UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Z.105

(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 combinado con módulos de la notación de sintaxis abstracta uno

Recomendación UIT-T Z.105

(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.105

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN COMBINADO CON MÓDULOS DE LA NOTACIÓN DE SINTAXIS ABSTRACTA UNO

Resumen

Objetivo

La presente Recomendación define cómo se pueden utilizar los módulos de la notación de sintaxis abstracta uno (ASN.1, *abstract syntax notation one*) en combinación con el lenguaje de especificación y descripción (SDL, *specification and description language*). El propósito es que la estructura y el comportamiento de los sistemas se describan con el SDL, mientras que los parámetros de los mensajes intercambiados se describan con la ASN.1. Esta Recomendación define una correspondencia de construcciones de ASN.1 con construcciones de SDL ya existentes y contiene solamente una pequeña ampliación de la Recomendación Z.100 para poder utilizar los módulos ASN.1.

Alcance

Esta Recomendación presenta una definición de semántica para la combinación de SDL y módulos ASN.1. Se proporciona una correspondencia de los datos ASN.1 definidos en un módulo con las correspondientes construcciones SDL definidas en la Recomendación Z.100 [1], incluidos los operadores que pueden ser aplicados a los datos ASN.1. Los ítems de datos ASN.1 pueden ser utilizados dentro del SDL (empleando la notación SDL).

El uso de la notación ASN.1 incorporada en el SDL se define en la Recomendación Z.107 [2].

Aplicación

La principal esfera de aplicación de la presente Recomendación es la especificación de sistemas de telecomunicaciones. La utilización combinada de SDL y ASN.1 permite especificar de manera coherente la estructura y el comportamiento de los sistemas de telecomunicaciones, junto con los datos, mensajes y codificación de mensajes que estos sistemas utilizan.

NOTA – En la presente Recomendación el término "especificación" incluye la definición de requisitos en una norma, Recomendación o documento de adquisición, y descripción de una implementación.

Una especificación es conforme a la presente Recomendación solamente si es conforme a las reglas gramaticales sintácticas y semánticas para el lenguaje técnico formal definido por la Recomendación (que incluye los lenguajes ASN.1 y SDL mencionados). La conformidad entraña que cada posible interpretación dinámica de la especificación sea conforme a las reglas del lenguaje. Una especificación que utiliza extensiones del lenguaje no es conforme.

Una herramienta no soporta totalmente el lenguaje si rechaza algunas construcciones del lenguaje o tiene una interpretación estática o dinámica de una especificación en el lenguaje que no se conforma con la semántica del lenguaje.

Estado/estabilidad

La presente Recomendación sustituye las correspondencias semánticas de la ASN.1 con el SDL definidas en la Recomendación Z.105 (l995). El uso de la notación ASN.1 insertada en el SDL previamente definido en la Recomendación Z.105 (l995) no se define en la presente Recomendación.

Las modificaciones de las Recomendaciones X.680 [3], X.681 [4], X.682 [5] y X.683 [6] o Z.100 [1] pueden requerir que se enmiende la presente Recomendación.

Esta Recomendación es el manual de referencia completo que describe la combinación de SDL y módulos ASN.1.

Trabajo conexo

Recomendación UIT Z.100 (1999), *Lenguaje de especificación y descripción*.

Recomendación UIT.T X.680 (1997), *Especificación de la notación básica.*

Recomendación UIT.T X.681 (1997), *Especificación de objetos de información*.

Recomendación UIT.T X.682 (1997), *Especificación de constricciones*.

Recomendación UIT.T X.683 (1997), *Parametrización de especificaciones ASN.1*.

Recomendación UIT.T Z.107 (1999), *SDL con ASN.1 incorporada*.

Orígenes

La Recomendación UIT-T Z.105 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.

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

Recomendación Z.105

LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN COMBINADO CON MÓDULOS DE LA NOTACIÓN DE SINTAXIS ABSTRACTA UNO

(revisada en 1999)

1 Introducción

La presente Recomendación define cómo los módulos de la notación de sintaxis abstracta uno (ASN.1, *abstract syntax notation one*) pueden ser utilizados en combinación con el lenguaje de especificación y descripción (SDL, *specification and description language*). Los módulos ASN.1 son importados en descripciones SDL de modo que las definiciones de datos ASN.1 correspondan con la representación SDL interna utilizando construcciones SDL equivalentes y formando junto con el resto de la descripción SDL una especificación completa.

