 7.2 y D.3), los géneros predefinidos (por ejemplo, booleano y natural) y sus operaciones pueden ser utilizados libremente a través del sistema. La semántica de la igualdad (12.2.5), expresión condicional (12.2.6) y sintipos (12.1.9.4) se apoyan en la definición del tipo de datos booleano (véase D.3.1). La semántica de la clase de nombre (véase 12.1.9.1) también se basa en la definición de carácter (Character) y cadena de caracteres (Charstring) (véanse D.3.2 y D.3.4).

Gramática abstracta

Data-type-definition	=	*Value-data-type-definition*
	|	*Object-data-type-definition*
	|	*Interface-definition*
Value-data-type-definition	::	*Sort*
		Data-type-identifier
		*Literal-signature**-set***
		*Static-operation-signature**-set***
		*Dynamic-operation-signature**-set***
Object-data-type-definition	::	*Sort*
		Data-type-identifier
		*Literal-signature**-set***
		*Static-operation-signature**-set***
		*Dynamic-operation-signature**-set***
Interface-definition	::	*Sort*
		*Data-type-identifier**
Data-type-identifier	=	*Identifier*
Sort-reference-identifier	=	*Sort-identifier*
	|	*Syntype-identifier*
	|	*Expanded-sort-identifier*
	|	*Reference-sort-identifier*
Sort-identifier	=	*Identifier*
Expanded-sort-identifier	=	*Sort-identifier*
Reference-sort-identifier	=	*Sort-identifier*
Sort	=	*Name*

Una *Data-type-definition* introduce un género que es visible en la unidad de ámbito circundante en la sintaxis abstracta. Adicionalmente puede introducir un conjunto de literales y operaciones.

Gramática textual concreta

<data definition> ::=
    <data type definition>
  | <interface definition>
  | <syntype definition>
  | <synonym definition>

Una definición de datos representa una *Data-type-definition* si es una <data type definition>, <interface definition>, o <syntype definition>.

<sort> ::=
    <basic sort>
  | <anchored sort>
  | <expanded sort>
  | <reference sort>
  | <pid sort>

<basic sort> ::=
<sort identifier>
| <syntype>

<anchored sort> ::=
this [<basic sort>]

<expanded sort> ::=
value <basic sort>

<reference sort> ::=
object <basic sort>

<pid sort> ::=
<sort identifier>

Un <anchored sort> con <basic sort> sólo se permite dentro de la definición de <basic sort>.

Un <anchored sort> es una sintaxis concreta legal sólo si se encuentra dentro de un <data type definition>. El <basic sort> en el <anchored sort> debe dar nombre al <sort> introducido por la <data type definition>.

Semántica

Una definición de datos se utiliza para la definición de un tipo de datos o interfaz, o para una expresión como se define ulteriormente en 12.1, 12.1.9.4, ó 12.1.9.6.

Cada <data type definition> introduce un género con el mismo nombre que el <data type name> (véase 12.1.1). Cada <interface definition> introduce un género con el mismo nombre que el <interface name> (véase 12.1.2).

NOTA 1 – Para evitar textos recargados se sigue el convenio de utilizar la frase "el género S" en lugar de "el género definido por el tipo de datos S" o "el género definido por la interfaz S" cuando no sea probable que con ello se produzca confusión.

Un <sort identifier> da nombre a un <sort> introducido por una definición de tipos de datos.

Un género es un conjunto de elementos: valores, objetos (es decir, referencias a valores) o pids (es decir, referencias a agentes). Dos géneros distintos no tienen elementos en común. Una <value data type definition> introduce un género que es un conjunto de valores. Una <object data type definition> introduce un género que es un conjunto de objetos. Una <interface definition> introduce un género pid.

Si un <sort> es un <expanded sort>, entonces variables, sinónimos, campos, parámetros, retornos, señales, temporizadores y excepciones definidas con dicho <sort> se asociarán a valores del género más que a referencias a dichos valores, incluso si el género se ha definido como un conjunto de objetos. Un *Expanded-sort-identifier* se representa por un <expanded sort>.

Si un <sort> es un <reference sort>, entonces variables, sinónimos, campos, parámetros, retornos, señales, temporizadores y excepciones definidas con dicho <sort> se asociarán a valores del género más que a referencias a dichos valores, incluso si el género se ha definido como un conjunto valores. Un *Referenced-sort-identifier* se representa por un <reference sort>.

El significado de un <anchored sort> figura en 12.1.3.

El <sort identifier> en un <pid sort> debe hacer referencia a género pid.

Modelo

Un <expanded sort> con un <basic sort> que representa un género valor se sustituye por el <basic sort>.

Un <reference sort> con un <basic sort> que representa un género objeto se sustituye por el <basic sort>.

NOTA 2 – En consecuencia, la palabra clave **value** no tiene efecto si el género se ha definido como un conjunto de valores, y la palabra clave **object** no tiene efecto si el género se ha definido como un conjunto de objetos.

Un <anchored sort> sin un <basic sort> es una forma taquigráfica de especificar un <basic sort> que hace referencia al nombre de la definición de tipo de datos o de la definición de sintipo en cuyo contexto aparece <anchored sort>.

12.1.1 Definición de tipos de datos

Gramática textual concreta

```
<data type definition> ::=
                        {<package use clause>}*
                        <type preamble> <data type heading> [<data type specialization>]
                        {       <end> [ <data type definition body> <data type closing> <end>]
                              | <left curly bracket> <data type definition body> <right curly bracket> }

<data type definition body> ::=
                        {<entity in data type>}* [<data type constructor>] <operations>
                        [<default initialization>[ <end> ] ]

<data type closing> ::=
                        { endvalue | endobject } type [<data type name>]

<data type heading> ::=
                        { value | object } type <data type name>
                              [ <formal context parameters> ] [<virtuality constraint>]

<entity in data type> ::=
                        <data type definition>
                      | <syntype definition>
                      | <synonym definition>
                      | <exception definition>

<operations> ::=
                        <operation signatures>
                        <operation definitions>

<data type reference> ::=
                        <type preamble>
                        { value | object } type <data type identifier> <type reference properties>
```

Una <value data type definition> contiene la palabra clave **value** en <data type heading>. Una <object data type definition> contiene la palabra clave **object** en <data type heading>.

Un <formal context parameter> de <formal context parameters> debe ser un <sort context parameter> o un <synonym context parameter>.

Las palabras clave **value** u **object** y <data type name> en un <data type closing> tienen que concordar con las palabras clave **endvalue** o **endobject**, respectivamente, y nombre en el correspondiente <data type heading>, si está presente.

Gramática gráfica concreta

```
<data type reference area> ::=
                        <type reference area>
```

La <type reference area> que forma parte de una <data type reference area> debe tener un <graphical type reference heading> que contenga un <data type name>.

Semántica

Una <data type definition> consta de <data type constructor> que describe los elementos del género (véase 12.1.6) y las operaciones inducidas por la forma en la que se construye el género, y <operations> que define un conjunto de operaciones que pueden aplicarse a los elementos de un

género (véase 12.1.4). Un tipo de datos puede también estar basado en un supertipo mediante especialización (véase 12.1.3).

12.1.2 Definición de interfaces

Las interfaces se definen en paquetes, agentes o tipos de agentes. Una interfaz define un género pid cuyos elementos son referencias a agentes.

Una interfaz puede definir señales, procedimientos remotos, variables remotas y excepciones. El contexto definidor de entidades definidas en la interfaz es la unidad de ámbito de la misma, y las entidades definidas son visibles donde la interfaz también lo es. Una interfaz también puede hacer referencia a señales, procedimientos remotos o variables remotas definidas fuera de la interfaz por la <interface use list>.

Una interfaz se utiliza en una lista de señales para denotar que las señales, llamadas a procedimientos remotos y variables remotas que son parte de la definición de la interfaz se incluyen en la lista de señales.

Gramática textual concreta

```
<interface definition> ::=
                        {<package use clause>}*
                        [<virtuality>] <interface heading>
                        [<interface specialization>]
                            <end> <entity in interface>* [<interface use list>]
                        <interface closing>
                    |   {<package use clause>}*
                    |   [<virtuality>] <interface heading>
                        [<interface specialization>] <end>
                    |   {<package use clause>}*
                    |   [<virtuality>] <interface heading>
                        [<interface specialization>] <left curly bracket>
                            <entity in interface>* [<interface use list>]
                        <right curly bracket>

<interface closing> ::=
                        endinterface [<interface name>] <end>

<interface heading> ::=
                        interface <interface name>
                            [<formal context parameters>] [<virtuality constraint>]

<entity in interface> ::=
                        <signal definition>
                    |   <interface variable definition>
                    |   <interface procedure definition>
                    |   <exception definition>

<interface use list> ::=
                        use <signal list> <end>

<interface variable definition> ::=
                        dcl <remote variable name> { , <remote variable name>}* <sort> <end>

<interface procedure definition> ::=
                        procedure <remote procedure name> <procedure signature> <end>

<interface reference> ::=
                        [<virtuality>]
                        interface <interface identifier> <type reference properties>
```

La <signal list> en una <interface definition> sólo contendrá <signal identifier>s, <remote procedure identifier>s, <remote variable identifier>s y <signal list identifier>s. Si un <signal list identifier> es parte de la <signal list> también debe respetar esta restricción.

<formal context parameters> sólo contendrá <signal context parameter>, <remote procedure context parameter>, <remote variable context parameter>, <sort context parameter> o <exception context parameter>.

El contexto definidor de entidades definidas en la interfaz (<entity in interface>) es la unidad de ámbito de la interfaz, siendo las entidades definidas visibles donde la interfaz también lo es.

Gramática gráfica concreta

<interface reference area> ::=
<type reference area>

La <type reference area> que forma parte de una <interface reference area> debe tener un <graphical type reference heading> que contenga un <interface name>.

Semántica

La semántica de <virtuality> se define en 8.3.2.

La inclusión de un <interface identifier> en una <signal list> significa que todos los identificadores de señal, identificadores de procedimientos remotos e identificadores de variables remotas que forman parte de la <interface definition> se incluyen en la <signal list>.

Modelo

Las interfaces están implícitamente definidas por las definiciones del agente, de la máquina de estados del agente, y del tipo de agente. Una interfaz definida implícitamente tiene el mismo nombre que el agente o tipo de agente que la ha definido.

NOTA 1 – Como cada agente y tipo de agente tienen una interfaz implícitamente definida con el mismo nombre, cualquier interfaz definida explícitamente debe tener un nombre distinto de cada agente y tipo de agente definidos en el mismo ámbito; de lo contrario, hay coincidencia de nombres.

La interfaz definida por una máquina de estados contiene en su <interface specialization> todas las interfaces dadas en la lista de señales entrantes asociada con puertas explícitas o implícitas de la máquina de estados. La interfaz contiene también en su <interface use list> todas las señales, variables remotas y procedimientos remotos dados en la lista de señales entrantes asociada con puertas explícitas o implícitas de la máquina de estados.

La interfaz definida por un agente o tipo de agente contiene en su <interface specialization> la interfaz definida por el estado compuesto que representa su máquina de estados.

La interfaz definida por un agente basado en tipo contiene en su <interface specialization> la interfaz definida por su tipo.

NOTA 2 – Para evitar textos recargados, se sigue el convenio de utilizar la frase "el género pid del agente A" en lugar de "el género pid definido por la interfaz implícitamente definida por el agente A" cuando no sea probable que con ello se produzca confusión.

12.1.3 Especialización de tipos de datos

La especialización permite la definición de un tipo de datos basados en otro (super) tipo. Se considera que el género definido por la especialización es un subgénero del género definido por el tipo base. El género definido por el tipo base es un supergénero del género definido por la especialización.

Gramática textual concreta

<data type specialization> ::=
inherits <data type type expression> [<renaming>] [**adding**]

<interface specialization> ::=
inherits <interface type expression> { **,** <interface type expression> }* [**adding**]

<renaming> ::=
(<rename list> **)**

<rename list> ::=
<rename pair> { **,** <rename pair> }*

<rename pair> ::=
<operation name> <equals sign> <base type operation name>
| <literal name> <equals sign> <base type literal name>

El *Data-type-identifier* en la *Data-type-definition* representada por la <data type definition> en la cual está contenida <data type specialization> (o <interface specialization>) identifica el tipo de datos representados por la <data type type expression> en su <data type specialization> (véase también 8.1.2).

Una *Interface-definition* puede también contener una lista de *Data-type-identifier*s. La interfaz denotada por una *Interface-definition* es una especialización de todas las interfaces denotadas por los *Data-type-identifier*s.

El contenido que resulta de una definición de interfaz especializada (<interface specialization>) consta del contenido de los supertipos seguido del contenido de la definición especializada. Ello implica que el conjunto de señales, procedimientos remotos y variables remotas de la definición de la interfaz especializada es la unión de las que vienen dadas por la propia definición especializada y las de los supertipos. El conjunto resultante de definiciones debe obedecer a las reglas definidas en 6.3.

El <data type constructor> debe ser del mismo tipo que el <data type constructor> utilizado en la <data type definition> del género al que hace referencia la <data type type expression> en la <data type specialization>. Es decir, si el <data type constructor> utilizado en un supertipo (directo o indirecto) es una <literal list> (<structure definition>, <choice definition>), también el <data type constructor> debe ser una <literal list> (<structure definition>, <choice definition>).

La redenominación puede utilizarse para cambiar el nombre de los literales, operadores y métodos heredados en el tipo de datos derivados.

Todos los <literal name>s y los <base type literal name>s de una <rename list> deben ser distintos.

Todos los <operation name>s y los <base type operation name>s de una <rename list> deben ser distintos.

Un <base type operation name> especificado en una <rename list> debe ser una operación cuyo <operation name> se define en la definición de tipo de datos definidora del <base type> de <data type type expression>.

Semántica

La compatibilidad de género determina cuando puede utilizarse un género en lugar de otro y cuando esto no puede hacerse. Esta relación se utiliza en asignaciones (véase 12.3.3), para pasar parámetros (véanse 12.2.7 y 9.4), para declarar de nuevo los tipos de parámetros durante la herencia (véase 12.1.2) y para parámetros de contexto reales (véase 8.1.2).

Sean T y V dos géneros. V es un género compatible con T si y solo si se cumple una de las condiciones siguientes:

a) V y T son el mismo género;

b) V es un género directamente compatible con T;

c) T se ha definido por un tipo de objeto o una interfaz y, para algún género U, V es un género compatible con U y U es un género compatible con T.

NOTA 1 – La compatibilidad de géneros es transitiva sólo para géneros definidos por tipos de objetos o interfaces, pero no para géneros definidos por tipos de valores.

Sean T y V dos géneros. V es un género directamente compatible con T si y solo si se cumple una de las condiciones siguientes:

a) V se denota por un <basic sort>, T es un género objeto y T es un supergénero de V;

b) V se denota por un <anchored sort> de la forma **this** T;

c) V se denota por un <reference sort> de la forma **object** T;

d) T se denota por un <reference sort> de la forma **object** V;

e) V se denota por un <expanded sort> de la forma **value** T;

f) T se denota por un <expanded sort> de la forma **value** V; o

g) V se denota por un <pid sort> (véase 12.1.2) y T es un supergénero de V.

Modelo

Se utiliza el modelo para la especialización de 8.2.14, ampliado de la forma que se indica a continuación.

Un tipo de datos especializado se basa en otro tipo de datos (base) utilizando una <data type definition> conjuntamente con una <data type specialization>. El género que define la especialización es disjunto del género definido por el tipo base.

Si el género definido por el tipo base tiene definidos literales, los nombre de literales se heredan como nombres para literales del género definido por el tipo especializado, salvo que se haya redenominado el literal en cuestión. La redenominación de literales tiene lugar para un literal si el nombre de literal de tipo base aparece como el segundo nombre de una <rename pair>, en cuyo caso el literal se redenomina con el primer nombre de dicha pareja.

Si el tipo base tiene definidos operadores o métodos, los nombre de operación se heredan como nombre para operadores o métodos del género definido, sujeto a las restricciones establecidas en 8.3.1, salvo que el operador o método en cuestión se hayan declarado privado (véase 12.1.9.3) o se haya redenominado el operador para dicho método u operador. La redenominación de operación se produce para un operador o método si el nombre de operación heredado aparece como segundo nombre de un <rename pair>, en cuyo caso el operador o método se redenomina con el primer nombre de dicha pareja.

Cuando varios operadores o métodos del <base type> de <sort type expression> tienen el mismo nombre que el <base type operation name> en un <rename pair>, se redenominan todos esos operadores o métodos.

En todas las ocurrencias de un <anchored sort> en el tipo especializado, el <basic sort> se sustituye por el subgénero.

Los géneros de argumento y los resultados de un operador o método heredado son los mismos que los del correspondiente operador o método del tipo base, excepto que en cada <argument> que contiene un <anchored sort> en el operador o método heredado, el <basic sort> se sustituye por el subgénero. Para métodos virtuales heredados, <argument virtuality> se añade a un <argument> que contiene un <anchored sort>, si no está ya presente.

NOTA 2 – Conforme al modelo para la especialización descrito en 8.2.1.4, solamente se hereda un operador si su signatura contiene al menos un <anchored sort> o ha tenido lugar la redenominación.

12.1.4 Operaciones

Gramática abstracta

Dynamic-operation-signature	=	*Operation-signature*
Static-operation-signature	=	*Operation-signature*
Operation-signature	::	*Operation-name* *Formal-argument** [*Result*]
Operation-name	=	*Name*
Formal-argument	=	*Virtual-argument*
	|	*Nonvirtual-argument*
Virtual-argument	::	*Argument*
Nonvirtual-argument	::	*Argument*
Argument	=	*Sort-reference-identifier*

La noción de compatibilidad de género se amplía a *Operation-signature*s. Una *Operation-signature* S1 es compatible en género con una *Operation-signature* S2 si:

a) S1 y S2 tienen el mismo número de Formal-arguments, y

b) para todos los *Virtual-argument* A de S1, el género identificado por su *Sort-reference-identifier* es compatible en género con el género especificado por el *Sort-reference-identifier* del correspondiente argumento en S2,

c) para todos los *Nonvirtual-argument* A de S1, el género identificado por su *Sort-reference-identifier* es el mismo género que el especificado por el *Sort-reference-identifier* del correspondiente argumento de S2.

Gramática textual concreta

<operation signatures> ::=
[<operator list>] [<method list>]

<operator list> ::=
operators <operation signature> { <end> <operation signature> }* <end>

<method list> ::=
methods <operation signature> { <end> <operation signature> }* <end>

<operation signature> ::=
<operation preamble>
{ <operation name> | <name class operation> }
[<arguments>] [<result>] [<raises>]

<operation preamble> ::=
[<virtuality>] [<visibility>] | [<visibility>] [<virtuality>]

<operation name> ::=
<operation name>
| <quoted operation name>

<arguments> ::=
(<argument> { **,** <argument> }* **)**

<argument> ::=
[<argument virtuality>] <formal parameter>

<formal parameter> ::=
<parameter kind> <sort>

<argument virtuality> ::=
virtual

<result> ::=
<result sign> <sort>

En una *Operation-signature*, cada *Sort-reference-identifier* en *Formal-argument* se representa por un argumento <sort>, y el *Result* se representa por el resultado <sort>. Un <sort> en un <argument> que contenga <argument virtuality> representa un *Virtual-argument*; si no es así, el <sort> del <argument> representa un *Nonvirtual-argument*.

El *Operation-name* es único en la unidad de ámbito definidora de la sintaxis abstracta aunque el correspondiente <operation name> no sea único. El *Operation-name* único se deriva de:

a) el <operator name>; más

b) la lista (posiblemente vacía) de identificadores de género de argumento; más

c) el identificador de género de resultado; más

d) el identificador de género de la definición de tipo de datos en que está definido el <operator name>.

<quoted operation name> permite adoptar para operador y método nombres con formas sintácticas especiales. La sintaxis especial se introduce de tal forma que, por ejemplo, las operaciones aritméticas y booleanas puedan tener su forma sintáctica habitual. Es decir, el usuario puede escribir "(1 + 1) = 2" en lugar de tener que escribir equal(add(1,1),2).

Si <operation signature> está contenida en una <operator list>, entonces <operation signature> representa una *Static-operation-signature*, y la <operation signature> no debe contener <virtuality> o <argument virtuality>.

Si <operation signature> está contenida en una <method list> y <virtuality> no está presente, la <operation signature> representa una *Static-operation-signature* y ninguno de los <argument>s debe contener <argument virtuality>.

Si <operation signature> está contenida en una <method list> y <virtuality> está presente, la <operation signature> representa una *Dynamic-operation-signature*. En este caso, se constituye un conjunto de *Dynamic-operation-signature*s que consta de la *Dynamic-operation-signature* representada por la <operation signature> y todos los elementos del conjunto de signaturas del método concordante en el suspertipo cuyo *Operation-name* se derive del mismo <operation name>, teniendo en cuenta la redenominación y de forma que la *Operation-signature* sea compatible en género con la *Operation-signature* del supertipo, si la hubiera.

Este conjunto debe ser cerrado en el sentido siguiente: para dos *Operation-signature*s cualesquiera S_i y S_j del conjunto de las *Operation-signature*s, la *Operation-signature* S única que cumpla que:

a) S es compatible en género con S_i y S_j; y

b) para toda *Operation-signature* S_k que sea compatible en género con S_i y S_j, dicha S_k también es compatible en género con S,

también pertenece al conjunto de *Dynamic-operation-signature*s.

Esta condición asegura que el conjunto de *Dynamic-operation-signature*s forma una retícula y garantiza que puede encontrarse una *Operation-signature* única con la mayor adaptación posible cuando se interpreta una aplicación de operación (véase 12.2.7). Si el conjunto de *Dynamic-operation-signature*s no satisface esta condición, la <sdl specification> no es válida.

NOTA – La especialización de un tipo puede exigir que se añadan *Operation-signature*s adicionales a la <method list> a fin de cumplir esta condición.

<result> en <operation signature> puede omitirse sólo si <operation signature> está presente en una <method list>.

<argument virtuality> sólo es legal si <virtuality> contiene las palabras claves **virtual** o **redefined**.