El SDL es un lenguaje para la especificación y descripción de sistemas de telecomunicaciones, que tiene conceptos para:

- estructurar sistemas;
- definir el comportamiento de sistemas;
- definir los datos utilizados por los sistemas.

La ASN.1 es un lenguaje para la definición de datos. Guardan relación con la ASN.1 las reglas de codificación, que definen cómo se transfieren valores ASN.1 como trenes de bits durante la comunicación.

1.1 Objetivo

La combinación de SDL y ASN.1 permite especificar de manera coherente la estructura y el comportamiento de sistemas de telecomunicación, junto con los datos, mensajes y codificación de mensajes que estos sistemas utilizan. La estructura y el comportamiento pueden ser descritos con el SDL, y los datos y mensajes con la ASN.1. La codificación de estos mensajes puede ser descrita por referencia a las reglas de codificación pertinentes definidas para la ASN.1.

Esta Recomendación admite la plena utilización del SDL (incluidos los tipos de datos).

1.2 Características de la combinación de SDL y ASN.1

Los sistemas descritos en el SDL combinado con módulos ASN.1 tienen las siguientes características:

- la estructura y el comportamiento se definen utilizando conceptos SDL;
- los parámetros de señales son definidos por tipos ASN.1;
- los datos utilizados en señales son definidos con definiciones de tipos ASN.1;
- los datos internos pueden ser definidos con tipos ASN.1 o géneros SDL;
- la codificación de valores de datos ASN.1 se puede definir mediante una referencia a las reglas de codificación pertinentes. La codificación está fuera del ámbito de la presente Recomendación.

1.3 ASN.1 que puede ser utilizada en combinación con el SDL

La ASN.1, según se define en las Recomendaciones X.680, X.681. X.682 y X.683, se puede soportar en combinación con el SDL, reconociendo que algunas construcciones ASN.1 no pueden corresponder satisfactoriamente con el SDL (o al menos la correspondencia no ha sido identificada ni especificada en la presente Recomendación). Las construcciones que no pueden tener correspondencia en el SDL existirán en lotes ASN.1 utilizados como una fuente de transformación. Durante la transformación en SDL son tratadas efectivamente como si no estuviesen presentes y no deben plantear problemas para la transformación satisfactoria de otras construcciones. Estas construcciones son el marcador de extensión y el marcador de excepción definidos en la Recomendación X.680, que pueden estar presentes en la ASN.1 pero que son pasados por alto en la transformación en SDL. Partes de la gramática ASN.1 (1997) relacionadas con el marcador de extensión y excepción ya no se utilizan en esta Recommendación. Algunas construcciones de la ASN.1 no son transformadas nunca en SDL como tales, pero contienen información que puede ser usada en la transformación. Los ejemplos prominentes de estas construcciones son la restricciones relacionales definidas en la Recomendación X.682, clases de objeto y conjuntos de objetos.

Se soporta la utilización de SDL, según se define en la Recomendación Z.100 [1].

Los módulos ASN.1 utilizados en la transformación en SDL pueden ser empleados también para la generación de codificadores y decodificadores, a condición de que se definan reglas de codificación. No se debe utilizar la especificación de datos SDL derivados de módulos ASN.1 para este fin porque al efectuar la transformación en SDL se puede perder cierta información que es pertinente para la codificación.

1.4 Estructura de esta Recomendación

La presente Recomendación no es independiente: la correspondencia definida en esta Recomendación se basa en la Recomendación Z.100 y en las Recomendaciones X.680, X.681, X.682 y X.683. Se aplica el lenguaje definido en la Recomendación Z.100, con la excepción de que la regla de producción <package> (lote) se amplía para permitir el uso directo de módulos ASN.1. La presente Recomendación está estructurada de la siguiente manera:

La cláusula 3 define las modificaciones de la Recomendación Z.100 para incorporar los módulos ASN.1.

La cláusula 4 define la correspondencia de los tipos y valores ASN.1 de la Recomendación X.680 con datos de la Recomendación Z.100 para incorporar los tipos de datos y valores ASN.1.

La cláusula 5 define la correspondencia de tipos ASN.1 definidos que utilizan objetos de información, clases y conjuntos de objetos de información. El uso de las construcciones X.682 se trata también en esta cláusula.

La cláusula 6 define la correspondencia de tipos ASN.1 parametrizados con datos de la Recomendación Z.100 para incorporar tipos de datos ASN.1 parametrizados.

La cláusula 7 define las adiciones necesarias al lote Predefined para sustentar el uso de la ASN.1.

1.5 Convenios utilizados en la presente Recomendación

Se aplican básicamente los convenios de la Recomendación Z.100, es decir, las palabras clave aparecen con minúsculas en negritas, los nombres predefinidos comienzan con letra mayúscula. Sin embargo, en los ejemplos ASN.1 se utilizan convenios de ASN.1 para respetar las reglas de la ASN.1 y mejorar la legibilidad para los usuarios ASN.1: por ejemplo, las palabras clave aparecen con mayúsculas (no en negritas).

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.

[1] Recomendación UIT-T Z.100 (1999), *Lenguaje de especificación y descripción*.

[2] Recomendación UIT-T Z.107 (1999), *Lenguaje de especificación y descripción con incorporación de ASN.1*.

[3] Recomendación UIT-T X.680 (1997) | ISO/CEI 8824-1:1988, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de la notación básica,* incluidas las enmiendas 1 y 2 (1999) y el corrigendum 1 (1999).

[4] Recomendación UIT-T X.681 (1997) | ISO/CEI 8824-2:1998, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de objetos de información,* incluidos la enmienda 1 (1999) y el corrigendum 1 (1999).

[5] Recomendación UIT-T X.682 (1997) | ISO/CEI 8824-3:1988, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de constricciones.*

[6] Recomendación UIT-T X.683 (1997) | ISO/CEI 8824-4:1998, *Tecnología de la información – Notación de sintaxis abstracta uno: Parametrización de especificaciones de notación de sintaxis abstracta uno,* incluida la enmienda 1 (1999).

[7] Recomendación UIT-T X.690 (1997) | ISO/CEI 8825-1:1998, *Tecnología de la información – Reglas de codificación de notación de sintaxis abstracta uno: Especificación de las reglas de codificación básica, de las reglas de codificación canónica y de las reglas de codificación distinguida*.

3 Lote (Package)

La producción <package> se amplía como sigue:

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

donde ModuleDefinition (definición de modelo) es un no-terminal definido en la Recomendación UIT-T X.680:1997.

Modelo

Una <module definition> tiene el mismo significado que una <package definition> donde:

- ModuleIdentifier (sin ningún DefinitiveIdentifier) corresponde con <package name>;
- Imports corresponde con <package reference clause>s;
- Exports corresponde con <interface>.

Un lote ASN.1 es transformado en el SDL equivalente, antes de que sea considerado como un lote, y antes de cualesquiera transformaciones Z.100. En esta transformación, los nombres son transformados en identificadores plenamente calificados cuando el SDL requiere o permite un

identificador en vez de un nombre. Sin embargo, en aras de la concisión, esto se omite a menudo en los ejemplos de la presente Recomendación.

Ejemplo

La definición de módulo ASN.1

```
myway DEFINITIONS ::=
BEGIN
EXPORTS yes, no;
yes    BOOLEAN ::= TRUE
no     BOOLEAN ::= FALSE
END
```

es igual que

```
package myway;
public synonym yes, synonym no;
synonym     yes <<package Predefined>>Boolean = true;
synonym     no <<package Predefined>>Boolean = false;
endpackage myway;
```

De manera similar, cuando el lote es usado en <imports> de otro lote:

```
IMPORTS yes FROM myway;
```

Esto es igual que <package reference clause>:

```
use myway/yes;
```

NOTA – Como el SDL no soporta valores de identificador de objeto para identificación de lotes, los módulos ASN.1 con la misma modulereference pero diferentes DefinitiveIdentifiers causarán potencialmente problemas de resolución de nombre.

4 Definición y utilización de datos

Las diferentes definiciones de la utilización de datos se describen de la siguiente manera:

Gramática ASN.1	Define las reglas de producción de gramática que representan la construcción que se ha de representar en SDL.
Modelo	Describe las transformaciones de las diferentes partes de la gramática ASN.1 en producciones SDL. Esta parte hace referencia a la gramática SDL, representada como <SDL grammar rule>, y a la gramática ASN.1, representada como **ASN1GrammarRule**.

4.1 Correspondencia de nombres

Gramática ASN.1

Los nombres ASN.1 pueden contener caracteres con guiones ("-"). Si esto se usa en SDL, se interpretaría como el operador menos.