Semántica

Las formas entrecomilladas de los operadores o métodos infijos o monádicos son nombres válidos para operadores y métodos.

Un operador o método tiene un género de resultado que es el identificado por el resultado.

Modelo

Si <operation signature> está contenido en una <method list>, es una sintaxis derivada que se transforma de la forma siguiente: se construye un <argument> a partir de la palabra clave **virtual**, si <virtuality> estuviera presente, el <parameter kind> **in/out**, y el <sort identifier> del género se definen por la <data type definition> circundante. Si no hubiera <arguments>, <arguments> se forma a partir del <argument> construido y se inserta en la <operation signature>. Si existen <arguments>, el <argument> construido se añade al comienzo de la lista original de los <argument> en <arguments>.

Si el <sort> de un <argument> es un <anchored sort>, el <argument> contiene implícitamente <argument virtuality>. Si una <operation signature> contiene la palabra clave **redefined** en <virtuality>, para todo <argument> en la correspondiente <operation signature> del tipo base que contenga (implícitamente o explícitamente) <argument virtuality>, el correspondiente <argument> en <operation signature> también contiene implícitamente <argument virtuality>.

Un <argument> sin un <parameter kind> explícito, contiene el <parameter kind> **in** implícito.

12.1.5 Any

Todo valor o tipo de objeto es (directa o indirectamente) un subtipo del tipo de objeto abstracto Any. Cuando se declara que una variable es del género Any, pueden asignarse a ella ítems de datos cualquiera que sea su género de valor u objeto.

Any se define implícitamente por la siguiente <data type definition>, donde Boolean es el género booleano predefinido:

```
abstract object type Any
operators
    equal          ( this Any, this Any    ) -> Boolean;
    clone          ( this Any              ) -> this Any;
methods
    virtual is_equal ( this Any            ) -> Boolean;
    virtual copy     ( this Any            ) -> this Any;
endobject type Any;
```

NOTA 1 – Debido a que todos los constructivos de tipos de datos redefinen implícitamente los métodos virtuales de tipo de datos Any, estos métodos no puede redefinirse explícitamente en una <data type definition>.

Además, todo <object data type definition> que introduzca un género denominado S implica *Operation-signature*s equivalentes a la inclusión de la definición explícita en las siguientes <operation signature>s en la <operator list>:

```
Null        -> this S;
Make        -> this S;
```

así como la siguiente <operation signature> en la <method list>:

```
virtual finalized -> <<S>> Null;
```

Gramática textual concreta

El tipo de datos Any puede ser calificado por **package** Predefined.

Semántica

Los operadores y métodos definidos por Any están disponibles para cualquier tipo de datos valor u objeto.

Cada <object type definition> añade un ítem de datos único que denota una referencia que todavía no ha sido asociada con un valor. El operador Null retorna este ítem de datos. Todo intento de obtener un valor asociado a partir del objeto retornado por Null generará la excepción predefinida InvalidReference (véase D.3.16).

El operador Make introducido por una <object data type definition> crea un nuevo elemento no inicializado del <result sort> del operador Make. Cada <object data type definition> proporciona una definición adecuada para el operador Make.

El operador *equal* compara la posible igualdad entre dos valores (cuando se definen por el tipo de valor), o compara dos valores referenciados por objetos para estimar la igualdad (cuando están definidos por un tipo de objeto). Si X e Y son los resultados de su parámetro real *Expression*s, entonces:

a) si X o Y es Null, el resultado es valor booleano predefinido verdadero si ambos son Null, y el valor booleano predefinido falso si uno es Null; en otro caso

b) si el género dinámico de Y no es compatible en género con el género dinámico de X, el resultado es el valor booleano predefinido falso; en otro caso

c) el resultado se obtiene interpretando *x.is_equal(y)*, donde *x* e *y* representan a X e Y, respectivamente.

El operador *clone* crea un nuevo ítem de datos que pertenece al género de su parámetro real e inicializa el ítem de datos recién creado aplicando copy a dicho ítem de datos, dando así el parámetro actual vigente. Después de aplicar *clone*, el ítem de datos recién creado es igual al parámetro real. Sea Y el resultado de su parámetro real *Expression*, entonces, el operador *clone* se define como:

a) si Y es Null, el resultado es Null; si no es así

b) si el género de X es un género de objeto, sea X el resultado de la interpretación del operador *Make* para el tipo de datos definido por el género de Y. El resultado se obtiene interpretando *x.copy(y)*, donde *x* e *y* representan a X e Y, respectivamente; en otro caso

c) si el género de X es un género de valor, sea X un elemento arbitrario del género de X. El resultado se obtiene interpretando *x.copy*(*y*), donde *x* e *y* representan a X e Y, respectivamente.

El método *is_equal* compara **this** con el parámetro real, componente a componente, si hubiera alguno. En general, para que el método de *is_equal* de un valor booleano predefinido verdadero, ni **this** ni el parámetro real deben ser Null, y el género del parámetro real debe ser compatible en género con el género de **this**.

Las definiciones de tipos de datos pueden redefinir *is_equal* para tener en cuenta las diferencias de semántica de sus correspondientes géneros. Los constructivos de tipo redefinirán implícitamente *is_equal* como se indica a continuación. Sean X e Y los resultados de su parámetro real *Expression*s, entonces:

a) si el género de X ha sido construido por una <literal list>, el resultado es el valor booleano predefinido verdadero si X e Y tienen el mismo valor;

b) si el género de X ha sido construido por una <structure definition>, el resultado es el valor booleano predefinido verdadero si todo componente de X es igual al correspondiente componente de Y, tal como se determina interpretando una *Equality-expression* con dichos componentes como operandos, omitiendo los componentes de X e Y que se han definido como opcionales y que no están presentes.

El método *copy* copia el parámetro real en **this**, componente a componente, si existe alguno. Cada constructivo de tipo de datos añade un método que redefine *copy*. En general, ni **this** ni el parámetro real deben ser Null y el género del parámetro real debe ser compatible en género con el género de **this**. Toda redefinición de *copy* debe satisfacer la poscondición de que después de aplicar el método *copy*, **this** *is_equal* al parámetro real.

Las redefiniciones de tipos de datos pueden redefinir *copy* para tener en cuenta las diferencias de semántica de sus correspondientes géneros. Los constructivos tipo redefinen automáticamente *copy* tal como se indica a continuación. Sean X e Y los resultados de su parámetro real *Expression*s, entonces:

a) si el género de X ha sido construido por una <literal list>, Y se copia en X;

b) si el género de X ha sido construido por una <structure definition>, para todo componente de X, el correspondiente componente de Y se copia en dicho componente de X interpretando *xc.modify*(*yc*) – donde *xc* representa el componente de X, *modify* es el método de modificación de campo para ese componente, e *yc* representa el correspondiente componente de Y, omitiendo los componentes de X e Y que se han definido como opcionales y que no están presentes.

NOTA 2 – La interpretación del método *modify* implica la asignación del parámetro real al parámetro formal y, en consecuencia, una llamada recurrente al método *copy* (véase 12.3.3).

Modelo

Si una <data type definition> no contiene una <data type specialization>, se trata de una notación taquigráfica para una <data type definition> con una <data type specialization>

inherits Any;

12.1.6 pid y géneros de pid

Toda interfaz es (directa o indirectamente) un subtipo de la interfaz pid. Cuando se declara que una variable es del género pid, a dicha variable pueden asignarse ítems de datos pertenecientes a cualquier género pid.

Gramática textual concreta

El tipo de datos pid puede ser calificado por **Package** Predefined.

Semántica

El género pid contiene un solo ítem de datos denotado por el literal Null. Null representa una referencia no asociada con ningún agente.

Una *Interface-definition* representada por una <interface definition> sin una <interface specialization> contiene solamente un *Data-type-identifier* que denota la interfaz pid.

Un elemento de un género pid introducido por una interfaz implícitamente definida por una definición de agente, está asociada a una referencia al agente por la interpretación de un *Create-request-node* (véase 11.13.2).

Cada interfaz añade una operación de verificación de compatibilidad que, dada una señal, determinará:

a) si la señal se define o se utiliza en la interfaz; o

b) si la señal está contenida en una puerta de entrada conectada a la máquina de estados del agente que ha definido implícitamente la interfaz; o

c) si la verificación de compatibilidad se cumple para un género pid definido por una interfaz contenida en su <interface specialization>.

Si ello no se cumple, se genera la excepción predefinida InvalidReference (véase D.3.16). La verificación de compatibilidad se define similarmente para variables remotas (véase 10.6) y procedimientos remotos (véase 10.5).

NOTA – Un género pid puede asignarse polimórficamente (véase 12.3.3).

12.1.7 Constructivos de tipo de datos

Los constructivos de tipo de datos especifican el contenido del género de un tipo de datos, ya sea enumerando los elementos que constituyen el género o recopilando todos los ítems de datos que pueden obtenerse construyendo una tupla a partir de los elementos de los géneros dados.

Gramática textual concreta

<data type constructor> ::=
<literal list>
| <structure definition>
| <choice definition>

12.1.7.1 Literales

El constructivo de tipo de datos literal especifica el contenido del género de un tipo de datos enumerando los (posiblemente muchísimos) elementos del género. El constructivo de tipo de datos literal define implícitamente operaciones que permiten una comparación entre los elementos del género. Los elementos de un género literal se denominan literales.

Gramática abstracta

Literal-signature	::	*Literal-name* *Result*
Literal-name	=	*Name*

Gramática textual concreta

<literal list> ::=
[<visibility>] **literals** <literal signature> { **,** <literal signature> }* <end>

<literal signature> ::=
<literal name>
| <name class literal>
| <named number>

<literal name> ::=
<literal name>
| <string name>

<named number> ::=
<literal name> <equals sign> <Natural simple expression>

En una *Literal-signature*, el *Result* es el género introducido por la <data type definition> que define la <literal signature>.

El *Literal-name* es único dentro de la unidad de ámbito definidora en la sintaxis abstracta, incluso aunque el <literal name> no sea único. El *Literal-name* único se deriva de:

a) el <literal name>; más

b) el identificador de género de la definición del tipo de datos en la que se define el <literal name>.

NOTA – Un <string name> es una de las unidades de léxico <character string>, <bit string> y <hex string>.

Cada resultado de <Natural simple expression> que tenga lugar en un <named number> debe ser único entre todas las <literal signature>s en la <literal list>.

Semántica

Una <literal list> define un género enumerando todos los elementos del conjunto. Cada elemento del género está representado por una *Literal-signature*.

Los literales formados a partir de <character string> se utilizan para los géneros de datos predefinidos cadena de caracteres (Charstring) (véase D.3.4) y carácter (Character) (véase D.3.2). Estos también tienen una relación especial con las <regular expression>s (véase 12.1.9.1). Los literales formados a partir de <bit string> y <hex string> también se utilizan para el género de datos predefinido Integer (véase D.3.5). Estos literales pueden también definirse para otros usos.

Una <literal list> redefine las operaciones heredadas (directa o indirectamente) de Any tal como se describe en 12.1.5.

El significado de <visibility> en una <literal list> se explica en 12.1.9.3.

Modelo

Un <literal name> en una <literal list> es sintaxis derivada para un <named number> que contiene el <literal name> y una que contiene <Natural simple expression> que denota el valor Natural no negativo más bajo posible que no aparece en ninguna otra <literal signature>s de la <literal list>. La sustitución de <literal name>s por <named number>s tiene lugar uno a uno de izquierda a derecha.

Una lista de literales es sintaxis derivada para la definición de operadores que ordenen los elementos en el género definido por la <literal list>:

a) operadores que comparen dos ítems de datos con respecto a la ordenación establecida;

b) operadores que devuelvan el ítem de datos primero, último, siguiente o previo del ordenamiento; y

c) un operador que ofrezca la posición de cada ítem de datos del ordenamiento.

Una <data type definition> que mediante una <literal list> introduzca un género denominado S supone un conjunto de *Static-operation-signature*s equivalentes a las definiciones implícitas en la siguiente <operator list>:

```
"<"   ( this S, this S )   -> Boolean;
">"   ( this S, this S )   -> Boolean;
"<="  ( this S, this S )   -> Boolean;
">="  ( this S, this S )   -> Boolean;
first                      -> this S;
last                       -> this S;
succ  ( this S )           -> this S;
pred  ( this S )           -> this S;
num   ( this S )           -> Natural;
```

donde Boolean es el género predefinido Booleano, y Natural es el género Natural.

Las <literal signature>s en una <data type definition> se denominan en el orden ascendente de la <Natural simple expression>s. Por ejemplo,

```
literals C = 3, A, B;
```

implica A<B y B<C.

Los operadores de comparación "<" (">","<=",">=") representan la comparación normalizada menor que (mayor que, menor o igual que y mayor o igual que) entre las <Natural simple expression>s de dos literales. El operador first devuelve el primer ítem de datos del ordenamiento (el literal con la <Natural simple expression> más baja). El operador last devuelve el último ítem de datos del ordenamiento (el literal con la <Natural simple expression> más alta). El operador pred devuelve el ítem de datos precedente del ordenamiento, si hay alguno o, si no es así, el último ítem de datos. El

operador succ devuelve el ítem de datos siguiente del ordenamiento, si hay alguno o, si no es así, el primer ítem de datos. El operador num devuelve el valor Natural correspondiente a la <Natural simple expression> del literal.

Si <literal signature> es una <regular expression>, es una notación taquigráfica para enumerar un conjunto (posiblemente infinito) de <literal name>s tal como se describe en 12.1.9.1.

12.1.7.2 Tipos de datos estructura

El constructivo tipo de datos estructura determina el contenido de un género formando el producto cartesiano de un conjunto de géneros dados. Los elementos de un género estructura se denominan estructuras. El constructivo tipo de datos estructura define implícitamente operaciones que construyen estructuras a partir de los elementos de los géneros componentes, operaciones de proyección para acceder a los elementos componentes de una estructura, así como operaciones para actualizar los elementos componentes de una estructura.

Gramática textual concreta

<structure definition> ::=
[<visibility>] **struct** [<field list>] <end>

<field list> ::=
<field> { <end> <field> }*

<field> ::=
<fields of sort>
| <fields of sort> **optional**
| <fields of sort> <field default initialization>

<fields of sort> ::=
[<visibility>] <field name> { **,** <field name> }* <field sort>

<field default initialization> ::=
default <constant expression>

<field sort> ::=
<sort>

Cada <field name> de un género estructura debe ser distinto de cualquier otro <field name> de la misma <structure definition>.

Semántica

Una <structure definition> define un género estructura cuyos elementos son todos tuplas que pueden construirse a partir de ítems de datos que pertenezcan a los géneros dados en <field list>. Un elemento de un género estructura tiene tantos elementos componentes como hay en los <field>s de la <field list>, aunque un campo pueda no estar asociado con un ítem de datos si el correspondiente <field> se ha declarado con la palabra clave **optional**, o no ha sido aún inicializado.

Una <structure definition> redefine las operaciones heredadas (directa o indirectamente) de Any tal como se describe en 12.1.5.

El significado de <visibility> en <fields of sort> y <structure definition> se explica en 12.1.9.3.

Modelo

Una <field list> que contiene un <field> con una lista de <field name>s en un <fields of sort>, es sintaxis concreta derivada en la que <field> se sustituye por una lista de <field>s separada por un <end>, de forma que cada <field> de la lista se ha obtenido copiando el <field> original y sustituyendo un <field name> de la lista de <field name>s para todos los <field name> de la lista.

Una definición de estructura es sintaxis derivada para la definición de:

a) un operador, Make, para crear estructuras;

b) métodos para modificar estructuras y acceder a ítems de datos componentes de estructuras; y

c) métodos para verificar la presencia de ítem de datos componentes opcionales en las estructuras.

Los <arguments> del operador Make contienen la lista de <field sort>s que aparecen en la lista de campos en el mismo orden en que aparecen. El resultado <sort> del operador Make es el identificador de género del género estructura. El operador Make crea una nueva estructura y asocia cada campo con el resultado del correspondiente parámetro formal. Si se ha omitido el parámetro real en la aplicación del operador Make, el campo correspondiente no obtiene ningún valor, es decir, se torna "indefinido".

Una <structure definition> que introduzca un género denominado S supone un conjunto de *Dynamic-operation-signatures* equivalentes a las definiciones explícitas en la siguiente <method list>, para cada <field> en su <field list>:

virtual *field-modify-operation-name* (<field sort>) -> S;
virtual *field-extract-operation-name* -> <field sort>;
field-presence-operation-name -> Boolean;

donde Boolean es el género predefinido Booleano, y <field sort> es el género del campo.

El nombre del método implícito para modificar un campo, *field-modify-operation-name*, es el nombre del campo concatenado con "Modify". El método implícito para modificar un campo asocia éste con el resultado de su argumento *Expression*. Cuando <field sort> es un <anchored sort>, esta asociación sólo tiene lugar si el género dinámico del argumento *Expression* es compatible con el género <field sort> de ese campo. En cualquier otro caso, se genera la excepción predefinida UndefinedField (véase D.3.16).

El nombre del método implícito para acceder a un campo, *field-extract-operation-name*, es el nombre del campo concatenado con "Extract". El método para acceder a un campo devuelve el ítem de datos asociado a dicho campo. Si durante la interpretación el campo de una estructura es "indefinido", entonces la aplicación del método de acceso a ese campo a la estructura provoca que se genere la excepción predefinida UndefinedField.

El nombre del método implícito para verificar la presencia de un ítem de datos de campo, *field-presence-operation-name*, es el nombre del campo concatenado con "Present". El método para verificar la presencia de un ítem de datos de campo devuelve el valor booleano predefinido falso si el campo es "indefinido", y el valor booleano predefinido verdadero en cualquier otro caso. Únicamente se define un método para verificar la presencia de un ítem de datos de campo si el <field> contiene la palabra clave **optional**.

Si un <field> se define con una <field default initialization>, es sintaxis derivada para la definición de dicho <field> como opcional. Cuando se crea una estructura de este género y no se proporciona un argumento real para el campo por defecto, se añade una modificación inmediata del campo por la <constant expression> asociada tras la creación de una estructura.

12.1.7.3 Tipos de datos elección

Un constructivo de tipos de datos elección es una notación taquigráfica para definir un tipo de estructura cuyos componentes son todos opcionales y asegurando que cada ítem de datos estructura tendrá siempre exactamente un ítem de datos componente. El tipo de datos elección simula, por tanto, un género que es la suma disjunta de los elementos de los géneros componentes.

Gramática textual concreta

<choice definition> ::=
[<visibility>] **choice** [<choice list>] <end>

<choice list> ::=
<choice of sort> { <end> <choice of sort> }*

<choice of sort> ::=
[<visibility>] <field name> <field sort>

Cada <field name> de un género elección debe ser distinto de cualquier otro <field name> de la misma <choice definition>.

Semántica

Una <choice definition> redefine las operaciones heredadas (directa o indirectamente) de Any tal como se describe en 12.1.5.

El significado de <visibility> en <choice of sort> y <choice definition> se explica en 12.1.9.3.

Modelo

Una definición de tipo de datos que contenga una <choice definition> es sintaxis derivada y se transforma en los pasos siguientes: sea *Choice-name* el <data type name> de la definición de tipo de datos originales, entonces:

a) Se añade un <value data type definition> con un nombre anónimo, *anon*, y una <structure definition> como el constructivo tipo. En la <value data type definition>, para cada <choice of sort>, se construye un <field> que contiene el <fields of sort> equivalente con la palabra clave **optional**.

b) Se añade una <value data type definition> con un nombre anónimo, *anonPresent*, con una <literal list> que contiene todos los <field name>s de la <choice list> como <literal name>s. El orden de los literales es el mismo que el orden en el que se han especificado los <field name>.

c) Se construye una <data type definition> con un nombre anónimo, *anonChoice*, de la forma siguiente:

```
object type anonChoice
struct
        protected Present anonPresent;
        protected Choice anon;
endobject type anonChoice;
```

si la definición de tipos de datos originales había definido un género objeto. En cualquier otro caso, la <data type definition> es una <value data type definition>.

d) Se construye una <data type definition> tal como sigue:

```
object type Choice-name inherits anonChoice (anonMake = Make,
                        anonPresentModify = PresentModify,
                        anonPresentExtract = PresentExtract,
                        anonChoiceModify = ChoiceModify,
                        anonChoiceExtract = ChoiceExtract )
adding
        operations
endobject type Choice-name;
```

si la definición de tipos de datos original había definido un tipo de objeto, y donde las *operations* son <operations>, tal como se definen a continuación. Si no es así, la <data type definition> es una <value data type definition>. El <renaming> redenomina las operaciones mencionadas heredadas *anonChoice* como nombres anónimos.

e) Para cada <choice of sort>, se añade una <operation signature> a la <operator list> de *operations* lo cual representa un operador supuesto para la creación de ítems de datos:

```
field-name ( field-sort ) -> Choice-name;
```

donde *field-name* es el <field name> y *field-sort* es el <field sort> en <choice sort>. El operador implícito para la creación de ítems de datos crea una nueva estructura llamando a *anonMake*, inicializando el campo Choice con una estructura recién creada inicializada con <field name>, y asignando el literal correspondiente a <field name> al campo Present.

f) Para cada <choice of sort>, se añaden <operation signature>s a la <method list> de *operations*, representando métodos implícitos para la modificación y acceso a los ítems de datos:

virtual *field-modify* (*field-sort*) -> *Choice-name*;
virtual *field-extract* -> *field-sort*;
field-present -> Boolean;

donde *field-extract* es el nombre del método implícito por *anon* para acceder al correspondiente campo, *field-modify* es el nombre del método implícito por *anon* para modificar dicho campo, y *field-present* es el nombre del método implícito por *anon* para verificar la presencia de un campo ítem de datos. Las llamadas a *field-extract* y *field-present* se dirigen a Choice. Las llamadas a *field-modify* asignan a Choice una estructura recién creada inicializada con <field name> y asignan a Present el literal correspondiente a <field name>.

g) Se añade una <operation signature> a la <operator list> de *operations* representando un operador implícito para la obtención del género del ítem de datos presente en Choice:

PresentExtract (*Choice-name*) -> *anonPresent*;

PresentExtract devuelve el valor asociado con el campo Present.