Modelo

Los nombres ASN.1 que contienen caracteres con guiones corresponden con nombres SDL léxicamente iguales, salvo que los guiones se convierten en subrayados.

Ejemplo

El nombre ASN.1 my-example-name corresponde con my_example_name en SDL.

4.2 Definiciones de variables y datos

4.2.1 Asignación de tipo (Type assignment)

Gramática ASN.1

TypeAssignment ::= typereference "::=" Type

Modelo

Si el **Type** es una **typereference**, entonces **TypeAssignment** es igual que una <syntype definition> que sólo contiene el equivalente SDL de **Type**.

Si **Type** es un **constrainedType**, entonces **TypeAssignment** es igual que una <syntype definition> que sólo contiene el equivalente SDL de **Constraint**.

Si **Type** no es una **typereference** ni un **constrainedType, TypeAssignment** es representada por una <partial type definition> donde <properties expression> está vacía y donde se omite <formal context parameters>.

Ejemplo

La asignación de tipo ASN.1

Mytype ::= AnotherType -- *typereference*

es igual que

syntype Mytype = AnotherType **endsyntype** Mytype; /* full qualification omitted here. */

La asignación de tipo ASN.1

S ::= INTEGER (0..5 | 10)

es igual que

syntype S = <<**package** Predefined>>Integer **constants** 0:5,10 **endsyntype** S;

La asignación de tipo ASN.1

Integerlist ::= SEQUENCE OF INTEGER

es igual que

value type Integerlist **inherits** <<**package** Predefined>>String
< <<**package** Predefined>> Integer> (" = <<**package** Predefined>>Emptystring) **endvalue type** Integerlist;

4.2.2 Asignación de valor (Value assignment)

Gramática ASN.1

ValueAssignment ::= valuereference Type "::=" Value

Modelo

Una **ValueAssignment** es representada por <synonym definition item>

Ejemplo

La definición ASN.1

```
yes BOOLEAN ::= TRUE
```

es igual que

```
synonym yes <<package Predefined>>Boolean = <<package Predefined>>true;
```

4.3 Expresiones de tipo (Type expressions)

4.3.1 Secuencia (Sequence)

Gramática ASN.1

```
SequenceType          ::=  SEQUENCE "{" ComponentTypeList "}"  |
                           SEQUENCE "{"  "}"

ComponentTypeList ::=      ComponentType  |
                                 ComponentTypeList "," ComponentType

ComponentType         ::=  NamedType  |
                           NamedType OPTIONAL  |
                           NamedType DEFAULT Value  |
                           COMPONENTS OF Type
NamedType                  ::=  identifier Type
```

Modelo

Un **SequenceType** es representado como una <structure definition> que contiene un <field> para cada **NamedType** de **SequenceType**. El <field> contiene un <field name>, que es igual que el **identifier** ASN.1 de **NamedType**, y un <field sort>, que es el **Type** transformado en un <sort identifier> SDL.

Si el **ComponentType** que contiene el **NamedType** es **OPTIONAL**, el campo SDL tiene la palabra clave **optional**.

Si el **ComponentType** que contiene el **NamedType** tiene un **DEFAULT Value**, el campo SDL tiene la palabra clave **default** y el valor es transformado en <ground expression> después de **default**.

Un **ComponentType** que es **COMPONENTS OF Type** es representado como una lista de <field>s ordenados, uno para cada campo asociado con **Type**. Estos campos están insertados en la posición de **COMPONENTS OF Type** en el orden que existen los campos en el **Type**.

Ejemplo

El tipo ASN.1:

```
S     ::=   SEQUENCE {
a     INTEGER,
b     IA5String OPTIONAL,
c     PrintableString DEFAULT "d"}
```

es igual que

```
value type S struct
      a <<package Predefined>> Integer;
      b <<package Predefined>> IA5String optional;
      c <<package Predefined>> PrintableString default 'd';
endvalue type S;
```

NOTA 1 – No hay distinción entre el uso de las palabras clave SEQUENCE y SET. Ésta es una mitigación en comparación con la Recomendación X.680.

NOTA 2 – En la presente Recomendación, no se necesitan rótulos para distinguir entre componentes del mismo tipo: se supone rotulación automática ASN.1.