12.1.8 Comportamiento de las operaciones

Una <data type definition> permite que se añadan operaciones a un tipo de datos. El comportamiento de las operaciones puede definirse de forma similar a un valor que devuelve llamadas a procedimientos. Sin embargo, las operaciones de un tipo de datos no deben acceder o modificar el estado global de las colas de entrada de los agentes en los que son llamadas. Por lo tanto, sólo contienen una transición sencilla.

Gramática textual concreta

```
<operation definitions> ::=
                        {       <operation definition>
                        |       <textual operation reference>
                        |       <external operation definition> }*

<operation definition> ::=
                        {<package use clause>}*
                        <operation heading> <end>
                                <entity in operation>*
                                <operation body>
                        { endoperator | endmethod } [ [<qualifier>] <operation name>] <end>
                    |   {<package use clause>}*
                        <operation heading>
                                [ <end> <entity in operation>+ ] <left curly bracket>
                                        <statement list>
                                 <right curly bracket>

<operation heading> ::=
                        { operator | method } <operation preamble> [<qualifier>] <operation name>
                                [<formal operation parameters>]
                                [<operation result>] [<raises>]

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

<formal operation parameters> ::=
                        ( <operation parameters> {, <operation parameters> }* )

<operation parameters> ::=
                        [<argument virtuality>] <parameter kind> <variable name> {, <variable name>}* <sort>
```

<entity in operation> ::=
<data definition>
| <variable definition>
| <exception definition>
| <select definition>
| <macro definition>

<operation body> ::=
[<on exception>] <start> { <free action> | <exception handler> }*

<operation result> ::=
<result sign> [<variable name>] <sort>

<textual operation reference> ::=
<operation heading> **referenced** <end>

<external operation definition> ::=
<operation heading> **external** <end>

Las palabras clave **operator** o **method** utilizadas en <operation heading> deben emparejarse con **endoperator** y **endmethod**, respectivamente, en la correspondiente <operation definition>.

No existe sintaxis gráfica correspondiente para <external operation definition>.

<formal operation parameters> y <operation result> en <textual operation reference> y <external operation definition> pueden omitirse si dentro del mismo género no existe otra <textual operation reference> ni <external operation definition> respectivamente que tengan el mismo nombre. En este caso, <formal operation parameters> y <operation result> se derivan de la <operation signature>.

Para cada <operation signature> puede darse como máximo una correspondiente <operation definition>.

Ni el <operation body> ni los <statement>s de la <operation definition> pueden contener una <imperative expression> ni un <identifier> definido fuera de la <operation definition> o el <operation diagram> circundante, respectivamente, excepto para <synonym identifier>s, <operation identifier>s, <literal identifier>s y <sort>s.

Si en una operación puede generarse una excepción cuando no hay activo ningún manejador de excepciones con la correspondiente cláusula de excepción, (es decir, no es manejada), <raises> debe mencionar esa excepción. Se considera que en una operación no se maneja una excepción si existe un potencial flujo de control dentro de la operación que produce dicha excepción y ninguno de los manejadores de excepciones activados en ese flujo de control maneja dicha excepción.

Se considera que la lista de <variable name>s vincula más estrechamente que la lista de <operation parameters> dentro de <formal operation parameters>.

Gramática gráfica concreta

<operation diagram> ::=
<frame symbol> ***contains*** {
<operation heading>
{ <operation text area>* <operation graph area> }***set*** }
[***is associated with*** <package use area>]

<operation graph area> ::=
[<on exception association area>] <procedure start area>
{ <in connector area> | <exception handler area> }*

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

La <package use area> debe estar situada encima del <frame symbol>.

El <start> en <operation diagram> no debe contener <virtuality>.

Debido a que no hay gramática gráfica para definiciones de género, un <operation diagram> sólo puede utilizarse en una <textual operation reference>.

Para cada <operation diagram> debe existir una <operation signature> en la misma unidad de ámbito que tenga el mismo <operation name>, que tenga los <sort>s de argumento en las posiciones especificadas en <formal operation parameters> (si existe) y que tenga el mismo <sort> de resultado especificado en <operation result> (si existe). Si sólo hay una operación que tenga el <operation identifier> o el <operation name> dado, pueden omitirse los <sort>s de argumento en <formal operation parameters>. Igualmente, <sort> puede omitirse en <operation result>.

Para cada <operation signature> puede darse como máximo un <operation diagram> correspondiente.

El <operation body> y los <statement>s de la <operation definition> no pueden contener una <imperative expression> ni un <identifier> definido fuera de la <operation definition> o el <operation diagram> circundante, respectivamente, excepto para <synonym identifier>s, <operation identifier>s, <literal identifier>s y <sort>s.

Semántica

Un operador es un contructivo para elementos del género que identifica el resultado. Siempre debe devolver un valor o un objeto recién construido. Por contra, un método puede devolver un objeto existente.

Un operador no debe modificar objetos que sean alcanzables por referencias de los parámetros reales o por los propios parámetros reales. Se considera que un objeto es modificado en un operador si existe un potencial flujo de control dentro del operador que da lugar a dicha modificación.

Una definición de operación es una unidad de ámbito que define sus propios datos y variables que pueden ser manipulados dentro del <operation body>.

Si el <operation heading> comienza con la palabra clave **operator**, la <operation definition> define el comportamiento de un operador. Si el <operation heading> comienza con la palabra clave **method**, la <operation definition> define el comportamiento de un método.

Las variables que se introducen en <formal operation parameters> son variables locales del operador o método y pueden modificarse dentro del <operation body>.

Una <external operation definition> es un operador o método cuyo comportamiento no está incluido en la descripción SDL. Conceptualmente, se asume que una <external operation definition> recibe un comportamiento y se transforma en una <operation definition> como parte de la transformación de sistema genérica (véase el anexo F).

Modelo

Para toda <operation definition> que no tiene una correspondiente <operation signature> se construye una <operation signature>.

Una <operation definition> se transforma en una <procedure definition> o en un <procedure diagram> respectivamente, con nombre anónimo, teniendo <procedure formal parameters> derivados de los <formal operation parameters> y con un <result> derivado del <operation result>. El <procedure body> se deriva de <operation body> si hay uno presente, o bien, si la <operation definition> contiene una <statement list>, el resultado de esta transformación es una <procedure definition> (véase 9.4). Después de aplicar el *Modelo* de <procedure definition>, el arranque virtual que inserta dicho *Modelo* es sustituido por un arranque sin <virtuality>.

Un <operation diagram> se transforma en un <procedure diagram> de forma similar.

La *Procedure-definition* correspondiente a la <procedure definition>, o al <procedure diagram> resultante se asocia a la *Operation-signature* representada por la <operation signature>.

Si la <operation definition>, o el <operation diagram> definen un método, se inserta un parámetro inicial con <parameter kind> **in/out** en <formal operation parameters> durante la transformación en una <procedure definition> o <procedure diagram>, siendo el <argument sort> el género que define la <data type definition> que constituye la unidad de ámbito en la que tiene lugar la <operation definition>. El <variable name> en <formal operation parameters> para este parámetro insertado es un nombre anónimo recientemente formado.

NOTA – No es posible especificar una <operation definition> para una <literal signature>.

Si una <operation definition> contiene texto informal, la interpretación de expresiones que impliquen la aplicación del correspondiente operador o método no se define formalmente mediante SDL, pero por el intérprete puede determinarlo a partir del texto informal. Si se especifica texto informal, no se da una especificación formal completa en SDL.

12.1.9 Constructivos adicionales de definición de datos

Esta subcláusula introduce constructivos adicionales que pueden utilizarse para datos.

12.1.9.1 Clase de nombre

Una clase de nombre es una notación taquigráfica para escribir un conjunto (posiblemente infinito) de nombres de literales o nombres de operadores definidos por una expresión regular.

Un <name class literal> es una forma alternativa de especificar un <literal name>. Una <name class operation> es una forma alternativa de especificar un <operation name> de una operación de nulidad.

Gramática textual concreta

```
<name class literal> ::=
                        nameclass <regular expression>

<name class operation> ::=
                        <operation name> in nameclass <regular expression>

<regular expression> ::=
                        <partial regular expression> { [or ] <partial regular expression> }*

<partial regular expression> ::=
                        <regular element> [ <Natural literal name> | <plus sign> | <asterisk> ]

<regular element> ::=
                        ( <regular expression> )
                    |   <character string>
                    |   <regular interval>

<regular interval> ::=
                        <character string> <colon> <character string>
```

Los nombres formados por la <regular expression> deben cumplir las reglas de léxico para nombres, o bien, el <character string>, <hex string> o <bit string> (véase 6.1).

Las <character string>s de un <regular interval> deben tener ambas una longitud de uno, excluidos los <apostrophe>s anteriores y posteriores.

Una <name class operation> sólo puede ser utilizada en una <operation signature>. Una <operation signature> que contenga <name class operation> sólo debe ocurrir en una <operator list> y no debe contener <arguments>.

Cuando un nombre contenido en el conjunto equivalente de nombres de una <name class operation> ocurre como <operation name> en una <operation application>, no debe tener <actual parameters>.

El conjunto equivalente de nombres de una clase de nombres se define como el conjunto de nombres que satisfacen la sintaxis especificada por la <regular expression>. Los conjuntos equivalentes de nombres para las <regular expression>s contenidos en una <data type definition> no deben solaparse.

Modelo

Un <name class literal> es equivalente a este conjunto de nombres en la sintaxis abstracta. Cuando se utiliza una <name class operation> en una <operation signature>, se crea un conjunto de <operation signature>s sustituyendo cada nombre en el conjunto equivalente de nombres para la <name class operation> en la <operation signature>.

Una <regular expression> que sea una lista de <partial regular expression>s sin un **or** determina 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 una partícula **or** entre dos <partial regular expression>s, los nombres se forman a partir de la primera o segunda de esas <partial regular expression>s. **or** vincula más estrechamente que un simple secuenciamiento.

Si un <regular element> viene seguido de <Natural literal name>, la <partial regular expression> es equivalente al <regular element> repitiéndose el número de veces especificado por el <Natural literal name>.

Si un <regular element> es seguido por '*', la <partial regular expression> es equivalente a <regular element> que puede repetirse cero o más veces.

Si un <regular element> es seguido por <plus sign> la <partial regular expression> equivale al <regular element> repetido una o más veces.

Un <regular element> que sea una <regular expression> entre corchetes define las secuencias de caracteres definidas por la <regular expression>.

Un <regular element> que sea una <character string> define la secuencia de caracteres dada en la cadena de caracteres (omitiendo las comillas).

Un <regular element> que sea un <regular interval> define todos los caracteres especificados por <regular interval> como secuencias de caracteres alternativos. Los caracteres definidos por <regular interval> son todos los caracteres mayores o iguales que el primer carácter y menores o iguales que el segundo carácter conforme a la definición del género carácter (Character) (véase D.2).

Los nombres generados por un <name class literal> se definen en orden alfabético conforme al ordenamiento del género carácter. Se considera que los caracteres son sensibles a la escritura en mayúscula/minúscula, y que un prefijo verdadero de una palabra es menor que la palabra completa.

NOTA – En el anexo D figuran ejemplos.

12.1.9.2 Correspondencia de clase de nombre

Una correspondencia de clase de nombre es una notación taquigráfica utilizada para definir un número (posiblemente infinito) de definiciones de operaciones que abarcan todos los nombres de una <name class operation>. Una correspondencia de clase de nombre permite definir el comportamiento para los operadores y métodos definidos por una <name class operation>. Una correspondencia de clase de nombre tiene lugar cuando en <operation definitions> o en <operation diagram>s se utiliza un <operation name> que ha aparecido en una <name class operation> dentro de una <operation signature> de la <data type definition> circundante.

Un término de ortografía (*spelling*) de una correspondencia de clase de nombre se refiere a la cadena de caracteres que contiene la ortografía del nombre. Este mecanismo permite utilizar las operaciones Charstring para definir correspondencias de clase de nombre.

Gramática textual concreta

<spelling term> ::=

spelling (<operation name>)

Un <spelling term> es sintaxis concreta legal únicamente dentro de una <operation definition> o un <operation diagram> si ha tenido lugar una correspondencia de clase de nombre.

Modelo

Una correspondencia de clase de nombre es una notación taquigráfica para un conjunto de <operation definition>s o un conjunto de <operation diagram>s. El conjunto de <operation definition>s se deriva de una <operation definition> por sustitución de cada nombre del conjunto equivalente de nombres de la correspondiente <name class operation> para cada ocurrencia de <operation name> en la <operation definition>. El conjunto derivado de <operation definition>s contiene todas las posibles <operation definition> que pueden generarse de esta forma. El mismo procedimiento se sigue para obtener un conjunto de <operation diagram>s.

Las <operation definition>s y <operation diagram>s derivados se consideran legales incluso aunque un <string name> no esté permitido como <operation name> en la sintaxis concreta.

Las <operation definition> derivadas se añaden a <operation definitions> (si existe) en la misma <data type definition>. Las <operation diagram>s derivadas se añaden a la lista de diagramas en las que ha aparecido la <operation definition> original.

Si una <operation definition> o un <operation diagram> contiene uno o más <spelling term>s, cada <spelling term> se sustituye por una literal Charstring (véase D.3.4).

Si durante la transformación anterior el <operation name> del <spelling term> ha sido sustituido por un <operation name>, el <spelling term> constituye una notación taquigráfica para un Charstring derivado del <operation name>. El Charstring contiene la ortografía del <operation name>.

Si durante la transformación anterior el <operation name> del <spelling term> ha sido sustituido por un <string name>, el <spelling term> constituye una notación taquigráfica para un Charstring derivado del <string name>. El Charstring contiene la ortografía del <string name>.

12.1.9.3 Visibilidad restringida

Gramática textual concreta

<visibility> ::=

public | **protected** | **private**

<visibility> no debe preceder a una <literal list>, <structure definition>, o <choice definition> en una <data type definition> que contenga <data type specialization>. <visibility> no debe utilizarse en una <operation signature> que redefina una signatura de operación heredada.

Semántica

<visibility> controla la visibilidad de un nombre de literal o de un nombre de operación.

Cuando una <literal list> es precedida por una <visibility>, esta <visibility> se aplica a todas las <literal signature>s. Cuando una <structure definition> o <choice definition> es precedida por una <visibility>, esta <visibility> se aplica a todas las <operation signatures> implícitas.

Cuando <fields of sort> o <choice of sort> está precedido de una <visibility>, esta <visibility> se aplica a todas las <operation signatures> implícitas.

Si una <literal signature> o una <operation signature> contiene la palabra clave **private** en <visibility>, el *Operation-name* derivado de esta <operation signature> sólo es visible en el ámbito de la <data type definition> que contiene la <operation signature>. Cuando una <data type definition> que contiene la <operation signature> está especializada, el <operation name> en <operation signature> es redenominado implícitamente con un nombre anónimo. Cada ocurrencia de

esta <operation name> dentro de la <operation definitions> o de los <operation diagram>s correspondientes a dicha <operation signature> se redenomina con el mismo nombre anónimo cuando la <operation signature> y la correspondiente definición de operación se heredan por especialización.

NOTA 1 – En consecuencia, el operador o método definido por esta <operation signature> sólo puede utilizarse en aplicaciones de operaciones dentro de la definición de tipo de datos que originalmente se ha definido en esta <operation signature>, pero no en cualquier subtipo.

Si una <literal signature> o una <operation signature> contiene la palabra clave **protected** en <visibility>, el *Operation-name* derivado de esta <operation signature> sólo es visible dentro del ámbito de la <data type definition> que contienen la <operation signature>.

NOTA 2 – Dado que los operadores y métodos heredados se copian en el cuerpo del subtipo, el operador o método que define esta <operation signature> puede ser accedido dentro del ámbito de cualquier <data type definition> que sea un subtipo de la <data type definition> que originalmente definió dicha <operation signature>.

NOTA 3 – Si <literal signature> o una <operation signature> no contienen <visibility>, el *Operation-name* derivado de esta <operation signature> es visible donde también sea visible el <sort name> definido por la <data type definition> circundante.

Modelo

Si una <literal signature> o una <operation signature> contiene la palabra clave **public** en <visibility>, es sintaxis derivada para una signatura sin protección.

12.1.9.4 Sintipos

Un sintipo especifica un conjunto de elementos de un género. Un sintipo utilizado como género tiene la misma semántica que el género referenciado por el sintipo, salvo verificaciones de que unos ítems de datos pertenezcan al subconjunto especificado de elementos 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> ::=
                    {<package use clause>}*
                    syntype <syntype name> <equals sign> <parent sort identifier>
                        [<default initialization>] [<constraint> <end>]
                        [<syntype closing> <end> ]
                |   {<package use clause>}*
                |   <type preamble> <data type heading> [<data type specialization>]
                    {   <constraint> <end>
                      |  <end> <data type definition body> <constraint> <end> <data type closing> <end>
                      |  <left curly bracket> <data type definition body> <constraint> <end>
                            <right curly bracket> }

<syntype closing> ::=
                    endsyntype [<syntype name>]

<parent sort identifier> ::=
                    <sort>
```

Un <syntype> es una alternativa para un <sort>.

Una <syntype definition> con las palabras clave **value type** u **object type** 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 <sort> en <arguments> cuando se define una operación, el género de los correspondientes *Formal-argument*s es el *Parent-sort-identifier* del sintipo.

Cuando un <syntype identifier> se utiliza como resultado de una operación, el género del *Result* es el *Parent-sort-identifier* del sintipo.

Cuando un <syntype identifier> se utiliza como calificador para un nombre, el *Qualifier* es el *Parent-sort-identifier* del sintipo.

Si se usa la palabra clave **syntype** y se omite la <constraint>, los <syntype identifier>s para el sintipo están en la gramática abstracta representada como *Parent-sort-identifier*.

Semántica

Una definición de sintipo define un sintipo que hace referencia a un identificador de género y una constricción. Especificar un identificador de sintipo es lo mismo que especificar el identificador de género progenitor del sintipo, a excepción de los casos siguientes:

a) la asignación a una variable declarada con un sintipo (véase 12.3.3);

b) la salida de una señal si uno de los géneros especificados para la señal es un sintipo (véanse 10.3 y 11.13.4);

c) la llamada a un procedimiento cuando uno de los géneros especificados para el procedimiento en variables parámetro del procedimiento es un sintipo (véanse 9.4 y 11.13.3);

d) la creación de un agente cuando uno de los géneros especificados para los parámetros del proceso es un sintipo (véanse 9.3 y 11.13.2);

e) la entrada de una señal y una de las variables asociada a la entrada tiene un género que es un sintipo (véase 11.2.2);

f) la llamada a una aplicación de operación que tiene un sintipo definido como un género argumento o como un género resultado (véase 12.2.7);

g) la cláusula o la expresión activa de inicializar o reinicializar un temporizador y uno de los géneros en la definición de temporizador es un sintipo (véanse 11.15 y 12.3.4.4);

h) la definición de variable remota o de procedimiento remoto si uno de los géneros para la obtención de señales implícitas es un sintipo (véanse 10.5 y 10.6);

i) el parámetro de contexto formal de procedimiento con un parámetro in/out o out en <procedure signature> se corresponde con un parámetro de contexto real en el cual el parámetro formal correspondiente o el parámetro in/out o out en <procedure signature> es un sintipo;

j) <any expression>, en la cual el resultado está dentro del intervalo (véase 12.3.4.5).

k) la generación de una excepción si uno de los géneros especificados para la excepción es un sintipo (véase 11.12.2.5).

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 progenitor dado en la definición de sintipo.

Un sintipo tiene una *Range-condition* que constriñe al género. Si se utiliza una condición de intervalo, el género está constreñido al conjunto de ítems de datos especificados por las constantes de la definición de sintipo. Si se utiliza una constricción de tamaño, el género está constreñido a los ítems de datos de contener dados por la constricción de tamaño.

Modelo

Una <syntype definition> con las palabras clave **value type** u **object type** puede distinguirse de una <data type definition> por la inclusión de una <constrain>. Dicha <syntype definition> es una notación taquigráfica para introducir una <data type definition> con un nombre anónimo seguido de una <syntype definition> con la palabra clave **syntype** basada en este género con denominación anónima y que incluye <constraint>.

12.1.9.5 Condición de intervalo

Gramática abstracta

Range-condition	::	*Condition-item-**set***
Condition-item	=	*Open-range* | *Closed-range*
Open-range	::	*Operation-identifier*
		Constant-expression
Closed-range	::	*Open-range*
		Open-range

Gramática textual concreta

```
<constraint> ::=
                        constants ( <range condition> )
                    |   <size constraint>

<range condition> ::=
                        <range> { , <range> }*

<range> ::=
                        <closed range>
                    |   <open range>

<closed range> ::=
                        <constant> <colon> <constant>

<open range> ::=
                        <constant>
                    |   {   <equals sign>
                        |   <not equals sign>
                        |   <less than sign>
                        |   <greater than sign>
                        |   <less than or equals sign>
                        |   <greater than or equals sign> } <constant>

<size constraint> ::=
                        size ( <range condition> )

<constant> ::=
                        <constant expression>
```

El símbolo "<" sólo debe utilizarse en la sintaxis concreta de la <range condition> si dicho símbolo se ha definido con una <operation signature>:

```
"<"  ( P, P ) -> <<package Predefined>>Boolean;
```

donde P es el género del sintipo e igualmente ocurre para los símbolos ("<=", ">", ">=", respectivamente). Estos símbolos representan un *Operation-identifier*.

Un <closed range> sólo debe utilizarse si el símbolo "<=" está definido con una <operation signature>:

"<="(P, P) -> <<**package** Predefined>>Boolean;

donde P es el género del sintipo.

Una <constant expression> en una <range condition> tiene que tener el mismo género que el subtipo.

Una <size constraint> sólo debe utilizarse en la sintaxis concreta de la <range condition> si la longitud del símbolo se ha definido con una <operation signature>:

Length (P) -> <<**package** Predefined>>Natural;

donde P es el género del sintipo.

Semántica

Una constricción define una verificación de intervalo. Una verificación de intervalo se utiliza cuando un sintipo tiene semántica adicional al género del sintipo (véanse 12.3.1, 12.1.9.4 y los casos de sintipos con semántica diferente – véanse las subcláusulas a las que se hace referencia en los apartados a) a k) de 12.1.9.4, *Semántica*]. Una verificación de intervalo se utiliza también para determinar la interpretación de una decisión (véase 11.13.5).

La verificación de intervalo es la aplicación de la operación formada a partir de la condición de intervalo o constricción de tamaño. Para las verificaciones de intervalo de sintipo, la aplicación de este operador tiene que ser equivalente al valor booleano predefinido verdadero; de no ser así, se genera la excepción predefinida OutOfRange (véase D.3.16). La verificación de intervalo se deriva como sigue:

a) Cada <open range> o <closed range> en la <range condition> tiene un *Open-range* (booleano predefinido **or**) o *Closed-range* (booleano predefinido **and**) 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 una expresión dada, A, entonces

   1) un <open range> de la forma = <constant>, /= <constant>, < <constant>, <less than or equals sign> <constant>, > <constant> y <greater than or equals sign> <constant>, tiene una subexpresión en la verificación de intervalo de la forma A = <constant>, A /= <constant>, A < <constant>, A <less than or equals sign> <constant>, A > <constant> y A <greater than or equals sign> <constant> respectivamente;

   2) un <closed range> de la forma *first* <constant> : *second* <constant> tiene una subexpresión en la verificación de intervalo de la forma *first* <constant> <less than or equals sign> A **and** A <less than or equals sign> *second* <constant> donde **and** corresponde al booleano predefinido **and**;

   3) una <size constraint> tiene subexpresión en la verificación de intervalo de la forma Length(A) = <range condition>.

d) Hay un booleano predefinido **or** para la operación distribuida entre todos los ítems de datos del *Condition-item*-**set**. La verificación de intervalo es la expresión formada a partir del booleano predefinido **or** de todos los subtérminos derivados de la <range condition>.

Si un sintipo se especifica sin una <constraint>, la verificación de intervalo es el valor booleano predefinido verdadero.

12.1.9.6 Definición de sinónimo

Un sinónimo da un nombre a una expresión constante que representa uno de los ítems de datos de un género.

Gramática textual concreta

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

<synonym definition item> ::=
<internal synonym definition item>
| <external synonym definition item>

<internal synonym definition item> ::=
<synonym name> [<sort>] <equals sign> <constant expression>

<external synonym definition item> ::=
<synonym name> <predefined sort> = **external**

La <constant_expression> en la sintaxis concreta denota una *Constant-expression* en la sintaxis abstracta, tal como se define en 12.2.1.

Si se especifica un <sort>, el resultado de la <constant expression> tiene un género estático de <sort>. <constant expression> puede tener dicho género.

Si el género de <constant_expression> no puede determinarse de manera inequívoca, el género debe especificarse en la <synonym definition>.

La <constant_expression> no referirá al sinónimo definido por la <synonym definition>, directa ni indirectamente (por otro sinónimo).

Un <external synonym definition item> define un <synonym> cuyo resultado no está definido en una especificación. Conceptualmente, se supone que un <external synonym definition item> recibe un resultado y se transforma en un <internal synonym definition item> como parte de la transformación de sistema genérica (véase anexo F).

Semántica

El resultado que representa el sinónimo está determinado por el contexto en el que aparece la definición de sinónimo.

Si el género de la expresión constante no puede determinarse de manera inequívoca en el contexto del sinónimo, el género viene dado por el <sort>.

Un sinónimo tiene un resultado que es el de la expresión constante en la definición de sinónimo.

Un sinónimo tiene un género que es el de la expresión constante en la definición de sinónimo.

12.2 Utilización pasiva de los datos

En las subcláusulas siguientes se define cómo se interpretan en las expresiones los géneros, literales, operadores, métodos y sinónimos.

12.2.1 Expresiones

Gramática abstracta

Expression	=	*Constant-expression*
	|	*Active-expression*
Constant-expression	=	*Literal*
	|	*Conditional-expression*
	|	*Equality-expression*
	|	*Operation-application*
	|	*Range-check-expression*

Active-expression = *Variable-access*
| *Conditional-expression*
| *Operation-application*
| *Equality-expression*
| *Imperative-expression*
| *Range-check-expression*
| *Value-returning-call-node*

Gramática textual concreta

Por simplicidad de la descripción no se hace distinción entre la sintaxis concreta de *Constant-expression* y de *Active-expression.*

```
<expression> ::=
                    <operand>
                |   <create expression>
                |   <value returning procedure call>

<operand> ::=
                    <operand0>
                |   <operand> <implies sign> <operand0>

<operand0> ::=
                    <operand1>
                |   <operand0> { or | xor } <operand1>

<operand1> ::=
                    <operand2>
                |   <operand1> and <operand2>

<operand2> ::=
                    <operand3>
                |   <operand2> { <greater than sign>
                                 | <greater than or equals sign>
                                 | <less than sign>
                                 | <less than or equals sign>
                                 | in } <operand3>
                |   <range check expression>
                |   <equality expression>

<operand3> ::=
                    <operand4>
                |   <operand3> { <plus sign> | <hyphen> | <concatenation sign> } <operand4>

<operand4> ::=
                    <operand5>
                |   <operand4> { <asterisk> | <solidus> | mod | rem } <operand5>

<operand5> ::=
                    [ <hyphen> | not ] <primary>

<primary> ::=
                    <operation application>
                |   <literal>
                |   ( <expression> )
                |   <conditional expression>
                |   <spelling term>
                |   <extended primary>
                |   <active primary>
                |   <synonym>

<active primary> ::=
                    <variable access>
                |   <imperative expression>