4.3.2 Secuencia de (Sequenceof)

Gramática ASN.1

SequenceOfType ::= SEQUENCE OF Type
SetOfType ::= SET OF Type

Modelo

La especificación de una **SequenceOfType/SetOfType** es igual que la especificación del género String o Bag abstracto predefinido que tiene la transformación SDL de Type como el primer <actual context parameter> y el nombre Emptystring definido como el nombre literal para la cadena vacía. El género abstracto es <<**package** Predefined>> String para **SEQUENCE OF**, y <<**package** Predefined>> Bag para **SET OF**.

Si se especifica una constricción de tamaño ASN.1 para **Type**, **SequenceOfType** (o **SetOfType**) es un syntype (sintipo) que tiene la constricción de tamaño transformado como una <range condition> (véase 4.4). El género progenitor del sintipo es la **SequenceOfType** (o **SetOfType** respectivamente) sin la constricción de tamaño ASN.1. Este género progenitor tiene un nombre implícito y único y está definido en la unidad de alcance más cercana que abarca la ocurrencia de **SequenceOfType** (o **SetOfType** respectivamente).

Ejemplo

La definición ASN.1:

phonenumber ::= SEQUENCE SIZE (8) OF INTEGER (0..9)

es igual que las tres definiciones SDL:

value type S1 **inherits** <<**package** Predefined>> String <S2> (" = Emptystring) **endvalue type** S1;
syntype S2 = <<**package** Predefined>> Integer constants 0:9 **endsyntype**;
syntype phonenumber = S1 **constants size** (8) **endsyntype** phonenumber;

4.3.3 Elección (Choice)

Gramática ASN.1

ChoiceType ::= CHOICE "{" AlternativeTypeList "}"

AlternativeTypeList ::=
NamedType |
AlternativeTypeList "," NamedType

Modelo

Un **ChoiceType** es representado como una <choice definition> que contiene un <field> para cada **NamedType** del **ChoiceType**.

Ejemplo

El tipo de elección ASN.1

C ::= CHOICE {
a INTEGER,
b REAL }

es igual que

value type C **choice**
a <<**package** Predefined>> Integer;
b <<**package** Predefined>> Real;
endvalue type C;

4.3.4 Enumerado (Enumerated)

Gramática ASN.1

EnumeratedType ::= ENUMERATED "{" Enumeration "}"

Enumeration ::= EnumerationItem | EnumerationItem "," Enumeration

EnumerationItem ::= identifier | NamedNumber

NamedNumber ::= identifier "(" SignedNumber ")"|
identifier "(" DefinedValue ")"

Modelo

Para cada **EnumerationItem**, el **identifier** es transformado en una <literal signature> que tiene el mismo nombre que **EnumerationItem**. Si **EnumerationItem** contiene un **SignedNumber** (o **DefinedValue**), el <literal name> de <literal signature> va seguido por la transformación en SDL del **SignedNumber** (o **DefinedValue** respectivamente).

La definición ASN.1:

colours ::= ENUMERATED {blue(3),red, yellow(0)};

es igual que

value type colours
literals blue = 3, red, yellow = 0-
end value type colours;

4.3.5 Denominación de entero y bit (Integer and Bit Naming)

Gramática ASN.1

IntegerType ::= INTEGER |
INTEGER "{" NamedNumberList "}"

NamedNumberList ::= NamedNumber |
NamedNumberList "," NamedNumber

NamedNumber ::= identifier "(" SignedNumber ")" |
identifier "(" DefinedValue ")"

BitStringType ::= BIT STRING | BIT STRING "{" NamedBitList "}"

NamedBitList ::= NamedBit | NamedBitList "," NamedBit

NamedBit ::= identifier "(" number ")" |
identifier "(" DefinedValue ")"

Modelo

La especificación de un **IntegerType** con una **NamedNumberList** (o **BitStringType** con una **NamedBitList**) es igual que la especificación de una <synonym definition> en la unidad de alcance contenedora más cercana con un <synonym definition item> para cada **NamedNumber** (o **NamedBitList** respectively). El **identifier** del **NamedNumber** (o **NamedBit** respectivamente), es transformado en el <synonym name>. El <sort> de la <synonym definition item> es <<**package** Predefined>>Integer en el caso de un **NamedNumber**, y <<**package** Predefined>>Bit en el caso de un **NamedBit**. El **SignedNumber** o **DefinedValue** o **number** del **NamedNumber** o **NamedBitList** se utiliza como la <ground expression> del <synonym definition item>.