<expression list> ::=
                    <expression> { , <expression> }*
```

<simple expression> ::=
<constant expression>

<constant expression> ::=
<constant expression>

Una <expression> que no contenga ningún <active primary>, una <create expression>, o un <value returning procedure call> con un <remote procedure call body> es una <constant expression>. Una <constant expression> representa una *Constant-expression* en la sintaxis abstracta. Cada <constant expression> se interpreta una sola vez durante la inicialización del sistema, y se preserva el resultado de la interpretación. Cuando se necesita el valor de la <constant expression> durante la interpretación, se utiliza una copia completa de dicho valor calculado.

Una <expression> que no es es una <constant expression> representa una *Active-expression*.

Si una <expression> contiene un <extended primary>, éste se sustituye en el nivel de sintaxis concreta tal como se define en 12.2.4 antes de considerar la relación con la sintaxis abstracta.

<operand>, <operand1>, <operand2>, <operand3>, <operand4> y <operand5> ofrecen formas sintácticas especiales para los nombres de operador y de método. La sintaxis especial se utiliza, por ejemplo, de forma que las operaciones aritméticas y booleanas pueden tener su forma sintáctica habitual. Es decir, el usuario puede escribir "(1 + 1) = 2" en lugar de estar obligado a utilizar, por ejemplo, la expresión equal(add(1,1),2). Los géneros que son válidos para cada operación dependen de la definición del tipo de datos.

Una <simple expression> sólo debe contener literales, operadores y métodos definidos en el lote Predefined, tal como se define en el anexo D.

Semántica

Un operador o método infijo es una expresión que tiene la semántica normal de un operador o método pero con sintaxis de infijo o de prefijo entre comillas.

Un operador o método monádico en una expresión tiene la semántica normal de un operador o método pero con la sintaxis de prefijo o prefijo entre comillas.

Los operadores o métodos infijos tienen un orden de precedencia que determina la vinculación de los operadores o métodos. Cuando la vinculación es ambigua, se efectúa de izquierda a derecha.

Cuando se interpreta una expresión, ésta devuelve un ítem de datos (un valor, objeto o pid). Al ítem de datos devuelto se hace referencia como cl resultado de la expresión.

El género (estático) de una expresión es el del ítem de datos que devolvería la interpretación de la expresión tal como determina el análisis de la especificación sin considerar la semántica de la interpretación. El género dinámico de una expresión es el del resultado de la expresión. El género estático y dinámico de expresiones activas puede diferir debido a asignaciones polimórficas (véase 12.3.3). Para una expresión constante, el género dinámico de la misma coincide con su género estático.

NOTA – Para evitar textos recargados la palabra "género" siempre se refiere a género estático a menos que esté seguida de la palabra "dinámico". Para mayor claridad, en algunos casos se escribe explícitamente "género estático".

Una expresión simple es una *Constant-expression*.

Modelo

Una expresión de la forma:

<expression> <infix operation name> <expression>

es sintaxis derivada para:

<quotation mark> <infix operation name> <quotation mark> **(** <expression>**,** <expression> **)**

donde <quotation mark> <infix operation name> <quotation mark> representa un *Operation-name*.

Igualmente,

<monadic operation name> <expression>

es sintaxis derivada para:

<quotation mark> <monadic operation name> <quotation mark> **(** <expression> **)**

donde <quotation mark> <monadic operation name> <quotation mark> representa un *Operation-name*.

12.2.2 Literal

Gramática abstracta

Literal	::	*Literal-identifier*
Literal-identifier	=	*Identifier*

El *Literal-identifier* denota una *Literal-signature*.

Gramática textual concreta

<literal> ::=
<literal identifier>

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

Siempre que se especifica un <literal identifier>, el *Literal-name* único en *Literal-identifier* se deriva de la misma forma, derivando del contexto el género del resultado. Un *Literal-identifier* se deriva del contexto (véase 6.3) de forma que si el <literal identifier> está sobrecargado (es decir, se utiliza el mismo nombre para más de un literal u operación), entonces el *Literal-name* identifica un literal visible con el mismo nombre y género de resultado consistente con el literal. Dos literales con el mismo <name> pero que difieren en los géneros del resultado, tienen *Literal-names* distintos.

Debe poderse vincular cada <literal identifier> no calificado con exactamente un *Literal-identifier* definido que cumpla las condiciones del constructivo en el que se utiliza <literal identifier>.

Cuando un <qualifier> de un <literal identifier> contiene un <path item> con la palabra clave **type**, el <sort name> que sigue a dicha palabra clave no forma parte del *Qualifier* del *Literal-identifier*, pero, sin embargo, 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 preceden a la palabra clave **type**.

Semántica

Un *Literal* devuelve el ítem de datos único correspondiente a su *Literal-signature*.

El género del <literal> es el *Result* en su *Literal-signature*.

12.2.3 Sinónimo

Gramática textual concreta

<synonym> ::=
<synonym identifier>

Semántica

Un sinónimo es una notación taquigráfica para denotar una expresión definida en otro sitio.

Modelo

Un <synonym> representa la <constant expression> definida por la <synonym definition> identificada por el <synonym identifier>. Un <identifier> utilizado en la <constant expression> representa un *Identifier* en la sintaxis abstracta conforme al contexto de la <synonym definition>.

12.2.4 Primario ampliado

Un primario ampliado es una notación sintáctica taquigráfica; sin embargo, aparte de la forma sintáctica especial, un primario ampliado no tiene propiedades especiales y denota a una operación y a su parámetro o parámetros.

Gramática textual concreta

<extended primary> ::=
<indexed primary>
| <field primary>
| <composite primary>

<indexed primary> ::=
<primary> **(** <actual parameter list> **)**
| <primary> <left square bracket> <actual parameter list> <right square bracket>

<field primary> ::=
<primary> <exclamation mark> <field name>
| <primary> <full stop> <field name>
| <field name>

<field name> ::=
<name>

<composite primary> ::=
[<qualifier>] <composite begin sign> <actual parameter list> <composite end sign>

Modelo

Un <indexed primary> es sintaxis concreta derivada para:

<primary> <full stop> Extract **(** <actual parameter list> **)**

La sintaxis abstracta se determina a partir de esta expresión concreta conforme a 12.2.1.

Un <field primary> es sintaxis concreta derivada para:

<primary> <full stop> *field-extract-operation-name*

donde *field-extract-operation-name* se forma mediante la concatenación del nombre del campo y de "Extract" en dicho orden. La sintaxis abstracta se determina de esta expresión concreta conforme a 12.2.1. La transformación conforme a este modelo se realiza antes de la modificación de la signatura de métodos descrita en 12.1.4.

Cuando el <field primary> tiene la forma <field name>, es sintaxis derivada para:

this ! <field name>

Un <composite primary> es sintaxis concreta derivada para:

<qualifier> Make **(** <actual parameter list> **)**

si cualquier parámetro estuviera presente, o para:

<qualifier> Make

en cualquier otro caso, y cuando se inserta el <qualifier> únicamente si estaba presente en <composite primary>. La sintaxis abstracta se determina a partir de esta expresión concreta conforme a 12.2.1.

12.2.5 Expresión de igualdad

Gramática abstracta

Equality-expression	::	*First-operand*
		Second-operand
First-operand	=	*Expression*
Second-operand	=	*Expression*

Una *Equality-expression* representa la igualdad de referencias o de valores de su *First-operand* y de su *Second-operand*.

Gramática textual concreta

<equality expression> ::=
<operand2> { <equals sign> | <not equals sign> } <operand3>

Una <equality expression> es sintaxis concreta legal únicamente si el género de uno de sus operandos es compatible con el del otro operando.

Semántica

La interpretación de la *Equality-expression* se realiza mediante la interpretación de su *First-operand* y de su *Second-operand*.

Si después de la interpretación ambos operandos son objetos, la *Equality-expression* denota igualdad de referencia. Devuelve el valor booleano predefinido verdadero si y sólo si ambos operandos son Null o hacen referencia al mismo ítem de datos objeto.

Si después de la interpretación ambos operandos son pids, la *Equality-expression* denota identidad de agente. Devuelve el valor booleano predefinido verdadero si y sólo si ambos operandos son Null o hacen referencia al mismo ejemplar de agente.

Si después de la interpretación uno de los operandos es un valor, la *Equality-expression* denota igualdad de valores como sigue:

a) Si el género dinámico de *First-operand* es compatible con el género dinámico de *Second-operand*, la <equality expression> devuelve el resultado de la aplicación del operador igual (equal) a *First-operand* y *Second-operand*, donde igual es el *Operation-identifier* representado por <operation identifier> en la <operation application>:

equal(<operand2>, <operand3>)

b) En otro caso, la <equality expression> devuelve el resultado de la aplicación del operador igual a *Second-operand* y *First-operand*, donde igual es el *Operation-identifier* representado por el <operation identifier> en la <operation application>:

equal(<operand3>, <operand2>)

La forma de sintaxis concreta:

<operand2> <not equals sign> <operand3>

es sintaxis concreta derivada para:

not (<operand2> = <operand3>)

donde **not** es una operación de tipo de datos predefinido booleano.

12.2.6 Expresión condicional

Gramática abstracta

Conditional-expression	::	*Boolean-expression*
		Consequence-expression
		Alternative-expression
Boolean-expression	=	*Expression*
Consequence-expression	=	*Expression*
Alternative-expression	=	*Expression*

Una *Conditional-expression* es una *Expression* que se interpreta como la *Consequence-expression* o como la *Alternative-expression*.

El género de *Consequence-expression* debe ser el mismo que el de *Alternative-expression*.

Gramática textual concreta

<conditional expression> ::=
if <Boolean expression>
then <consequence expression>
else <alternative expression>
fi

<consequence expression> ::=
<expression>

<alternative expression> ::=
<expression>

El género de <consequence expression> debe ser el mismo que el de <alternative expression>.

Semántica

Una expresión condicional representa una *Expression* que se interpreta como *Consequence-expression* o como *Alternative-expression*.

Si la *Boolean-expression* devuelve el valor booleano predefinido verdadero, entonces no se interpreta la *Alternative-expression*. Si la *Boolean-expression* devuelve el valor booleano predefinido falso, no se interpreta la *Consequence-expression*.

Una expresión condicional tiene un género que es el de la expresión consecuencia (e igualmente el género de la expresión alternativa).

El resultado de la expresión condicional es el resultado de interpretar *Consequence-expression* o *Alternative-expression*.

El género estático de una expresión condicional es el de *Consequence-expression* (que también es el género de *Alternative-expression*). El género dinámico de la expresión condicional es el género del resultado de interpretar la expresión condicional.

12.2.7 Aplicación de operación

Gramática abstracta

Operation-application	::	*Operation-identifier* [*Expression*]*
Operation-identifier	=	*Identifier*

El *Operation-identifier* denota una *Operation-signature*, ya sea una *Static-operation-signature* o una *Dynamic-operation-signature*. Cada *Expression* de la lista de *Expression*s posterior al *Operation-identifier* debe ser compatible con el correspondiente género (en posición) en la lista de *Formal-argument*s de la *Operation-signature*.

Cada *Operation-signature* tiene asociada una *Procedure-definition*, tal como se describe en 12.1.8.

Cada *Expression* que corresponde por posición a un *Inout-parameter* o *Out-parameter* en la *Procedure-definition* asociada con la *Operation-signature*, debe tener un *Variable-identifier* con el mismo *Sort-reference-identifier* que el correspondiente género (en posición) en la lista de *Formal-argument*s de la *Operation-signature*.

Gramática textual concreta

<operation application> ::=
<operator application>
| <method application>

<operator application> ::=
<operation identifier> [<actual parameters>]

<method application> ::=
<primary> <full stop> <operation identifier> [<actual parameters>]

Cuando se especifica un <operation identifier>, el *Operation-name* único en el *Operation-identifier* se deriva de la misma forma. La lista de géneros de argumentos se deriva de los parámetros reales y el género del resultado se deriva del contexto (véase 6.3). Por tanto, si <operation name> está sobrecargado (es decir, se utiliza el mismo nombre para más de un literal u operación), el *Operation-name* identifica una operación visible con el mismo nombre y con los géneros de argumentos y de resultados consistentes con la aplicación de la operación. Dos operaciones con el mismo <name> pero distintas en uno o más de los argumentos o géneros de resultados, tienen distintos *Operation-names*.

Debe poderse vincular cada <operation identifier> no calificado con exactamente un *Operation-identifier* definido que cumpla las condiciones del constructivo en el que se utiliza <literal identifier>.

Cuando la aplicación de la operación tiene la forma sintáctica:

<operation identifier> [<actual parameters>]

entonces, durante la derivación del *Operation-identifier* a partir del contexto, también se considera la forma:

this <full stop> <operation identifier> [<actual parameters>]

El modelo de 12.3.1 se aplica antes de intentar la resolución por contexto.

Cuando un <qualifier> de un <operation identifier> contiene un <path item> con la palabra clave **type**, el <sort name> posterior a dicha palabra clave no forma parte del *Qualifier* del *Operation-identifier*, pero, sin embargo, 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 preceden a la palabra clave **type**.

Si todas las <expression>s de la lista entre paréntesis de <expression>s son <constant expression>s, la <operation application> representa una *Constant-expression* tal como se define en 12.2.1.

Una <method application> es sintaxis concreta legal únicamente si <operation identifier> representa un método.

Una <expression> en <actual parameters> que corresponda a un *Inout-parameter* o *Out-parameter* en la *Procedure-definition* asociada con la *Operation-signature* no puede omitirse y debe ser un <variable access> o <extended primary>.

Si se utiliza **this**, <operation identifier> tiene que denotar un operador o método circundante.

NOTA – <actual parameters> puede omitirse en una <operation application> si todos los parámetros reales han sido omitidos.

Semántica

La resolución por contexto (véase 6.3) garantiza que se selecciona una operación tal que los tipos de los argumentos reales son compatibles en género por parejas con los tipos de los argumentos formales.

Una aplicación de operación con un *Operation-identifier* que denota una *Static-operation-signature* se interpreta transfiriendo la interpretación a la *Procedure-definition* asociada con la *Operation-signature*, interpretándose dicho gráfico de procedimiento (la explicación se encuentra en 9.4).

Una aplicación de operación con un *Operation-identifier* que denota una *Dynamic-operation-signature*, se interpreta mediante los pasos siguientes:

a) se interpretan los parámetros reales;

b) si el resultado de un parámetro real correspondiente a un *Virtual-argument* es Null, se genera la excepción predefinida InvalidReference;

c) se recopilan en un conjunto todas las *Dynamic-operation-signatures*, de manera que la signatura de operación formada a partir de un *Operation-name* derivado del <operation name> en <operation identifier> y los géneros dinámicos del resultado de interpretar los parámetros reales son compatible en género con la *Dynamic-operation-signature* candidata;

d) se selecciona la *Dynamic-operation-signature* única que es compatible en género con todas las demás *Dynamic-operation-signatures* de ese conjunto; y

e) la interpretación se transfiere a la *Procedure-definition* asociada con la *Operation-signature* seleccionada y se interpreta el gráfico del procedimiento (la explicación se encuentra en 9.4).

La existencia de dicha signatura única está garantizada por el requisito de que el conjunto de *Dynamic-operation-signatures* formen una retícula (véase 12.1.2).

La lista de parámetros efectivos *Expression*s en una *Operation-application* se interpretan en el orden dado de izquierda a derecha antes de que se interprete la propia operación.

Si se omite una <expression> en <actual parameters>, el correspondiente parámetro formal no tiene ningún ítem de datos asociado, es decir es "indefinido".

Si un género de argumento de la signatura de operación es un sintipo, se aplica al resultado de la *Expression* la verificación de intervalo definida en 12.1.9.5. Si la verificación de intervalo da como resultado el valor booleano predefinido falso cuando se realiza la interpretación, se genera la excepción predefinida OutOfRange (véase D.3.16).

La interpretación de la transición que contiene la <operation application> continua cuando termina la interpretación del procedimiento llamado. El resultado de la aplicación de operación es el resultado que devuelve la interpretación de la definición del procedimiento referenciado.

Si el género del resultado de la signatura de operación es un sintipo, se aplica al resultado de la aplicación de operación la verificación de intervalo definida en 12.1.9.5. Si la verificación de intervalo da como resultado el valor booleano predefinido falso cuando se realiza la interpretación, se genera la excepción predefinida OutOfRange (véase D.3.16).

Una <operation application> tiene un género que coincide con el del resultado obtenido de la interpretación del procedimiento asociado.

Modelo

La forma de sintaxis concreta:

<expression> <full stop> <operation identifier> [<actual parameters>]

es sintaxis derivada para:

<operation identifier> *new-actual-parameters*

donde *new-actual-parameters* es <actual parameters> que sólo contiene <expression> si no está presente <actual parameters>; en cualquier otro caso, *new-actual-parameters* se obtiene insertando <expression> antes de la primera expresión opcional en <actual parameters>.

12.2.8 Expresión de verificación de intervalo

Gramática abstracta

Range-check-expression :: *Range-condition Expression*

Gramática textual concreta

<range check expression> ::=
<operand2> **in type** { <sort identifier> <constraint> | <sort> }

El género de <operand2> debe ser el mismo que el identificado por <sort identifier> o <sort>.

Semántica

Una *Range-Check-Expression* es una expresión del género booleano predefinido que da como resultado verdadero si el resultado de la *Expression* cumple la *Range-condition* que corresponde a <constraint> tal como se define en 12.1.9.5; en otro caso, su resultado es falso.

Modelo

La especificación de un <sort> es sintaxis derivada para la especificación de la <constraint> del tipo de datos que ha definido el <sort>. Si dicho tipo de datos no se ha definido con una <constraint>, no se evalúa la <range check expression> y la <range check expression> es sintaxis derivada para la especificación del valor predefinido booleano verdadero.

12.3 Utilización activa de datos

En esta subcláusula se define la utilización de datos y de variables declaradas, cómo se interpreta una expresión que implique variables y las expresiones imperativas que obtienen resultados del sistema subyacente.

Una variable tiene un género y un ítem de datos asociado de dicho género. El ítem de datos asociado con una variable puede cambiarse asignando un nuevo ítem de datos a la variable. El ítem de datos asociado con la variable puede utilizarse en una expresión accediendo a la variable.

Toda expresión que contenga una variable se considera que está "activa" ya que los ítems de datos obtenidos interpretando la expresión pueden variar de acuerdo con el último ítem de datos asignado a la variable. El resultado de la interpretación de una expresión activa dependerá del estado vigente del sistema.

12.3.1 Definición de variable

Una variable tiene asociado un ítem de datos o bien es "indefinida".

Gramática abstracta

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

Si está presente la *Constant-expression*, debe ser del mismo género que denota el *Sort-reference-identifier*.

Gramática textual concreta

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

<variables of sort> ::=
<variable name> [<exported as>] { **,** <variable name> [<exported as>] }*****
<sort> [<is assigned sign> <constant expression>]

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

<exported as> sólo puede utilizarse para una variable que tenga **exported** en su <variable definition>. Dos variables exportadas en un agente no pueden mencionar el mismo <remote variable identifier>.

La *Constant-expression* se representa por lo siguiente:

a) si en la <variable definition> existe una <constant expression>, por dicha <constant expression>;

b) o bien, si el tipo de datos que define el <sort> tiene una <default initialization>, por la <constant expression> de la <default initialization>.

En otro caso, *Constant-expression* no existe.

Semántica

Cuando se crea una variable y existe la *Constant-expression*, la variable se asocia al resultado de la *Constant-expression*. En cualquier otro caso, la variable no tiene ítems de datos asociados, es decir, la variable es "indefinida".

Si *Sort-reference-identifier* es un *Syntype-identifier*, *Constant-expression* está presente y el resultado de *Constant-expression* no cumple la *Range-condition*, se genera la excepción predefinida OutOfRange (véase D.3.16).

La palabra clave **exported** permite utilizar una variable como variable exportada tal como se describe en 10.6.

12.3.2 Acceso a variable

Gramática abstracta

Variable-access = *Variable-identifier*

Gramática textual concreta

<variable access> ::=
<variable identifier>
| **this**

this sólo puede ocurrir en definiciones de método.

Semántica

Un acceso a variable se interpreta dando el ítem de datos asociado a la variable identificada.

Un acceso a variable tiene un género estático que coincide con el de la variable identificada por el propio acceso a variable. Tiene un género dinámico que es el del ítem de datos asociado a la variable identificada.

Un acceso a variable tiene un resultado que es el último ítem de datos asociado a la variable. Si la variable es "indefinida", se interpreta un Raise-node para la excepción predefinida UndefinedVariable (véase D.3.16).

Modelo

Un <variable access> que utilice la palabra clave **this** se sustituye por el nombre anónimo introducido como nombre del parámetro inicial en <arguments> conforme a 12.1.8.

12.3.3 Asignación e intento de asignación

Una asignación crea una asociación entre la variable y el resultado de la interpretación de una expresión. En un intento de asignación, esta asociación se crea únicamente si los géneros dinámicos de la variable y la expresión son compatibles.

Gramática abstracta

Assignment :: *Variable-identifier*
Expression

Assignment-attempt :: *Variable-identifier*
Expression

En una *Assignment*, el género de la *Expression* debe ser compatible en género con el de *Variable-identifier*.

En un *Assignment-attempt*, el género del *Variable-identifier* debe ser compatible en género con el de *Expression*.

Si se declara que la variable tiene un *Syntype* y la *Expression* es una *Constant-expression*, la verificación de intervalo definida en 12.1.9.5, aplicada a la *Expression,* debe ser el valor booleano predefinido verdadero.

Gramática textual concreta

<assignment> ::=
<variable> <is assigned sign> <expression>

<variable> ::=
<variable identifier>
| <extended variable>

Si la <variable> es un <variable identifier> la <expression> en la sintaxis concreta representa la *Expression* en la sintaxis abstracta. Una <extended variable> es sintaxis derivada y se sustituye a nivel de sintaxis concreta tal como se define en 12.3.3.1 antes de que se considere la relación con la sintaxis abstracta.

Si el <variable identifier> se ha declarado con un género de objetos y el género de la <expression> es un supergénero (directo o indirecto) del género del <variable identifier>, la <assignment> representa un *Assignment-attempt*. En cualquier otro caso, la <assignment> representa una *Assignment*.

Semántica

Una *Assignment* se interpreta creando una asociación entre la variable identificada en la asignación y el resultado de la expresión en la asignación. La asociación previa de la variable se pierde.

La manera en la que se establece esta asociación depende del género del <variable identifier> y del género de la <expression>:

a) Si el <variable identifier> tiene un género de valores, el resultado de la *Expression* se copia en el valor actualmente asociado con el *Variable-identifier* interpretando el método de *copia* definido por la definición de tipo de datos que introdujo el género del <variable identifier>, dando *Variable-identifier* y *Expression* como parámetros reales. Si la *Expression* es Null, se genera la excepción predefinida InvalidReference (véase D.3.16).

b) Si el <variable identifier> tiene un género de objetos y el resultado de la *Expression* es un objeto, el *Variable-identifier* se asocia con el objeto resultado de *Expression*. No está permitido que el género de la <expression> sea un sintipo que restrinja los elementos del género del <variable identifier>.

c) Si el <variable identifier> tiene un género de objetos y el resultado de la *Expression* es un valor, se construye un clon del resultado de *Expression* interpretando el operador *clone* definido por la definición de tipo de datos que introdujo el género del <variable identifier>, dando *Expression* como parámetro real. El *Variable-identifier* se asocia con una referencia al valor clonado. No está permitido que el género de la <expression> sea un sintipo que restrinja los elementos del género del <variable identifier>.

d) Si el <variable identifier> tiene un género de pid y el resultado de la *Expression* es un pid, el *Variable-identifier* se asocia con el pid que constituye el resultado de *Expression*.

Si la variable se declara con un sintipo, se aplica a la expresión la verificación de intervalo definida en 12.1.9.5. Si esta verificación de intervalo devuelve el valor booleano predefinido falso, se genera la excepción predefinida OutOfRange (véase D.3.16).

Cuando se interpreta un *Assignment-attempt*, y si el género dinámico de la *Expression* es compatible en género con el del *Variable-identifier*, se interpreta una Assignment que implique al *Variable-identifier* y a la *Expression*. En otro caso, el *Variable-identifier* se asocia con Null.

NOTA – Es posible determinar mediante un intento de asignación el género dinámico de una *Expression*.

12.3.3.1 Variable ampliada (o variable extendida)

Una variable ampliada (o variable extendida) es una notación sintáctica taquigráfica; sin embargo, además de la forma sintáctica especial, una variable ampliada no tiene propiedades especiales y denota una operación y sus parámetros.

Gramática textual concreta

<extended variable> ::=
<indexed variable>
| <field variable>

<indexed variable> ::=
<variable> **(** <actual parameter list> **)**
| <variable> <left square bracket> <actual parameter list> <right square bracket>

<field variable> ::=
<variable> <exclamation mark> <field name>
| <variable> <full stop> <field name>

Modelo

<indexed variable> es sintaxis concreta derivada para:

<variable> <is assigned sign> <variable> <full stop> Modify **(** *expressionlist* **)**

donde *expressionlist* se construye añadiendo <expression> a la <actual parameter list>. La gramática abstracta se determina a partir de esta expresión concreta de acuerdo con 12.2.1. El mismo modelo se aplica a la segunda forma de <indexed variable>.

La forma de sintaxis concreta:

<variable> <exclamation mark> <field name> <is assigned sign> <expression>

es sintaxis concreta derivada para:

<variable> <full stop> *field-modify-operation-name* **(** <expression> **)**

donde el *field-modify-operation-name* se forma mediante la concatenación del nombre de campo y de "Modify". La sintaxis abstracta se determina a partir de esta expresión concreta de acuerdo con 12.2.1. El mismo modelo se aplica a la segunda forma de <field variable>.

12.3.3.2 Inicialización por defecto

Una inicialización por defecto permite inicializar todas las variables de un género especificado con el mismo ítem de datos, cuando se crean las variables.

Gramática textual concreta

<default initialization> ::=
default [<virtuality>] [<constant expression>] [<end>]

Una <data type definition> o <syntype definition> no debe contener más de una <default initialization>.

La <constant expression> sólo puede omitirse si <virtuality> es **redefined** o **finalized**.

Semántica

Una inicialización por defecto puede añadirse a la <operations> de una definición de tipo de datos. Una inicialización por defecto especifica que cualquier variable declarada con el género introducido por la definición de tipo de datos o definición de sintipo es inicialmente asociada con el resultado de <constant expression>.

Modelo

Una inicialización por defecto es una notación taquigráfica para especificar una inicialización explícita para todas las variables que se declara son del <sort>, pero donde la <variable definition> no ha recibido una <constant 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 resultado esté en el intervalo.

Cualquier género definido por una <object data type definition> recibe implícitamente una <default initialization> de valor Null, salvo que en la <object data type definition> hubiera una <default initialization> explícita.

Todo género de pid se trata como su hubiera recibido implícitamente una <default initialization> de valor Null.

Si en una inicialización por defecto redefinida se omite la <constant expression>, no se añade la inicialización explícita.

12.3.4 Expresiones imperativas

Las expresiones imperativas obtienen resultados 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 una expresión imperativa se desplaza a la primera tarea insertada durante la transformación descrita. Si aparecen varias expresiones imperativas en una expresión, las tareas se insertan en el mismo orden en que aparecen las expresiones imperativas en la expresión.

Gramática abstracta

Imperative-expression = *Now-expression*
| *Pid-expression*
| *Timer-active-expression*
| *Any-expression*

Gramática textual concreta

<imperative expression> ::=
<now expression>
| <import expression>
| <pid expression>
| <timer active expression>
| <any expression>
| <state expression>

Las expresiones imperativas son expresiones para acceder al reloj de sistema, al resultado de variables importadas, al pid asociado con un agente, al estado de los temporizadores o para suministrar ítems de datos no especificados.

12.3.4.1 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 del 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.

12.3.4.2 Expresión import

Gramática textual concreta

La sintaxis concreta para una expresión import (importación) se define en 10.6.

Semántica

Además de la semántica definida en 10.6, una expresión importación se interpreta como un acceso a variable (véase 12.3.2), a la variable implícita para la expresión importación.

Modelo

La expresión importación tiene una sintaxis implícita para la importación del resultado tal como se define en 10.6, y también un *Variable-access* implícito de la variable implícita para la importación en el contexto en el 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 resultado 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.

12.3.4.3 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**

<create expression> ::=
<create request>

Semántica

Una expresión pid accede a una de las variables anónimas implícitas self (mismo), parent (progenitor), offspring (vástago) o sender (emisor) (véase la cláusula 9, *Modelo*). La expresión pid **self**, **parent**, **offspring** o **sender** tiene un resultado que es el último pid asociado con la correspondiente variable implícita tal como se define en la cláusula 9.

El género dinámico de una <pid expression> es el género dinámico de su resultado.

Una expresión pid **parent**, **offspring** o **sender** tiene un género estático que es pid.

Si una <create expression> incluye un <agent identifier>, tiene un género estático, que es el género pid del agente denotado por <agent identifier>. Si una <create expression> incluye un <agent type identifier>, tiene un género estático, que es el género pid del tipo de agente identificado por el <agent type identifier>. Si la <create expression> incluye **this**, tiene un género estático, que es el género pid del agente o del tipo de agente en el que aparece la expresión crear. Si <create expression> incluye **this** y aparece en un contexto que no está dentro de un agente o de un tipo de agente (por ejemplo, en un procedimiento global), tiene un pid de género estático.

Una expresión **self** tiene un género estático que es el pid del agente o tipo de agente en el que aparece la expresión **self**. Si aparece en un contexto que no está dentro de un agente o de un tipo de agente (por ejemplo, en un procedimiento global), tiene un pid de género estático.

Modelo

La utilización de <create expression> en una expresión es una notación taquigráfica para insertar una petición de creación justamente antes de la acción en la que tiene lugar <create expression> seguida de una asignación de **offspring** a una variable anónima declarada implícitamente del mismo género que el género estático de <create expression>. La variable implícita se utiliza entonces en la expresión. Si <create expression> aparece varias veces en una expresión, se utiliza una variable distinta para cada ocurrencia. En este caso, el orden de las peticiones de creación insertadas y de las asignaciones variables coincide con el orden de <create expression>s.

Si <create expression> contiene un <agent type identifier>, las transformaciones que se aplican a un enunciado crear que contiene un <agent type identifier>, se aplican también a los enunciados crear implícitos que resultan de la transformación de una <create expression> (véase 11.13.2).

12.3.4.4 Expresión Timer active

Gramática abstracta

Timer-active-expression :: *Timer-identifier*
*Expression**

Los géneros de la lista de *Expression* en *Timer-active-expression* tienen que corresponder en posición con la lista de *Sort-reference-identifier* que sigue al *Timer-name* (11.15) 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 presenta el resultado verdadero si el temporizador identificado por el identificador de temporizador e inicializado con los mismos resultados que los denotados por la lista de expresiones (si existe) está activo (véase 11.15). De no ser así, la expresión temporizador activo tiene el resultado falso. Las expresiones se interpretan en el orden dado.

Si un género especificado en una definición de temporizador es un sintipo, la verificación de intervalo definida en 12.1.9.5 aplicada a la expresión correspondiente en <expression list> debe ser el valor booleano predefinido verdadero o; en caso contrario, se genera la excepción predefinida OutOfRange (véase D.3.16).

12.3.4.5 Expresión any

Any-expression es útil para modelar el comportamiento en los casos en que indicar un ítem de datos específico implicaría una sobreespecificación. A partir de un valor devuelto por una *Any-expression* no pueden suponerse otros resultados devueltos por la interpretación de *Any-expression*.

Gramática abstracta

Any-expression :: *Sort-reference-identifier*

Gramática textual concreta

<any expression> ::=
any (<sort> **)**

El <sort> debe contener elementos.

Semántica

Una *Any-expression* devuelve un elemento no especificado del género o sintipo designado por el *Sort-reference-identifier,* si ese género o sintipo es un género de valor. Si *Sort-reference-identifier* denota un *Syntype-identifier*, el resultado estará dentro del intervalo de dicho sintipo. Si el género o sintipo designado por *Sort-reference-identifier* es un género de objeto o género de pid, la *Any-expresssion* retorna Null.

12.3.4.6 Expresión state

Gramática textual concreta

<state expression> ::=
state

Semántica

Una expresión state (estado) denota el valor de la cadena de Character que contiene la ortografía del último estado al que se ha entrado.

Modelo

<state expression> es sintaxis derivada para un literal Charstring que contenga la ortografía del nombre de último estado al que se ha pasado de la unidad de ámbito circundante más próxima. Si dicho estado no existe, <state expression> denota la cadena vacía (' '). El constructivo se transforma junto con <dash nextstate> (véase anexo F).

12.3.5 Llamada a procedimiento de devolución de valor

La gramática para una llamada a procedimiento de devolución de valor y las limitaciones de la semántica estática se muestran en 11.13.3.

Gramática textual concreta

<value returning procedure call> ::=
[**call**] <procedure call body>
| [**call**] <remote procedure call body>

La palabra clave **call** no puede omitirse si la <value returning procedure call> es sintácticamente ambigua con una operación (o variable) con el mismo nombre seguido de una lista de parámetros.

NOTA 1 – Esta ambigüedad no se resuelve por contexto.

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 que tenga un <procedure result>.

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

Después de que se ha aplicado el *Modelo* para **this**, el <procedure identifier> debe denotar un procedimiento que contenga una transición de arranque.

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

El <procedure call body> representa un *Value-returning-call-node*, donde *Procedure-identifier* se representa por el <procedure identifier>, y la lista de *Expression*s se representa por la lista de parámetros reales. El <remote procedure call body> representa un *Value-returning-call-node*, donde *Procedure-identifier* contiene únicamente el *Procedure-identifier* del procedimiento implícitamente definido por el *Modelo* siguiente. La semántica del *Value-returning-call-node* se muestra en 11.13.3.

Modelo

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

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

Cuando <value returning procedure call> contiene un <remote procedure call body> el procedimiento siguiente con un nombre anónimo al que se hace referencia como a *RPCcall*, está implícitamente definido. *RPCsort* es el <sort> en <procedure result> de la definición de procedimiento denotada por el <procedure identifier>.

```
procedure RPCcall -> RPCsort;
    start ;
    return call <remote procedure call body>;
endprocedure ;
```

NOTA 2 – Esta transformación no se aplica de nuevo a la definición de procedimiento implícito.

13 Definición de sistema genérica

Una especificación de sistema puede tener partes facultativas y parámetros de sistema con resultados 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 concreta seleccionando un subconjunto adecuado de la misma y proporcionando un ítem de datos para cada uno de los parámetros de sistema. La especificación de sistema resultante no contiene sinónimos externos, y se denomina especificación de sistema específica.

Una definición de sistema genérica es una definición de sistema que contiene un sinónimo definido por un <external synonym definition item> (véase 12.1.9.6), una operación definida por una <external operation definition> (véase 12.1.8), un procedimiento definido por una <external procedure definition> (véase 9.4) o <informal text> en una opción de transición (véase 13.2). Una definición de sistema específico se crea a partir de una definición de sistema genérica proporcionando resultados para los <external synonym definition item>s, proporcionando el comportamiento para <external operation definition>s y <external procedure definition>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.

13.1 Definición facultativa

Gramática textual concreta

<select definition> ::=

select if (<Boolean simple expression> **)** <end>
{ <agent type definition>
| <agent type reference>
| <agent definition>
| <channel definition>
| <signal definition>
| <signal list definition>
| <remote variable definition>
| <remote procedure definition>
| <data definition>
| <data type reference>
| <composite state type definition>
| <composite state type reference>
| <state partitioning>
| <textual typebased state partition definition>
| <timer definition>
| <variable definition>
| <procedure definition>
| <procedure reference>
| <channel to channel connection>
| <select definition>
| <macro definition>
| <exception 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 tipos de datos definidos dentro del lote 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> ::=

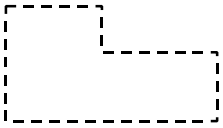
<option symbol> ***contains***
{ **select if (** <Boolean simple expression> **)**
{ <agent type diagram>
| <agent type reference area>
| <agent area>
| <channel definition area>
| <agent text area>
| <procedure text area>
| <composite state type diagram>
| <composite state type reference area>
| <state partition area>
| <procedure area>
| <create line area>
| <option area> } + }

<option symbol> ::=

{ <dashed line symbol> }***-set***

<dashed line symbol> ::=

El <option symbol> debe formar 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 interior).

Una <option area> puede aparecer en cualquier lugar, salvo dentro de <state machine graph area> y <type state machine graph area>. Una <option area> puede contener solamente las áreas y diagramas que están sintácticamente permitidos en ese lugar.

Semántica

Si el resultado de la <Boolean simple expression> es el valor booleano predefinido 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 suprimen y 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.

13.2 Cadena de transición facultativa

Gramática textual concreta

```
<transition option> ::=
                        alternative <alternative question> <end>
                            <decision body>
                        endalternative