Ejemplo

La definición ASN.1:

```
Standards   ::=   SEQUENCE OF INTEGER{z100(0),x680(1),z10x(2)}
```

es igual que

```
value type standards inherits
      << package Predefined >> String < <<package Predefined>> Integer> ("= EmptyString)
endvalue type standards;
synonym   z100      Integer = 0;
synonym   x680      Integer = 1,
synonym   z10x      Integer = 2;
```

4.3.6 Subgama (Subrange)

Modelo

La especificación de una restricción de subgama ASN.1 es representada como que especifica el <sort> contenido y que añade la representación de la restricción de subgama ASN.1 después de la palabra clave **constants** en el <syntype> (si se especifica, en los demás casos se introduce la construcción).

Ejemplo

La definición ASN.1:

```
S     ::=   INTEGER(0..5 | 10)
```

es equivalente a

```
syntype S = <<package Predefined>> Integer constants 0:5, 10 endsyntype S;
```

A continuación se describe cómo se deriva <range condition>.

4.3.7 Cadena de bits (BitString)

Gramática ASN.1

```
BitStringType ::=
      BIT STRING
      BIT STRING "{" NamedBitList "}"
```

Modelo

El tipo ASN.1 BitStringType corresponde con <<**package** Predefined>> Bitstring de SDL.

4.3.8 Cadena de octetos

Gramática ASN.1

OctetStringType ::= OCTET STRING

Modelo

El tipo ASN.1 **OctetStringType** corresponde con <<**package** Predefined >>Octetstring de SDL.

4.4 Condición de intervalo (Range condition)

Modelo

Una condición de intervalo define un conjunto de valores. Se utiliza para definir un sintipo. Tiene un género progenitor asociado, que es el género especificado en la definición de sintipo. Un valor está dentro del conjunto de valores si el operador denotado por el identificador de operador da true (verdadero) cuando se aplica al valor.

El identificador de operador para una condición de intervalo dada se define como:

value type A
operators o: S -> Boolean;
/* where o is derived from the ASN.1 concrete syntax as explained below */
endvalue type A**;**

Cada **Range** en la condición de intervalo ASN.1 contribuye a las propiedades del operador que define el conjunto de valores:

o(V) == range1 **or** range2 **or** ... **or** rangeN

Si se especifica un sintipo sin una condición de intervalo, el resultado del operador es true.

En la siguiente explicación de cómo cada **Range** contribuye al resultado de operador, V indica el valor de argumento. Cada contribución debe estar bien formada, lo que significa que los operadores utilizados deben existir con una signatura apropiada para el contexto.

- Si las palabras clave **MIN** y **MAX** no están especificadas en **ClosedRange**, entonces **ClosedRange** contribuye con:

  E1 rel1 V and V rel2 E2

  donde E1 es **Value** de **LowerEndValue** y E2 es **Value** de **UpperEndValue**.

  Si "<"es especificado para **LowerEndValue** entonces rel1 es el operador "<", en los demás casos es el operador "<=".

  Si "<"es especificado para **UpperEndValue** entonces rel2 es el operador "<", en los demás casos es el operador "<=".

  Si se especifica la palabra clave **MIN** y no se especifica la palabra clave **MAX**, **Range** contribuye con:

  V rel2 E2

  Si se especifica la palabra clave **MAX** y no se especifica la palabra clave **MIN**, **Range** contribuye con:

  E1 rel1 V

  Si se especifican ambas palabras claves **MIN** y **MAX**, el operador siempre da true.

- Un **ContainedSubType** contribuye con:

  o1(V)

donde o1 es el operador implícito que define el conjunto de valores para el **Type** mencionado en **ContainedSubType**.

- Un **SizeConstraint** contribuye con:

  o1(length(V))

  donde o1 es el operador implícito que define el conjunto de valores para la <range condition> mencionadas en **SizeConstraint**.

- **InnerTypeConstraints** contribuye con:

  if length(V) = 0 **then** true **else** o1(first(V)) **and** o(Substring(V,2,length(V)-1)) **fi**; or

  if length(V) = 0 **then** true **else** o1(take(V)) **and** o(del(take(V), V)) **fi**

  lo que sea apropiado para el género de V. o es el operador implícito al que **InnerTypeConstraints** contribuye y o1 es el operador implícito para **Range** especificado en **InnerTypeConstraints**.