<alternative question> ::=
                        <simple expression>
                    |   <informal text>
```

Toda <constant expression> en <answer> de <decision body> tiene que ser una <simple expression>. Las <answer>s en la <decision body> de una <transition option> tienen que ser mutuamente exclusivas. Si la <alternative question> es una <expression>, la *Range-condition* de las <answer>s en el <decision body> tienen que ser del mismo género que el de <alternative question>.

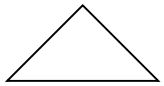
Hay ambigüedad sintáctica entre <informal text> y <character string> en <alternative question> y <answer>s en el <decision body>. Si la <alternative question> y todas las <answer>s son <character string>, todas ellas se interpretan como <informal text>. Si la <alternative answer> o cualquier <answer> es una <character string> y ésta no concuerda con el contexto de la opción de transición, la <character string> denota <informal text>.

No puede omitirse ninguna <answer> en <answer part>s de una <decision body> de una <transition option>.

Gramática gráfica concreta

<transition option area> ::=
<transition option symbol> ***contains*** <alternative question>
is followed by <graphical decision body>

<transition option symbol> ::=



Los <flow line symbol>s en <graphical decision body> están conectados a la parte inferior del <transition option symbol>.

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

Las <graphical answer>s en el <graphical decision body> tienen que ser mutuamente exclusivas.

Semántica

Se seleccionan constructivos en una <graphical answer part> si la <answer> contiene el resultado de la <alternative question>. Si ninguna de las <answer>s contiene el resultado de la <alternative question>, se seleccionan los constructivos en la <graphical else part>.

Si no existe una <graphical else part> y no se selecciona ninguno de los trayectos salientes, la selección es no válida.

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) un <join> hacia el <terminator statement> siguiente en caso de que la opción de transición sea el último <action statement> en una <transition string>; o

b) un <join> hacia el primer <action statement> que sigue a la opción de transición.

<transition option> de terminación se define en 11.13.5.

En una transformación, <transition option> y <transition option area> se suprimen y se reemplazan por los constructivos seleccionados contenidos.

ANEXO A

Índice de no terminales

Los siguientes no terminales (en inglés) han sido intencionalmente definidos y no utilizados: <macro call>, <page>, <comment area>, <text extension area> y <sdl specification>.

ANEXO B

Reservado para uso futuro

ANEXO C

Reservado para uso futuro

ANEXO D

Datos predefinidos SDL

D.1 Introducción

Los datos predefinidos SDL se basan en tipos de datos abstractos que se definen en términos de sus propiedades abstractas más que en términos de alguna implementación concreta. Aunque la definición de un tipo abstracto es una forma posible de implementar dichos tipos de datos, una implementación no está obligada a elegir la forma de implementar el tipo de datos abstractos, en la medida en que se preserva el mismo comportamiento abstracto.

En este anexo se definen los tipos de datos predefinidos, incluido el género booleano que define propiedades para dos literales true (verdadero) y false (falso). Los dos *terms* booleanos verdadero y falso no deben definirse (directa o indirectamente) como equivalentes. Toda expresión constante booleana utilizada al margen de las definiciones de tipos de datos debe interpretarse como verdadera o falsa. Si no es posible reducir dicha expresión a verdadero o falso, la especificación queda incompleta y permite más de una interpretación del tipo de datos.

Los datos predefinidos se definen en un lote de utilización implícita Predefined (véase 7.2). En este anexo se define este lote.

D.2 Notación

Este anexo amplía, a tal fin, la sintaxis concreta de SDL describiendo las propiedades abstractas de las operaciones añadidas por una definición de tipo de datos. No obstante, esta sintaxis adicional se utiliza exclusivamente con fines explicativos y no amplía la sintaxis definida en el texto principal. Una especificación que utilice la sintaxis definida en este anexo no es, por tanto, un SDL válido.

Las propiedades abstractas aquí descritas no especifican una determinada representación de los datos predefinidos. En cambio, una interpretación debe ser conforme con estas propiedades. Cuando se interpreta una <expression>, la evaluación de la expresión produce un valor (por ejemplo, el resultado de una <operation application>). Dos expresiones E1 y E2 son equivalentes, si:

a) hay una <equation> E1 = = E2, o

b) una de las ecuaciones derivadas del conjunto dado de <quantified equation>s es E1 = = E2, o

c) i) E1 es equivalente a EA; y

   ii) E2 es equivalente a EB; y

   iii) hay una ecuación, o una ecuación derivada del conjunto dado de ecuaciones cuantificadas es tal que EA = = EB; o

d) sustituyendo un subtérmino de E1 por un término de la misma clase que la del subtérmino que produce un término E1A es posible mostrar que E1A pertenece a la misma clase que E2.

En otro caso, las dos expresiones no son equivalentes.

Dos expresiones que son equivalentes representan el mismo valor.

La interpretación de expresiones es conforme con estas reglas si dos expresiones equivalentes representan el mismo valor, y dos expresiones no equivalentes representan valores diferentes.

D.2.1 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 argumentos y valores de resultados de operadores, dando así significado a los operadores. Los axiomas pueden ser axiomas booleanos o tener forma de ecuaciones de equivalencia algebraica.

Una operación definida por una <axiomatic operation definition> se trata como una definición completa en lo que se refiere a la especialización. Es decir, cuando un tipo de datos definido por el lote Predefined es especializado y se redefine una operación en el tipo especializado, todos los axiomas que mencionan el nombre de la operación son sustituidos por la correspondiente definición en el tipo especializado.

Gramática textual concreta

<axiomatic operation definitions> ::=
axioms <axioms>

<axioms> ::=
<equation> { <end> <equation>}***** [<end>]

<equation> ::=
<unquantified equation>
| <quantified equations>
| <conditional equation>
| <literal equation>
| <noequality>

<unquantified equation> ::=
<term> == <term>
| <Boolean axiom>

<term> ::=
<constant expression>
| <error term>

<quantified equations> ::=
<quantification> **(**<axioms> **)**

<quantification> ::=
for all <value name> { **,** <value name> }***** **in** <sort>

En este anexo se modifica <operations> (véase 12.1.1) tal como se describe a continuación.

<operations> ::=
<operation signatures>
{ <operation definitions> | <axiomatic operation definitions> }

<axiomatic operation definitions> sólo puede utilizarse para describir el comportamiento de operadores.

Un <identifier>, que es un nombre sin calificación que aparece en un <term>, representa:

a) un <operation identifier> (véase 12.2.7);

b) un <literal identifier> (véase 12.2.2);

c) un <value identifier> si existe una definición de dicho nombre en una <quantification> de <quantified equations> que circunda el <term>, que debe tener un género adecuado para el contexto; en caso contrario

d) un <value identifier> que tiene una ecuación cuantificada implícita para la <unquantified equation>.

Dos o más ocurrencias del mismo identificador de valor no limitado dentro de una <unquantified equation>, (o si la <unquantified equation> está contenida en una <conditional equation>, dentro de la <conditional equation>), entrañan una <quantification>.

Dentro de una <unquantified equation> (o en el caso de que la <unquantified equation> esté contenida en una <conditional equation>, dentro de la <conditional equation>), tiene que haber exactamente un género para cada identificador de valor implícitamente cuantificado, que sea consistente con todos los usos.

Semántica

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.

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.

En general no hay necesidad o motivo para distinguir entre un término fundamental y el resultado del término fundamental. Por ejemplo, el término fundamental para el elemento Entero unidad puede escribirse "1". Normalmente hay varios términos fundamentales que denotan el mismo ítem de datos, como por ejemplo, los términos fundamentales Enteros "0+1", "3–2" y "(7+5)/12", siendo habitual considerar una forma simple del término fundamental (en este caso "1") para denotar el ítem de datos.

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 siempre se introduce mediante ecuaciones cuantificadas, 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 in X (for all z in X ... )
```

introduce sólo un identificador denominado z de género X.

En la sintaxis concreta de axiomas 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 <género>. 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.

D.2.2 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 textual concreta

<conditional equation> ::=
<restriction> { , <restriction> }* ==><restricted equation>

<restricted equation> ::=
<unquantified equation>

<restriction> ::=
<unquantified equation>

Semántica

Una ecuación condicional establece que los términos denotan el mismo ítem de datos solamente cuando todo identificador de valor en las ecuaciones restringidas denota un ítem de datos del que puede mostrarse, mediante otras ecuaciones, que satisface la restricción.

La semántica de un conjunto de ecuaciones para un tipo de datos que incluya 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 cuantificadas. Como esto se aplica a la cuantificación explícita e implícita, se genera un conjunto de ecuaciones no cuantificadas únicamente en términos fundamentales.

b) Una ecuación condicional se denominará 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 ellas 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.

D.2.3 Igualdad

Gramática textual completa

<noequality> ::=
noequality

Modelo

Toda <data type definition> que introduzca un género denominado S tiene la siguiente <operation signature> implícita en su <operator list>, a menos que <noequality> esté presente en los <axioms>:

equal (S, S) **->** Boolean;

donde Boolean es el género booleano predefinido.

Toda <data type definition> que introduzca un género denominado S, tal que únicamente contenga <axiomatic operation definitions> en <operator list>, tiene un conjunto de ecuación implícito:

```
for all a,b,c in S (
  equal(a, a) == true;
  equal(a, b) == equal(b, a);
  equal(a, b) and equal(b, c)  ==> equal(a, c) == true;
  equal(a, b) == true          ==> a == b;)
```

y una <literal equation> implícita:

```
for all L1,L2 in S literals (
  spelling(L1) /= spelling(L2) ==> L1 = L2 == false;)
```

D.2.4 Axiomas booleanos

Gramática textual concreta

<Boolean axiom> ::=
<Boolean term>

Semántica

Un axioma booleano es un enunciado de verdad que se cumple en todas las condiciones para los tipos de datos definidos.

Modelo

Un axioma de la forma:

<Boolean term>;

es sintaxis derivada para la ecuación de sintaxis concreta:

<Boolean term> == << **package** Predefined/**type** Boolean >> true;

D.2.5 Término condicional

Semántica

Una ecuación que contenga un término condicional es semánticamente equivalente a un conjunto de ecuaciones en el que se han eliminado todos los identificadores de valores cuantificados en el término booleano. Este conjunto de ecuaciones puede formarse sustituyendo simultáneamente en toda la ecuación de término condicional, cada <value identifier> de la <conditional expression> por el término fundamental del género adecuado. En este conjunto de ecuaciones la <conditional expression> será siempre sustituida por un término fundamental booleano. En lo que sigue, a este conjunto de ecuaciones se hace referencia como conjunto fundamental ampliado.

Una ecuación de término condicional es equivalente al conjunto de ecuación que contiene:

a) para toda *equation* en el conjunto fundamental ampliado para el cual la <conditional expression> es equivalente a verdadero, la *equation* del conjunto fundamental ampliado en la que la <conditional expression> ha sido sustituida por la <consequence expression> (fundamental); y

b) para toda *equation* en el conjunto fundamental ampliado para el cual la <conditional expression> es equivalente a falso, la *equation* del conjunto fundamental ampliado en la que la <conditional expression> ha sido sustituida por la <alternative expression> (fundamental).

Nótese que en el caso especial de una ecuación de la forma:

ex1 == **if** a **then** b **else** c **fi**;

esto es equivalente a la pareja ecuaciones condicionales:

a == true ==> ex1 == b;
a == false ==> ex1 == c;

D.2.6 Término de error

Los errores se utilizan para permitir que se definan completamente las propiedades de un tipo de datos, incluso en aquellos casos en que no puede darse un significado específico al resultado de un operador.

Gramática textual concreta

<error term> ::=
raise <exception name>

Un <error term> no debe utilizarse como parte de una <restriction>.

No debe ser posible derivar de *equations* que un <literal identifier> es igual a <error term>.

Semántica

A término puede ser un <error term> de forma que sea posible especificar las circunstancias en las cuales un operador produce un error. Si esas circunstancias se producen durante la interpretación, se genera la excepción con <exception name>.

D.2.7 Literales no ordenados

Gramática textual concreta

<unordered> ::=
unordered

En este anexo se modifica la sintaxis concreta para el constructivo de tipo lista de literales (véase 12.1.7.1) como sigue.

<literal list> ::=
[<protected>] **literals** [<unordered>]
<literal signature> { **,** <literal signature> }* <end>

Modelo

Si se utiliza <unordered>, no se aplica el *Modelo* de 12.1.7.1. En consecuencia, las operaciones de ordenamiento "<", ">","<=",">=", first, last, pred, succ y num no se definen implícitamente para este tipo de datos.

D.2.8 Ecuaciones de literales

Gramática textual concreta

<literal equation> ::=
<literal quantification>
(<equation> { <end> <equation> }* [<end>] **)**

<literal quantification> ::=
for all <value name> { **,** <value name> }* **in** <sort> **literals**
| **for all** <value name> { **,** <value name> }* **in** { <sort> | <value identifier> } **nameclass**

En este anexo se modifica la sintaxis concreta para <spelling term> (véase 12.1.9.2) como sigue.

<spelling term> ::=
spelling ({ <operation name> | <value identifier> })

El <value identifier> en un <spelling term> debe ser un <value identifier> definido por una <literal quantification>.

Semántica

La correspondencia de literales es una notación taquigráfica que tiene por objeto definir un gran número (posiblemente infinito) de axiomas que pueden comprender todos los literales de un género o todos los nombres en una clase de nombre. La correspondencia de literales permite establecer una correspondencia entre los literales de un género y los valores del género.

<spelling term> se utiliza en calificaciones de literal para referirse a la cadena de caracteres que contiene la ortografía del literal. Este mecanismo permite utilizar los operadores cadena de caracteres (Charstring) para definir las ecuaciones de literales.

Modelo

Una <literal equation> es una notación taquigráfica para un conjunto de <axioms>. En cada una de las <equation>s contenidas en <literal equation>, los <value identifier>s definidos por el <value name> en la <literal quantification> son reemplazados. En cada <equation> derivada, cada ocurrencia del mismo <value identifier> se reemplaza por el mismo <literal identifier> del <sort> de la <literal quantification> (si se usó **literales**) o por el mismo <literal identifier> de la nameclass a que se hace referencia (si se usó **nameclass**). 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**.

Si una <literal quantification> contiene uno o más <spelling term>s, éstos se sustituyen por literales cadena de caracteres Charstring (véase D.3). Charstring se utiliza para sustituir el <value identifier> después de la <literal equation> que contiene el <spelling term> y se amplía como se define en 12.1.9.2, utilizando <value identifier> en lugar de <operation name>.

NOTA – Las ecuaciones de literales no afecta a los operadores de nulidad en <operation signature>s.

D.3 Lote predefinido

En las definiciones siguientes todas las referencias a nombres definidos en el lote Predefined se tratan como prefijadas por la calificación <<**package** Predefined>>. Para mejorar la legibilidad, se omite esta calificación.

```
/* */
package Predefined
/*
```

D.3.1 Género booleano (Boolean sort)

D.3.1.1 Definición

```
*/
value type Boolean
  literals true, false;
  operators
     "not"( this Boolean )                -> this Boolean;
     "and"( this Boolean, this Boolean )  -> this Boolean;
     "or" ( this Boolean, this Boolean )  -> this Boolean;
     "xor"( this Boolean, this Boolean )  -> this Boolean;
     "=>" ( this Boolean, this Boolean )  -> this Boolean;
axioms
     not( true  )     == false;
     not( false )     == true ;
/* */
     true  and true   == true ;
     true  and false  == false;
     false and true   == false;
     false and false  == false;
/* */
     true  or true    == true ;
     true  or false   == true ;
     false or true    == true ;
     false or false   == false;
/* */
     true  xor true   == false;
     true  xor false  == true ;
     false xor true   == true ;
     false xor false  == false;
/* */
```

```
     true => true    == true ;
     true => false   == false;
     false=> true    == true ;
     false=> false   == true ;
endvalue type Boolean;
/*
```

D.3.1.2 Utilización

El género Boolean (booleano) se utiliza para representar valores verdaderos y falsos. A menudo se utiliza como resultado de una comparación.

El género Boolean es ampliamente utilizado en SDL.

D.3.2 Género carácter (Character sort)

D.3.2.1 Definición

```
*/
value type Character
  literals
     NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
     BS,  HT,  LF,  VT,  FF,  CR,  SO,  SI,
     DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
     CAN, EM,  SUB, ESC, IS4, IS3, IS2, IS1,
     ' ', '!', '"', '#', '$', '%', '&', '''',
     '(', ')', '*', '+', ',', '-', '.', '/',
     '0', '1', '2', '3', '4', '5', '6', '7',
     '8', '9', ':', ';', '<', '=', '>', '?',
     '@', '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', '{', '|', '}', '~', DEL;
/* '''' is an apostrophe, ' ' is a space, '~' is a tilde */
/* */
  operators
     chr  ( Integer ) -> this Character;
/*   "<", "<=", ">", ">=", and "num" are implicitly defined (see 12.1.7.1). */
axioms
     for all a,b in Character (
     for all i in Integer (
/* definition of Chr */
     chr(num(a)) == a;
     chr(i+128)  == chr(i);
     ));
endvalue type Character;
/*
```

D.3.2.2 Utilización

El género Character (Character sort) se utiliza para representar caracteres del Alfabeto Internacional de Referencia (Recomendación T.50).

D.3.3 Género cadena (String sort)

D.3.3.1 Definición

```
*/
value type String < type Itemsort >
/* Strings are "indexed" from one */
  operators
     emptystring                                  -> this String;
     mkstring   ( Itemsort                    )  -> this String;
     make       ( Itemsort                    )  -> this String;
     length     ( this String                    ) -> Integer;
     first      ( this String                    ) -> Itemsort;
```

```
     last      ( this String                    ) -> Itemsort;
     "//"      ( this String, this String       ) -> this String;
     extract   ( this String, Integer           ) -> Itemsort raise InvalidIndex;
     modify    ( this String, Integer, Itemsort ) -> this String;
     substring ( this String, Integer, Integer ) -> this String raise InvalidIndex;
     /* substring (s,i,j) gives a string of length j starting from the ith element */
     remove    ( this String, Integer, Integer ) -> this String;
     /* remove (s,i,j) gives a string with a substring of length j starting from
        the ith element removed */
axioms
     for all e in Itemsort ( /*e – element of Itemsort*/
     for all s,s1,s2,s3 in String (
     for all i,j in Integer (
/* constructors are emptystring, mkstring, and "//" */
/* equalities between constructor terms */
     s // emptystring== s;
     emptystring // s== s;
     (s1 // s2) // s3== s1 // (s2 // s3);
/* */
/* definition of length by applying it to all constructors */
     <<type String>>length(emptystring)== 0;
     <<type String>>length(mkstring(e))== 1;
     <<type String>>length(s1 // s2)    == length(s1) + length(s2);
     make(s)                              == mkstring(s);
/* */
/* definition of extract by applying it to all constructors,
       error cases handled separately */
     extract(mkstring(e),1)                   == e;
     i <= length(s1) ==> extract(s1 // s2,i)== extract(s1,i);
     i > length(s1) ==> extract(s1 // s2,i)  == extract(s2,i-length(s1));
     i<=0 or i>length(s) ==> extract(s,i)   == raise InvalidIndex;
/* */
/* definition of first and last by other operations */
     first(s)  == extract(s,1);
     last(s)   == extract(s,length(s));
/* */
/* definition of substring(s,i,j) by induction on j,
       error cases handled separately */
     i>0 and i-1<=length(s)            ==>
                    substring(s,i,0)  == emptystring;
/* */
     i>0 and j>0 and i+j-1<=length(s)  ==>
                    substring(s,i,j)  == substring(s,i,j-1) // mkstring(extract(s,i+j-1));
/* */
     i<=0 or j<0 or i+j-1>length(s)    ==>
                    substring(s,i,j)  == raise InvalidIndex;
/* */
/* definition of modify by other operations */
     modify(s,i,e)== substring(s,1,i-1) // mkstring(e) // substring(s,i+1,length(s)-i);
/* definition of remove */
     remove(s,i,j)   == substring(s,1,i-1) // substring(s,i+j,length(s)-i-j+1);
     )));
endvalue type String;
/*
```

D.3.3.2 Utilización

Los operadores make, extract y modify se utilizarán típicamente con las notaciones taquigráficas definidas en 12.2.4 y 12.3.3.1 para ganar acceso a los valores de cadenas y asignar valores a las cadenas.

D.3.4 Género cadena de caracteres (Charstring sort)

D.3.4.1 Definición

```
*/
value type Charstring
     inherits String < Character > ( '' = emptystring )
     adding
        operators ocs in nameclass
             '''' ( (' ':'&') or '''''' or ('(': '~') )+ '''' -> this Charstring;
/* character strings of any length of any characters from a space ' ' to a tilde '~' */
axioms
     for all c in Character nameclass (
     for all cs, cs1, cs2 in ocs nameclass (
     spelling(cs) == spelling(c)                    ==> cs == mkstring(c);
/* string 'A' is formed from character 'A' etc. */
     spelling(cs) == spelling(cs1) // spelling(cs2),
     length(spelling(cs2)) == 1                     ==> cs  == cs1 // cs2;
     ));
endvalue type Charstring;
/*
```

D.3.4.2 Utilización

```
/*
```

El género cadena de caracteres (Charstring sort) define cadenas de caracteres.

Un literal Charstring puede contener caracteres imprimibles y espacios.

Un carácter no imprimible puede utilizarse como una cadena empleando mkstring, por ejemplo mkstring (DEL).

Ejemplo:

```
synonym newline_prompt Charstring = mkstring(CR) // mkstring(LF) // '$>';
```

D.3.5 Género entero (Integer sort)

D.3.5.1 Definición

```
*/
value type Integer
  literals unordered nameclass (('0':'9')*) ('0':'9'));
  operators
     "-"  ( this Integer               ) -> this Integer;
     "+"  ( this Integer, this Integer ) -> this Integer;
     "-"  ( this Integer, this Integer ) -> this Integer;
     "*"  ( this Integer, this Integer ) -> this Integer;
     "/"  ( this Integer, this Integer ) -> this Integer raise DivisionByZero;
     "mod"( this Integer, this Integer ) -> this Integer raise DivisionByZero;
     "rem"( this Integer, this Integer ) -> this Integer;
     "<"  ( this Integer, this Integer ) -> Boolean;
     ">"  ( this Integer, this Integer ) -> Boolean;
     "<=" ( this Integer, this Integer ) -> Boolean;
     ">=" ( this Integer, this Integer ) -> Boolean;
     power( this Integer, this Integer ) -> this Integer;
     bs in nameclass '''' ( (('0' or '1')*'''B') or ((('0':'9') or ('A':'F'))*'''H') )
              -> this Integer;
axioms  noequality
     for all a,b,c in Integer (
/* constructors are 0, 1, +, and unary - */
/* equalities between constructor terms */
     (a + b) + c          == a + (b + c);
     a + b                == b + a;
     0 + a                == a;
     a + (- a)            == 0;
     (- a) + (- b)     == - (a + b);
     <<type Integer>> - 0 == 0;
     - (- a)              == a;
/* */
```

```
/* definition of binary "-" by other operations */
     a - b               == a + (- b);
/* */
/* definition of "*" by applying it to all constructors */
     0 * a               == 0;
     1 * a               == a;
     (- a) * b           == - (a * b);
     (a + b) * c         == a * c + b * c;
/* */
/* definition of "<" by applying it to all constructors */
     a < b                                  == 0 < (b - a);
     <<type Integer>> 0 < 0                 == false;
     <<type Integer>> 0 < 1                 == true ;
     0 < a          == true ==> 0 < (- a)   == false;
     0 < a and 0 < b == true ==> 0 < (a + b) == true ;
/* */
/* definition of ">", "equal", "<=", and ">=" by other operations */
     a > b               == b < a;
     equal(a, b)         == not(a < b or a > b);
     a <= b              == a < b or a = b;
     a >= b              == a > b or a = b;
/* */
/* definition of "/" by other operations */
     a / 0                                      == raise DivisionByZero;
     a >= 0 and b > a          == true ==> a / b == 0;
     a >= 0 and b <= a and b > 0  == true ==> a / b == 1 + (a-b) / b;
     a >= 0 and b < 0          == true ==> a / b == - (a / (- b));
     a < 0 and b < 0             == true ==> a / b == (- a) / (- b);
     a < 0 and b > 0             == true ==> a / b == - ((- a) / b);
/* */
/* definition of "rem" by other operations */
     a rem b == a - b * (a/b);
/* */
/* definition of "mod" by other operations */
     a >= 0 and b > 0               ==> a mod b  == a rem b;
     b < 0                             ==> a mod b  == a mod (- b);
     a < 0 and b > 0 and a rem b = 0 ==> a mod b  == 0;
     a < 0 and b > 0 and a rem b < 0 ==> a mod b  == b + a rem b;
     a mod 0 == raise DivisionByZero;
/* */
/* definition of power by other operations */
     power(a, 0)            == 1;
     b > 0 ==> power(a, b)  == a * power(a, b-1);
     b < 0 ==> power(a, b)  == power(a, b+1) / a; );
/* */
/* definition of literals */
     <<type Integer>> 2 == 1 + 1;
     <<type Integer>> 3 == 2 + 1;
     <<type Integer>> 4 == 3 + 1;
     <<type Integer>> 5 == 4 + 1;
     <<type Integer>> 6 == 5 + 1;
     <<type Integer>> 7 == 6 + 1;
     <<type Integer>> 8 == 7 + 1;
     <<type Integer>> 9 == 8 + 1;
/* */
/* literals other than 0 to 9 */
     for all a,b,c in Integer nameclass (
     spelling(a) == spelling(b) // spelling(c),
     length(spelling(c)) == 1               ==> a == b * (9 + 1) + c;
     );
/* */
/* hex and binary representation of Integer */
     for all b in Bitstring nameclass (
     for all i in bs nameclass (
     spelling(i) == spelling(b)            ==> i == <<type Bitstring>>num(b);
     ));
endvalue type Integer;
/*
```

D.3.5.2 Utilización

El género entero (Integer sort) se utiliza para números enteros matemáticos con notación decimal, hexadecimal o binaria.