  InnerTypeConstraints tiene una contribución para cada **NamedConstraint** contenida que especifica constricciones del campo (véase 4.2.1) indicado por el **Identifier** del género progenitor.

  La palabra clave **PRESENT** se añade a **NamedConstraints** que no tienen palabra clave de fin (**PRESENT**, **ABSENT** u **OPTIONAL**) y **NamedConstraint**s de la forma **Identifier ABSENT** se añade para todos los campos (es decir, **Identifiers**) no mencionados explícitamente en **NamedConstraint**. Las **NamedConstraints** se añaden a las **InnerTypeConstraints** antes de derivar las contribuciones de cada **NamedConstraint**.

  Si se especifica **Range** para una **NamedConstraint**, la contribución es:

  E **and if** FPresent(V) **then** o1(V) **else** true **fi**

  donde E es la constricción presente para el campo, F (del nombre de operador FPresent) es el nombre del campo opcional y o1 es el operador implícito para **Range**. Si **Range** se omite, la contribución es sólo la constricción presente E.

  La constricción presente para un campo F es:

  FPresent(V)

  en el caso que **NamedConstraint** para el campo contenga la palabra clave **PRESENT**; y

  not FPresent(V)

  en caso que **NamedConstraint** para el campo contenga la palabra clave **ABSENT**. En todos los demás casos, la constricción presente es true.

4.5 Expresiones de valor (Value Expressions)

4.5.1 Primario de elección (Choice Primary)

Gramática ASN.1

ChoiceValue ::= identifier ":" Value

Modelo

Un **ChoiceValue** se representa como una <operator application> que tiene **Value** como argumento. El <operator identifier> en la <operator application> contiene un <qualifier> que representa el **Type** y un nombre de operador que es el **identifier**.

Ejemplo

El **ChoiceValue**:

myvalue : Mychoice

se representa como:

myvalue(Mychoice)

Cuando un **ChoiceValue** puede indicar una de varias aplicaciones de operador (es decir, un campo de más de un género de elección), se utiliza un calificador:

MyType ::= CHOICE …

myvalue : Mychoice

que se representa entonces como:

<< **type** Mytype >> myvalue(Mychoice)

4.5.2 Primario compuesto (Composite primary)

Un primario compuesto se construye con los valores para la representación en el SDL de respectivos tipos compuestos.

4.5.2.1 Valor de secuencia (Sequence value)

Gramática ASN.1

SequenceValue ::= "{" ComponentValueList "}" | "{" "}"
ComponentValueList ::= NamedValue |
ComponentValueList "," NamedValue

NOTA – No hay distinción entre SetValue y SequenceValue. Esta es una mitigación comparada con la Recomendación X.680.

Modelo

La especificación de **SequenceValue** en ASN.1 corresponde con la definición de **synonym** en SDL. En la correspondencia se proporciona **ComponentValueList** para estructurar el constructor de tipo de datos en SDL. El constructor de tipo de datos SDL requiere que todos los campos sean dados como entrada, de modo que los campos que son omitidos en **ComponentValueList** tienen que ser proporcionados vacíos en SDL. La aplicación de constructor de tipo de datos de estructura tendrá los mismos efectos en SDL que tendría en ASN.1.

Ejemplo

```
MYTYPE ::= SEQUENCE{
        a       INTEGER,
        b       INTEGER OPTIONAL,
        c       INTEGER DEFAULT 0,
        d       INTEGER,
        e       INTEGER OPTIONAL,
        f       INTEGER DEFAULT 0.
}
myValue MYTYPE ::= {a 1, b 1, c 1, d 1}
```

En este ejemplo, los campos a, b y c tienen un valor especificado y los campos c, d y e se omiten.

synonym myValue MYTYPE = (. 1, 1, 1, 1, , .);

Las consecuencias serían que los campos a, b, c y d se pondrían a 1, el campo e estaría ausente y el campo f obtendría el valor por defecto 0.

4.5.2.2 Valor de secuencia de (Sequenceof value)

Gramática ASN.1

SequenceOfValue ::= "{" ValueList "}" | "{" "}"

ValueList ::= Value | ValueList "," Value

NOTA – No hay distinción entre SetOfValue y SequenceOfValue. Ésta es una mitigación comparada con la Recomendación X.680.