D.3.6 Sintipo natural (natural syntype)

D.3.6.1 Definición

```
*/
syntype Natural = Integer constants >= 0 endsyntype Natural;
/*
```

D.3.6.2 Utilización

El sintipo natural (natural syntype) se utiliza cuando sólo se requieren enteros positivos. Todos los operadores serán los operadores enteros, pero cuando se utiliza un valor como parámetro, o se asigna un valor, se comprueba dicho valor. Un valor negativo produce un error.

D.3.7 Género real (Real sort)

D.3.7.1 Definición

```
*/
value type Real
  literals unordered nameclass
        ( ('0':'9')* ('0':'9') ) or ( ('0':'9')* '.'('0':'9')+ );
  operators
     "-"  ( this Real            ) -> this Real;
     "+"  ( this Real, this Real ) -> this Real;
     "-"  ( this Real, this Real ) -> this Real;
     "*"  ( this Real, this Real ) -> this Real;
     "/"  ( this Real, this Real ) -> this Real raise DivisionByZero;
     "<"  ( this Real, this Real ) -> Boolean;
     ">"  ( this Real, this Real ) -> Boolean;
     "<=" ( this Real, this Real ) -> Boolean;
     ">=" ( this Real, this Real ) -> Boolean;
     float( Integer              ) -> this Real;
     fix  ( this Real            ) -> Integer;
axioms noequality
     for all r,s in Real (
     for all a,b,c,d in Integer (
/* constructors are float and "/" */
/* equalities between constructor terms allow to reach always a form
        float(a) / float(b) where b > 0 */
     r / float(0)                                    == raise DivisionByZero;
     r / float(1)                                    == r;
     c /= 0 ==> float(a) / float(b)                  == float(a*c) / float(b*c);
     b /= 0 and d /= 0 ==>
     (float(a) / float(b)) / (float(c) / float(d))  == float(a*d) / float(b*c);
/* */
/* definition of unary "-" by applying it to all constructors */
     - (float(a) / float(b))                       == float(- a) / float(b);
/* */
/* definition of "+" by applying it to all constructors */
     (float(a) / float(b)) + (float(c) / float(d))  ==float(a*d + c*b) / float(b*d);
/* */
/* definition of binary "-" by other operations */
     r - s                                           == r + (- s);
/* */
/* definition of "*" by applying it to all constructors */
     (float(a) / float(b)) * (float(c) / float(d))  == float(a*c) / float(b*d);
/* */
/* definition of "<" by applying it to all constructors */
     b > 0 and d > 0 ==>
     (float(a) / float(b)) < (float(c) / float(d))  == a * d < c * b;
/* */
/* definition of ">", "equal", "<=", and ">=" by other operations */
     r > s      == s < r;
```

```
     equal(r, s) == not(r < s or r > s);
     r <= s      == r < s or r = s;
     r >= s      == r > s or r = s;
/* */
/* definition of fix by applying it to all constructors */
     a >= b and b > 0==> fix(float(a) / float(b)) == fix(float(a-b) / float(b)) + 1;
     b > a  and a >= 0  ==> fix(float(a) / float(b))== 0;
     a < 0  and b > 0==> fix(float(a) / float(b)) == - fix(float(-a)/float(b)) - 1;));
/* */
     for all r,s in Real nameclass (
     for all i,j in Integer nameclass (
        spelling(r) == spelling(i)          ==> r == float(i);
/* */
        spelling(r) == spelling(i)          ==> i == fix(r);
/* */
        spelling(r) == spelling(i) // spelling(s),
        spelling(s) == '.' // spelling(j) ==> r == float(i) + s;
/* */
        spelling(r) == '.' // spelling(i),
        length(spelling(i)) == 1            ==> r == float(i) / 10;
/* */
        spelling(r) == '.' // spelling(i) // spelling(j),
        length(spelling(i)) == 1,
        spelling(s) == '.' // spelling(j) ==> r == (float(i) + s) / 10;
     ));
endvalue type Real;
/*
```

D.3.7.2 Utilización

El género real se utiliza para representar números reales.

El género real comprende todos los números que pueden ser representados por un entero dividido por otro entero.

Los números que no puedan representarse de esta forma (números irracionales – por ejemplo la raíz cuadrada de 2) no pertenecen al género real. No obstante, en la ingeniería práctica, un número irracional generalmente puede ser aproximado por un número real con una exactitud suficiente.

D.3.8 Género array (array sort)

D.3.8.1 Definición

```
*/
value type Array < type Index; type Itemsort >
  operators
     make                                -> this Array ;
     make   ( Itemsort                 ) -> this Array ;
     modify ( this Array,Index,Itemsort ) -> this Array ;
     extract( this Array,Index          ) -> Itemsort raise InvalidIndex;
axioms
     for all item, itemi, itemj in Itemsort (
     for all i, j in Index (
     for all a, s in Array (
        <<type Array>>extract(make,i)                       == raise InvalidIndex;
        <<type Array>>extract(make(item),i)                 == item ;
        i = j ==> modify(modify(s,i,itemi),j,item)          == modify(s,i,item);
        i = j ==> extract(modify(a,i,item),j)               == item ;
        i = j == false ==> extract(modify(a,i,item),j)      == extract(a,j);
        i = j == false ==> modify(modify(s,i,itemi),j,itemj)==
                                                    modify(modify(s,j,itemj),i,itemi);
/*equality*/
        <<type Array>>make(itemi) = make(itemj)             == itemi = itemj;
        a=s == true, i=j == true, itemi = itemj ==>
                modify(a,i,itemi) = modify(s,j,itemj)       == true;
/* */
        extract(a,i) = extract(s,i) == false ==>  a = s     == false;)));
endvalue type Array;
/*
```

D.3.8.2 Utilización

Un array puede utilizarse para definir un género (sort) que es indizado por otro. Por ejemplo:

```
value type indexbychar inherits Array< Character, Integer >
endvalue type indexbychar;
```

define un array que contiene enteros y es indizado por caracteres.

Los arrays suelen utilizarse en combinación con las formas de notación taquigráfica de make, modify y extract definidas en 12.2.4 y 12.3.3.1. Por ejemplo:

```
dcl charvalue indexbychar;
task charvalue := (. 12 .);
task charvalue('A') := charvalue('B')-1;
```

D.3.9 Vector

D.3.9.1 Definición

```
*/
value type Vector < type Itemsort; synonym MaxIndex >
     inherits Array< Indexsort, Itemsort >;
syntype Indexsort = Integer constants 1:MaxIndex endsyntype;
endvalue type Vector;
/*
```

D.3.10 Género conjuntista (Powerset sort)

D.3.10.1 Definición

```
*/
value type Powerset < type Itemsort >
  operators
     empty                                    -> this Powerset;
     "in"   ( Itemsort, this Powerset       ) -> Boolean;       /* is member of          */
     incl   ( Itemsort, this Powerset       ) -> this Powerset; /* include item in set   */
     del    ( Itemsort, this Powerset       ) -> this Powerset; /* delete item from set  */
     "<"    ( this Powerset, this Powerset  ) -> Boolean;       /* is proper subset of   */
     ">"    ( this Powerset, this Powerset  ) -> Boolean;       /* is proper superset of */
     "<="   ( this Powerset, this Powerset  ) -> Boolean;       /* is subset of          */
     ">="   ( this Powerset, this Powerset  ) -> Boolean;       /* is superset of        */
     "and"  ( this Powerset, this Powerset  ) -> this Powerset; /* intersection of sets  */
     "or"   ( this Powerset, this Powerset  ) -> this Powerset; /* union of sets         */
     length ( this Powerset                 ) -> Integer;
     take   ( this Powerset                 ) -> Itemsort raise Empty;
axioms
     for all i,j in Itemsort (
     for all p,ps,a,b,c in Powerset (
/* constructors are empty and incl */
/* equalities between constructor terms */
     incl(i,incl(j,p))                    == incl(j,incl(i,p));
     i = j ==> incl(i,incl(j,p))          == incl(i,p);
/* definition of "in" by applying it to all constructors */
     i in <<type Powerset>>empty          == false;
     i in incl(j,ps)                      == i=j or i in ps;
/* definition of del by applying it to all constructors */
     <<type Powerset>>del(i,empty)        == empty;
     i =  j ==> del(i,incl(j,ps))       == del(i,ps);
     i /= j ==> del(i,incl(j,ps))       == incl(j,del(i,ps));
/* definition of "<" by applying it to all constructors */
     a < <<type Powerset>>empty         == false;
     <<type Powerset>>empty < incl(i,b)== true;
     incl(i,a) < b                      == i in b and del(i,a) < del(i,b);
/* definition of ">" by other operations */
     a > b == b < a;
/* definition of "=" by applying it to all constructors */
     empty = incl(i,ps)                 == false;
     incl(i,a) = b     == i in b and del(i,a) = del(i,b);
/* definition of "<=" and ">=" by other operations */
     a <= b                               == a < b or a = b;
     a >= b                               == a > b or a = b;
```

```
/* definition of "and" by applying it to all constructors */
     empty and b                          == empty;
     i in b     ==> incl(i,a) and b     == incl(i,a and b);
     not(i in b) ==> incl(i,a) and b      == a and b;
/* definition of "or" by applying it to all constructors */
     empty or b                           == b;
     incl(i,a) or b                       == incl(i,a or b);
/* definition of length */
     length(<<type Powerset>>empty)       == 0;
     i in ps ==> length(ps)               == 1 + length(del(i, ps));
/* definition of take */
     take(empty)                          == raise Empty;
     i in ps ==> take(ps)                 == i;
     ));
endvalue type Powerset;
/*
```

D.3.10.2 Utilización

Los Powersets se utilizan para representar conjuntos matemáticos. Por ejemplo:

```
value type Boolset inherits Powerset< Boolean > endvalue type Boolset;
```

puede utilizarse para una variable que puede estar vacía o contener (true), (false) o (true, false).

D.3.11 Género duración (Duration sort)

D.3.11.1 Definición

```
*/
value type Duration
  literals unordered nameclass ('0':'9')+ or (('0':'9')* '.' ('0':'9')+);
  operators
     protected duration ( Real           ) -> this Duration;
     "+"  ( this Duration, this Duration ) -> this Duration;
     "-"  ( this Duration                ) -> this Duration;
     "-"  ( this Duration, this Duration ) -> this Duration;
     ">"  ( this Duration, this Duration ) -> Boolean;
     "<"  ( this Duration, this Duration ) -> Boolean;
     ">=" ( this Duration, this Duration ) -> Boolean;
     "<=" ( this Duration, this Duration ) -> Boolean;
     "*"  ( this Duration, Real          ) -> this Duration;
     "*"  ( Real, this Duration          ) -> this Duration;
     "/"  ( this Duration, Real          ) -> Duration;
axioms noequality
/* constructor is duration(Real)*/
     for all a, b in Real nameclass (
     for all d, e in Duration nameclass (
/* definition of "+" by applying it to all constructors */
     duration(a) + duration(b)    == duration(a + b);
/* */
/* definition of unary "-" by applying it to all constructors */
     - duration(a)             == duration(-a);
/* */
/* definition of binary "-" by other operations */
     d - e                        == d + (-e);
/* */
/* definition of "equal", ">", "<", ">=", and "<=" by applying it to all constructors */
     equal(duration(a), duration(b)) == a = b;
     duration(a) > duration(b)    == a > b;
     duration(a) < duration(b)    == a < b;
     duration(a) >= duration(b) == a >= b;
     duration(a) <= duration(b) == a <= b;
/* */
/* definition of "*" by applying it to all constructors */
     duration(a) * b              == duration(a * b);
     a * d                        == d * a;
/* */
/* definition of "/" by applying it to all constructors */
     duration(a) / b           == duration(a / b);
/* */
```

```
     spelling(d) == spelling(a) ==>
                           d  == duration(a);
     ));
endvalue type Duration;
/*
```

D.3.11.2 Utilización

El género duración (duration sort) se utiliza para el valor que habrá de añadirse al tiempo actual para fijar temporizadores. Los literales del género duración son los mismos del género real. El significado de una unidad de duración dependerá del sistema que se defina.

Las duraciones pueden multiplicarse y dividirse por reales.

D.3.12 Género tiempo (Time sort)

D.3.12.1 Definición

```
*/
value type Time
  literals unordered nameclass ('0':'9')+ or (('0':'9')* '.' ('0':'9')+);
  operators
     protected time  ( Duration   ) -> this Time;
     "<"  ( this Time, this Time  ) -> Boolean;
     "<=" ( this Time, this Time  ) -> Boolean;
     ">"  ( this Time, this Time  ) -> Boolean;
     ">=" ( this Time, this Time  ) -> Boolean;
     "+"  ( this Time, Duration) -> this Time;
     "+"  ( Duration, this Time) -> this Time;
     "-"  ( this Time, Duration) -> this Time;
     "-"  ( this Time, this Time  ) -> Duration;
axioms noequality
/* constructor is time */
     for all t, u in Time nameclass (
     for all a, b in Duration nameclass (
/* definition of ">", "equal" by applying it to all constructors */
     time(a) > time(b)          == a > b;
     equal(time(a), time(b))    == a = b;
/* */
/* definition of "<", "<=", ">=" by other operations */
     t < u                      == u > t;
     t <= u                     == (t < u) or (t = u);
     t >= u                     == (t > u) or (t = u);
/* */
/* definition of "+" by applying it to all constructors */
     time(a) + b                == time(a + b);
     a + t                      == t + a;
/* */
/* definition of "-" : Time, Duration by other operations */
     t - b                      == t + (-b);
/* */
/* definition of "-" : Time, Time by applying it to all constructors */
     time(a) - time(b)          == a - b;
/* */
     spelling(a) == spelling(t) ==>
                           a  == time(t);
     ));
endvalue type Time;
/*
```

D.3.12.2 Utilización

La expresión **now** retorna un valor del género tiempo. Si a un valor de tiempo se le suma o se le resta una duración se obtiene otro valor de tiempo. Si a un valor de tiempo se le resta otro valor de tiempo se obtiene una duración. Se utilizan valores de tiempo para fijar el tiempo (o instante) de expiración de temporizadores.

El origen del tiempo depende del sistema. Una unidad de tiempo es la cantidad de tiempo representada por la adición de una unidad de duración a un tiempo.

D.3.13 Género bolsa (Bag sort)

D.3.13.1 Definición

```
*/
value type Bag < type Itemsort >
  operators
     empty                          -> this Bag;
     "in"   ( Itemsort, this Bag  ) -> Boolean;/* is member of  */
     incl   ( Itemsort, this Bag  ) -> this Bag;  /* include item in set     */
     del    ( Itemsort, this Bag  ) -> this Bag;  /* delete item from set    */
     "<"    ( this Bag, this Bag  ) -> Boolean;/* is proper subbag of       */
     ">"    ( this Bag, this Bag  ) -> Boolean;/* is proper superbag of     */
     "<="   ( this Bag, this Bag  ) -> Boolean;/* is subbag of  */
     ">="   ( this Bag, this Bag  ) -> Boolean;/* is superbag of            */
     "and"  ( this Bag, this Bag  ) -> this Bag;  /* intersection of bags    */
     "or"   ( this Bag, this Bag  ) -> this Bag;  /* union of bags           */
     length ( this Bag            ) -> Integer;
     take   ( this Bag            ) -> Itemsort raise Empty;
axioms
     for all i,j in Itemsort (
     for all p,ps,a,b,c in Bag (
/* constructors are empty and incl */
/* equalities between constructor terms */
     incl(i,incl(j,p))              == incl(j,incl(i,p));
/* definition of "in" by applying it to all constructors */
     i in <<type Bag>>empty         == false;
     i in incl(j,ps)                == i=j or i in ps;
/* definition of del by applying it to all constructors */
     <<type Bag>>del(i,empty)       == empty;
     i =  j ==> del(i,incl(j,ps))  == ps;
     i /= j ==> del(i,incl(j,ps))  == incl(j,del(i,ps));
/* definition of "<" by applying it to all constructors */
     a < <<type Bag>>empty          == false;
     <<type Bag>>empty < incl(i,b)  == true;
     incl(i,a) < b                 == i in b and del(i,a) < del(i,b);
/* definition of ">" by other operations */
     a > b == b < a;
/* definition of "=" by applying it to all constructors */
     empty = incl(i,ps)            == false;
     incl(i,a) = b                 == i in b and del(i,a) = del(i,b);
/* definition of "<=" and ">=" by other operations */
     a <= b                         == a < b or a = b;
     a >= b                         == a > b or a = b;
/* definition of "and" by applying it to all constructors */
     empty and b                    == empty;
     i in b     ==> incl(i,a) and b == incl(i,a and b);
     not(i in b) ==> incl(i,a) and b == a and b;
/* definition of "or" by applying it to all constructors */
     empty or b                     == b;
     incl(i,a) or b                 == incl(i,a or b);
/* definition of length */
     length(<<type Bag>>empty)      == 0;
     i in ps    ==> length(ps)      == 1 + length(del(i, ps));
/* definition of take */
     take(empty)                    == raise Empty;
     i in ps    ==> take(ps)        == i; ));
endvalue type Bag;
/*
```

D.3.13.2 Utilización

Se utilizan bolsas (bags) para representar multiconjuntos. Por ejemplo:

```
value type Boolset Bag< Boolean > endvalue type Boolset;
```

puede utilizarse para una variable que puede estar vacía o contener (true), (false), (true, false) (true, true), (false, false),... .

Se utilizan bolsas para representar la construcción SET OF de ASN.1.

D.3.14 Géneros bit (Bit) y cadena de bits (Bitstring) de la notación ASN.1

D.3.14.1 Definición

```
*/
value type Bit
     inherits Boolean ( 0 = false, 1 = true );
     adding
  operators
     num ( this Bit  ) -> Integer;
     bit ( Integer) -> this Bit raise OutOfRange;
axioms
     <<type Bit>>num (0)          == 0;
     <<type Bit>>num (1)          == 1;
     <<type Bit>>bit (0)          == 0;
     <<type Bit>>bit (1)          == 1;
     for all i in Integer (
        i > 1 or i < 0 ==> bit (i)== raise OutOfRange;
        )
endvalue type Bit;
/* */
value type Bitstring
  operators
     bs in nameclass
     '''' ( (('0' or '1')*'''B') or ((('0':'9') or ('A':'F'))*'''H') )-> this Bitstring;
/*The following operators that are the same as String except Bitstring
     is indexed from zero*/
     mkstring  (Bit                              ) -> this Bitstring;
     make      (Bit                              ) -> this Bitstring;
     length    ( this Bitstring                  ) -> Integer;
     first     ( this Bitstring                  ) -> Bit;
     last      ( this Bitstring                  ) -> Bit;
     "//"      ( this Bitstring, this Bitstring  ) -> this Bitstring;
     extract   ( this Bitstring, Integer         ) -> Bit raise InvalidIndex;
     modify    ( this Bitstring, Integer, Bit    ) -> this Bitstring;
     substring ( this Bitstring, Integer, Integer) -> this Bitstring raise InvalidIndex;
     /* substring (s,i,j) gives a string of length j starting from the ith element */
     remove    ( this Bitstring, Integer, Integer) -> this Bitstring;
     /* remove (s,i,j) gives a string with a substring of length j starting from
        the ith element removed */
/*The following operators are specific to Bitstrings*/
     "not"( this Bitstring                 ) -> this Bitstring;
     "and"( this Bitstring, this Bitstring ) -> this Bitstring;
     "or" ( this Bitstring, this Bitstring ) -> this Bitstring;
     "xor"( this Bitstring, this Bitstring ) -> this Bitstring;
     "=>" ( this Bitstring, this Bitstring ) -> this Bitstring
     num  ( this Bitstring                 ) -> Integer;
     bitstring ( Integer                   ) -> this Bitstring raise OutOfRange;
     octet ( Integer                       ) -> this Bitstring raise OutOfRange;
axioms
/* Bitstring starts at index 0 */
/* Definition of operators with the same names as String operators*/
     for all b in Bit ( /*b is bit in string*/
     for all s,s1,s2,s3 in Bitstring (
     for all i,j in Integer (
/* constructors are ''B, mkstring, and "//" */
/* equalities between constructor terms */
     s // ''B          == s;
     ''B// s           == s;
     (s1 // s2) // s3== s1 // (s2 // s3);
/* definition of length by applying it to all constructors */
     <<type Bitstring>>length(''B)               == 0;
     <<type Bitstring >>length(mkstring(b))      == 1;
     <<type Bitstring >>length(s1 // s2)         == length(s1) + length(s2);
     make(s)                                     == mkstring(s);
/* definition of extract by applying it to all constructors,
        with error cases handled separately */
     extract(mkstring(b),0)                      == b;
     i < length(s1)      ==> extract(s1 // s2,i) == extract(s1,i);
     i >= length(s1)     ==> extract(s1 // s2,i) == extract(s2,i-length(s1));
     i<0 or i=>length(s)  ==> extract(s,i)       == raise InvalidIndex;
```

```
/* definition of first and last by other operations */
     first(s)  == extract(s,0);
     last(s)   == extract(s,length(s)-1);
/* definition of substring(s,i,j) by induction on j,
        error cases handled separately */
     i>=0 and i < length(s)            ==>
          substring(s,i,0)  == ''B;
/* */
     i>=0 and j>0 and i+j<=length(s) ==>
          substring(s,i,j)  == substring(s,i,j-1) // mkstring(extract(s,i+j));
/* */
     i<0 or j<0 or i+j>length(s)       ==>
          substring(s,i,j)  == raise InvalidIndex;
/* */
/* definition of modify by other operations */
     modify(s,i,b)== substring(s,0,i) // mkstring(b) // substring(s,i+1,length(s)-i);
/* definition of remove */
     remove(s,i,j)   == substring(s,0,i) // substring(s,i+j,length(s)-i-j);
     )));
/*end of definition of string operators indexed from zero*/
/* */
/* Definition of ''H and 'x'H in terms of ''B, 'xxxx'B for Bitstring*/
     <<type Bitstring>>''H  == ''B;
     <<type Bitstring>>'0'H == '0000'B;
     <<type Bitstring>>'1'H == '0001'B;
     <<type Bitstring>>'2'H == '0010'B;
     <<type Bitstring>>'3'H == '0011'B;
     <<type Bitstring>>'4'H == '0100'B;
     <<type Bitstring>>'5'H == '0101'B;
     <<type Bitstring>>'6'H == '0110'B;
     <<type Bitstring>>'7'H == '0111'B;
     <<type Bitstring>>'8'H == '1000'B;
     <<type Bitstring>>'9'H == '1001'B;
     <<type Bitstring>>'A'H == '1010'B;
     <<type Bitstring>>'B'H == '1011'B;
     <<type Bitstring>>'C'H == '1100'B;
     <<type Bitstring>>'D'H == '1101'B;
     <<type Bitstring>>'E'H == '1110'B;
     <<type Bitstring>>'F'H == '1111'B;
/* */
/* Definition of Bitstring specific operators*/
     <<type Bitstring>>mkstring(0)          == '0'B;
     <<type Bitstring>>mkstring(1)          == '1'B;
/* */
     for all s, s1, s2, s3 in Bitstring (
     s  =  s                                == true;
     s1 =  s2                               == s2 = s1;
     s1 /= s2                               == not ( s1 = s2 );
     s1 =  s2 == true            ==> s1     == s2;
     ((s1 = s2) and (s2 = s3))  ==> s1 = s3  == true;
     ((s1 = s2) and (s2 /= s3)) ==> s1 = s3  == false;
/* */
     for all b, b1, b2 in Bit (
     not(''B)                       == ''B;
     not(mkstring(b) // s)          == mkstring( not(b) ) // not(s);
/* definition of or */
/* The length of or-ing two strings is the maximal length of both strings */
     ''B or ''B                     == ''B;
     length(s) > 0 ==> ''B or s == mkstring(0) or s;
     s1 or s2                       == s2 or s1;
     (b1 or b2) // (s1 or s2)       ==(mkstring(b1) // s1) or (mkstring(b2) // s2);
/* */
/* definition of remaining operators based on "or" and "not" */
     s1 and s2                      == not (not s1 or not s2);
     s1 xor s2                      == (s1 or s2) and not(s1 and s2);
     s1 => s2                       == not (s1 and s2);
     ));
/* */
/*Definition of 'xxxxx'B literals */
     for all s in Bitstring (
     for all b in Bit (
```

```
     for all i in Integer (
     <<type Bitstring>>num (''B)           == 0;
     <<type Bitstring>>bitstring (0)     == '0'B;
     <<type Bitstring>>bitstring (1)     == '1'B;
     num (s // mkstring (b))             == num (b) + 2 * num (s);
     i > 1               ==> bitstring (i) == bitstring (i / 2) // bitstring (i mod 2);
     i >= 0 and i <= 255  ==> octet (i)   == bitstring (i) or '00000000'B;
     i < 0               ==> bitstring (i) == raise OutOfRange;
     i < 0 or i > 255  ==> octet (i)   == raise OutOfRange;
     )))
/*Definition of 'xxxxx'H literals */
     for all b1,b2,b3,h1,h2,h3 in bs nameclass (
     for all bs1, bs2, bs3, hs1, hs2, hs3 in Charstring (
     spelling(b1) = '''' // bs1 // '''B',
     spelling(b2) = '''' // bs2 // '''B',
     bs1 /= bs2                          ==> b1 = b2    == false;
/* */
     spelling(h1) = '''' // hs1 // '''H',
     spelling(h2) = '''' // hs2 // '''H',
     hs1 /= hs2 ==> h1 = h2 == false;
     spelling(b1) = '''' // bs1 // '''B',
     spelling(b2) = '''' // bs2 // '''B',
     spelling(b3) = '''' // bs1 // bs2 // '''B',
     spelling(h1) = '''' // hs1 // '''H',
     spelling(h2) = '''' // hs2 // '''H',
     spelling(h3) = '''' // hs1 // hs2 // '''H',
     length(bs1) = 4,
     length(hs1) = 1,
     length(hs2) > 0,
     length(bs2) = 4 * length(hs2),
     h1 = b1                             ==> h3 = b3    == h2 = b2;
/* */
/* connection to the String generator */
     for all b in Bit literals (
     spelling(b1) = '''' // bs1 // bs2 // '''B',
     spelling(b2) = '''' // bs2 // '''B',
     spelling(b)  = bs1                  ==> b1         == mkstring(b) // b2;
     )));
endvalue type Bitstring;
/*
```

D.3.15 Géneros octeto (Octet) y cadena de octetos (Octetstring) de la notación ASN.1

D.3.15.1 Definición

```
*/
syntype Octet = Bitstring size (8)
endsyntype Octet;
/* */
value type Octetstring
     inherits String < Octet > ( ''B = emptystring )
     adding
     operators
     os in nameclass
          '''' ( ((('0' or '1')8)*'''B') or ((((('0':'9') or ('A':'F'))2)*'''H') )
           -> this Octetstring;
     bitstring   ( this Octetstring ) -> Bitstring;
     octetstring ( Bitstring ) -> this Octetstring;
axioms
     for all b,b1,b2 in Bitstring (
     for all s in Octetstring (
     for all o in Octet(
     <<type Octetstring>> bitstring(''B)           == ''B;
     <<type Octetstring>> octetstring(''B)         == ''B;
     bitstring( mkstring(o) // s )                 == o // bitstring(s);
/* */
     length(b1) > 0,
     length(b1) < 8,
     b2 == b1 or '00000000'B ==> octetstring(b1)  == mkstring(b2);
```

```
/* */
     b == b1 // b2,
     length(b1) == 8           ==> octetstring(b)== mkstring(b1) // octetstring(b2);
     )));
/* */
     for all b1, b2 in Bitstring (
     for all o1, o2 in os nameclass (
     spelling( o1 ) = spelling( b1 ),
     spelling( o2 ) = spelling( b2 ) ==> o1 = o2 == b1 = b2
     ));
endvalue type Octetstring;
/*
```

D.3.16 Excepciones predefinidas (Predefined exceptions)

```
*/
exception
     OutOfRange,          /* A range check has failed. */
     InvalidReference,    /* Null was used incorrectly. Wrong Pid for this signal. */
     NoMatchingAnswer,    /* No answer matched in a decision without else part. */
     UndefinedVariable, /* A variable was used that is “undefined”. */
     UndefinedField,      /* An undefined field of a choice or struct was accessed. */
     InvalidIndex,     /* A String or Array was accessed with an incorrect index. */
     DivisionByZero;      /* An Integer or Real division by zero was attempted. */
     Empty;               /* No element could be returned. */
/* */
endpackage Predefined;
```

ANEXO E

Reservado para ejemplos

ANEXO F

Definición formal

Se requieren estudios adicionales para el SDL-2000.

APÉNDICE I

Estado de la Recomendación Z.100, documentos y Recomendaciones conexos

Este apéndice contiene una lista del estado de todos los documentos relacionados con el SDL publicados por el UIT-T. La lista incluye todas las partes de las Recomendaciones Z.100, Z.105, Z.106, Z.107, Z.109 y cualquier documento conexo referente a la metodología. También se enumeran otros documentos relevantes tales como la Recomendación Z.110.

Esta lista será actualizada por los medios adecuados (por ejemplo, mediante un corrigendum) siempre que se acuerden cambios en el SDL y se aprueben nuevos documentos.

SDL-2000 se define en las siguientes Recomendaciones aprobadas por la Comisión de Estudio 10 del UIT-T el 19 de noviembre de 1999.

Z.100 (11/99) – Lenguaje de Especificación y Descripción (SDL).

Anexo A a Z.100 – Índice de no terminales.

Anexo D a Z.100 – Datos predefinidos SDL.

El anexo F está previsto para noviembre de 2000.

No hay planes específicos sobre la fecha de aprobación de los anexos B, C y E.

Suplemento 1 a Z.100 (05/97): Metodología SDL+. *Actualización de SDL-2000 en curso*.

Z.105 (11/99) SDL combinado con módulos ASN.1.

Z.106 (10/96) Formato de intercambio común para SDL. *Actualización de SDL-2000 en curso*.

Z.107 (11/99) SDL con ASN.1 insertada.

Z.109 (11/99) SDL combinado con UML.

Z.110 (10/96) Criterios para la utilización de técnicas de descripción formal por el UIT-T. *Actualización prevista para noviembre de 2000*.

Se puede obtener más información sobre SDL, así como también sobre libros y otras publicaciones, a través de: http://www.sdl-forum.org.

APÉNDICE II

Directrices para el mantenimiento del SDL

II.1 Mantenimiento de SDL

En esta subcláusula se describen la terminología y las reglas para el mantenimiento de SDL acordadas en la reunión de la Comisión de Estudio 10 de noviembre de 1993, y el "procedimiento de petición de cambio" asociado.

II.1.1 Terminología

a) Un *error* es una inconsistencia interna en Z.100.

b) Una *corrección textual* es un cambio en el texto o diagramas de Z.100 que corrige errores clericales o tipográficos.

c) Un *ítem abierto* es un elemento concernido identificado pero no resuelto. Un ítem abierto puede identificarse por una petición de cambio o por un acuerdo con la Comisión de Estudio o el Grupo de Trabajo.

d) Una *deficiencia* es un aspecto identificado en el que la semántica de SDL no está (claramente) definida por Z.100.

e) Una *clarificación* (o una *aclaración)* es un cambio en el texto o los diagramas de Z.100 que clarifica texto o diagramas anteriores cuya comprensión pueden presentar ambigüedades sin la clarificación. La clarificación debe intentar que Z.100 se corresponda con la semántica SDL tal como la entiende la Comisión de Estudio o el Grupo de Trabajo.

f) Una *modificación* es un cambio en el texto o los diagramas de Z.100 que altera la semántica SDL.

g) Una *característica descomprometida* es una característica de SDL que debe eliminarse de SDL en la siguiente revisión de Z.100.

h) Una *ampliación* (o una *extensión*) es una nueva característica que no debe modificar la semántica de las características definidas en Z.100.

II.1.2 Reglas de mantenimiento

En el texto siguiente se debe considerar que las referencias a Z.100 incluyen los anexos, apéndices y suplementos, así como cualquier Addendum, Modificación, Corrigendum o Guía de Implementación y los mismos textos incluidos en las Recomendaciones Z.105, Z.106, Z.107 y Z.109.

a) Cuando se detecta un error o deficiencia en Z.100, éste debe ser corregido o clarificado. La corrección de un error debe traer consigo el menor número de cambios posibles. Las correcciones de errores y clarificaciones deben incluirse en la Lista Maestra de Cambios de Z.100 y ser efectivas con carácter inmediato.

b) Excepto para correcciones de errores y resoluciones de ítems abiertos procedentes del periodo de estudio anterior, las modificaciones y las ampliaciones sólo deben considerarse como resultado de una petición de cambio que esté apoyada por una comunidad de usuarios significativa. Una petición de cambio debe ir seguida de una investigación por la Comisión de Estudio o por el Grupo de Trabajo en colaboración con los representantes del grupo de usuarios, de forma que se establezcan con claridad la necesidad y los beneficios de la misma y la certeza de que una característica existente de SDL es inadecuada.

c) Las modificaciones y ampliaciones obtenidas por medios distintos de la corrección de errores, deben ser dadas a conocer ampliamente, y los puntos de vista de los usuarios y de los implementadores de herramientas debe ser analizados antes de adoptar el cambio. Salvo que existan circunstancias especiales que requieran la implementación de dichos cambios cuanto antes, dichos cambios no serán recomendados hasta la siguiente revisión de Z.100.

d) Hasta que se publique una versión revisada de Z.100, se mantendrá una Lista Maestra de Cambios que abarque a Z.100 y todos los anexos excepto la definición formal. Los Apéndices, Addenda, Corrigenda, Guías de Implementación o Suplementos se publicarán tal como decida la Comisión de Estudios. Para asegurar la distribución efectiva de la Lista Maestra de Cambios de Z.100, ésta se publicará como Informes COM y por los medios electrónicos adecuados.

e) En caso de deficiencias de Z.100, debe consultarse la definición formal. Ello puede producir una clarificación o una corrección que se registre en la Lista Maestra de Cambios de Z.100.

II.1.3 Procedimiento de petición de cambio

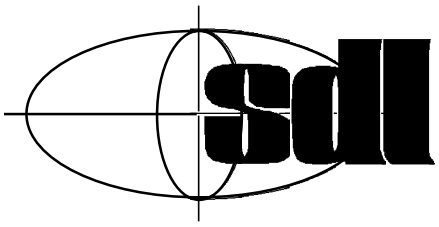
El procedimiento de petición de cambio está diseñado para permitir que usuarios de SDL, dentro y fuera del UIT-T, puedan formular preguntas sobre el significado preciso de Z.100, sugerir cambios al SDL o a Z.100, y proporcionar información sobre cambios que hayan sido propuestos al SDL. El Grupo de Expertos en SDL publicará los cambios propuestos al SDL antes de que éstos sean implementados.

Para las peticiones de cambios se debe utilizar el Formulario de Petición de Cambio (*Change Request Form*) (véase a continuación) o proporcionar la información que se enumera en dicho formulario. Debe indicarse claramente la clase de petición de que se trata (corrección de error, clarificación, simplificación, ampliación, modificación o característica descomprometida). También es importante que para cualquier cambio distinto a una corrección de error, se indique el grado de apoyo que recibe el usuario para la petición en cuestión.

Las reuniones de la Comisión de Estudio responsable de Z.100 deben analizar todos los cambios propuestos. Para correcciones o clarificaciones, los cambios pueden ponerse en la lista de correcciones sin consultar a los usuarios. En otros casos, se compila una lista de ítems abiertos. La información debe distribuirse a los usuarios:

- mediante informes presentados como contribuciones blancas del UIT-T;
- por correo electrónico a las listas de correo SDL (como por ejemplo, la lista informal del UIT-T, y sdlnews@sdl-forum.org);
- otras formas que acuerden los expertos de la Comisión de Estudio 10.

Los expertos de la Comisión de Estudio deben determinar el nivel de apoyo y de oposición que recibe cada modificación y evaluar la reacción de los usuarios. Una modificación sólo se incluirá en la lista de modificaciones si existe un apoyo significativo de los usuarios y no existen objeciones de peso por más de unos pocos usuarios. Finalmente, todos los cambios que se hayan aceptado se incorporarán a la versión revisada de Z.100. Los usuarios deben ser conscientes que mientras los cambios no hayan sido aprobados e incorporados por parte de la Comisión de Estudio responsable de Z.100, éstos no pueden ser considerados como recomendados por parte del UIT-T.



Formulario de petición de cambios

Sírvase rellenar los siguientes datos		
Tipo de cambio:	❑ corrección de errores ❑ simplificación ❑ modificación	❑ aclaración ❑ ampliación ❑ anulación
Breve resumen de la petición de cambios		
Resumen de los motivos que justifican la petición de cambios		
¿Su organización comparte esta opinión?	❑ sí	❑ no
¿Ha consultado con otros usuarios?	❑ sí	❑ no
¿A cuántos usuarios representa?	❑ 1-5 ❑ 11-100	❑ 6-10 ❑ más de 100
Su nombre y dirección		

sírvase adjuntar, en caso necesario, más hojas con información detallada

SDL (Z.100) Relator, c/o UIT-T, Place des Nations, CH-1211, Ginebra 20, Suiza.
Fax: +41 22 730 5853, correo-e: SDL.rapporteur@itu.int

APÉNDICE III

Conversión sistemática de SDL-92 en SDL-2000

Aunque no todas las especificaciones SDL-92 pueden convertirse automáticamente en SDL-2000, en muchos casos puede ser suficiente una simple transformación. El procedimiento siguiente asume un procesamiento de conversión SDL/PR; también es posible un procedimiento similar de transformación SDL/GR:

1) Ortografía correcta en relación con las mayúsculas/minúsculas y con las nuevas palabras clave:

   a) Sustituir todas las palabras clave por las correspondientes <lowercase keyword> (o <uppercase keyword>);

   b) Sustituir todas las <word>s que contienen caracteres <national> definidos en Z.100 (03/93) por una <word> única;

   c) Sustituir todos los <name>s por sus equivalentes en minúsculas;

   d) Si un <name>s entra en conflicto con <lowercase keyword>s, sustituir el primer carácter por una mayúscula.

   En muchos casos es posible utilizar un procedimiento menos exigente, por ejemplo, utilizando siempre la ortografía del <name> que define la ocurrencia de su correspondiente <identifier>. Ello provoca un cambio semántico únicamente si se modifica el nombre de un estado y dicho nombre se utiliza en una <state expression>, tal como se indica en el addendum 1 a SDL-92.

2) Sustituir todos los <qualifier>s por el correspondiente <qualifier> de SDL-2000 (es decir, la lista de ítems de trayecto está siempre incluida en las <composite special>s <qualifier begin sign> y <qualifier end sign>).

3) Transformar toda la utilización de las palabras clave fpar y return en <agent formal parameters>, <procedure formal parameters>, <procedure return>, <macro formal parameter>, <formal operation parameters> y <procedure signature> en la correspondiente sintaxis SDL-2000.

4) Sustituir todas las rutas de señales con nodelay <channel definition>s.

5) En los bloques que no tengan rutas de señales o canales, añadir puertas a todo proceso enumerando las señales que dicho proceso transmite o recibe. Alternativamente, añadir canales implícitos de acuerdo con el modelo para señales y rutas implícitas de SDL-92. Las especificaciones que se basan en canales implícitos tal como indica el addendum 1 deben también añadir puertas a los respectivos bloques.

6) Sustituir todas las ocurrencias de particiones de bloque. SDL-92 no especificaba cómo se seleccionaba un subconjunto consistente, por lo que este paso puede exigir conocimiento externo. Una conversión que asuma que siempre debe seleccionarse la subestructura, reflejaría probablemente una utilización típica de SDL-92.

   a) Mover todos los bloques de la subestructura directamente al bloque contenedor.

   b) Si existen conflictos en entidades del bloque y en entidades de la subestructura, redenominar una de las entidades con un nombre único.

   c) Ajustar todos los identificadores para entidades en bloques anidados para utilizar un nuevo calificador.

7) Sustituir todas las acciones de salida utilizando via all con una lista de acciones de salida. Si el <via path> fuera un canal entre conjuntos de ejemplares de bloques, no es posible la transformación automática.

8) Sustituir servicio y tipos de servicio por estado compuesto y tipos de estado compuesto, respectivamente. Si los servicios tienen acciones de entrada que se superponen (incluso si sus conjuntos de entradas válidas estaban disjuntos), hay que suprimir la de las transiciones duplicadas. Suprimir todas las rutas de señales entre servicios; la salida referente a estas rutas de señales debe hacer referencia a puertas del tipo de proceso. Sustituir <stop> por <return>. Los temporizadores, procedimientos exportados y variables exportadas de un servicio tienen que estar definidas en agente.

9) Sustituir todas las definiciones de tipos de datos que impliquen transformaciones de generadores por las definiciones equivalentes que emplean tipos parametrizados.

10) La transformación de axiomas de datos no es posible de forma automática, pero sólo ha habido unos pocos usuarios que hayan definido axiomáticamente sus propios tipos de datos. No obstante, los casos siguientes pueden transformarse fácilmente:

   a) Expresiones de datos predefinidos que permanecen válidos incluyendo la utilización de cadena (String), matriz (Array) y conjuntista (PowerSet), después de ajustes específicos en la ortografía de sus tipos.

   b) Una definición de newtype con literales (y operadores no definidos axiomáticamente) puede convertirse en una <value data type definition> con <literal list>.

   c) Una definición de newtype con propiedad de estructura puede convertirse en una <value data type definition> con <structure definition>.

Si una especificación SDL-92 utiliza constructivos que no pueden convertirse automáticamente en constructivos equivalentes SDL-2000, es necesaria una inspección cuidadosa de dicha especificación en caso de que ésta deba ser conforme con esta Recomendación.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Transmisiones de señales radiofónicas, de televisión y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos y comunicación entre sistemas abiertos
Serie Y	Infraestructura mundial de la información y aspectos del protocolo Internet
Serie Z	**Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación**

18357

Impreso en Suiza
Ginebra, 2000