Modelo

Una **SequenceOfValue** se representa como:

MkString(E1) // MkString(E2)// ... // MkString(En)

donde E1, E2, ..., En son los **Values** de **SequenceOfValue** en el orden de aparición. Si no se especifican **Values**, **SequenceOfValue** se representa como el nombre Emptystring.

El calificador **Type** del primario compuesto que contiene la SequenceOfValue precede cada operador MkString o el literal Emptystring, respectivamente.

4.5.2.3 Valor de identificador de objeto (Object identifier value)

Gramática ASN.1

ObjectIdentifierValue ::= "{" ObjIdComponentList "}" |
"{" DefinedValue ObjIdComponentList "}"

ObjIdComponentList ::= ObjIdComponent|
ObjIdComponent ObjIdComponentList

ObjIdComponent ::= NameForm |
NumberForm |
NameAndNumberForm

NameForm ::= identifier

NumberForm ::= number | DefinedValue

NameAndNumberForm ::= identifier "(" NumberForm ")"

Modelo

En ASN.1 el valor de identificador de objeto se utiliza para distinguir entre los módulos que tienen nombres iguales pero identificadores de objeto diferentes. Como los nombres de módulo y los identificadores de objeto no pueden corresponder de manera única con un identificador de lote que se utiliza en las cláusulas de uso de lotes, se prescinde del componente de identificador de objeto en la transformación en SDL. La identificación del módulo apropiado se deja abierta a soluciones manuales o específicas de la herramienta.

4.5.2.4 Valor real (Real value)

Gramática ASN.1

En ASN.1 el valor de un tipo real se define mediante la notación "RealValue":

```
RealValue ::=
        NumericRealValue | SpecialRealValue
NumericRealValue ::= 0    |
        SequenceValue     -- Value of the associated sequence type
SpecialRealValue ::=
        PLUS-INFINITY | MINUS-INFINITY
```

La forma **0** se utiliza para valores cero; la forma alterna para **NumericRealValue** no se utilizará para valores cero.

El tipo asociado para definición de valor y subtipificación es:

```
SEQUENCE {
    mantissa    INTEGER,
    base        INTEGER (2|10),
    exponent    INTEGER
        -- The associated mathematical real number is "mantissa"
        -- multiplied by "base" raised to the power "exponent"
    }
```

Modelo

Un **NumericalRealValue** de ASN.1 corresponde con un valor de género **real** de SDL, con el valor real calculado en la transformación. El **SpecialRealValue** será transformado respectivamente en el valor positivo o negativo mayor posible.

NOTA – La transformación de **SpecialRealValue** no está de acuerdo con la semántica ASN.1 prevista, ésta es una directriz al codificador/decodificador para utilizar un código especial que indica los valores infinitos. Como la codificación no está relacionada con datos en SDL transformados de datos ASN.1, esta mitigación debe ser aceptable.

Ejemplo

La definición ASN.1:

```
r50   REAL       ::= { mantissa 5, base 10, exponent 1}
```

es igual que:

```
synonym r50 Real = 50.0;
```

4.5.3 Primario de cadena (String primary)

Modelo

Un **StringValue** ASN.1 que contiene una **cstring** (nombre ASN.1 para cadena de caracteres delimitada por " al principio y al final) representa un <character string literal identifier> que consiste en el **Type** y un <character string literal> con el mismo <text> que la cadena ASN.1 **Text**. El **Type** para **cstring** es un 1A5Type definido por la presente Recomendación.

Un **StringValue** que contiene un **BitStringValue** o **HexStringValue** corresponde con operadores SDL <<**package** Predefined>> Bitstrings con la misma sintaxis.

4.5.4 Especificación de conjunto de elementos

Gramática ASN.1

```
ElementSetSpec ::= Unions |
                   ALL Exclusions
Unions ::= Intersections |
                   UElems UnionMark Intersections
UElems ::= Unions
Intersections ::= IntersectionElements |
                   IElems IntersectionMark IntersectionElements
IElems ::= Intersections
IntersectionElements ::= Elements | Elems Exclusions
Elems ::= Elements
Exclusions ::= EXCEPT Elements
UnionMark ::=  "|"   |   UNION
        IntersectionMark ::=  "^"   |   INTERSECTION
```

Modelo

Es posible combinar dos o más conjuntos de valores utilizando esta notación. El conjunto resultante es evaluado en la transformación y el resultado se hace corresponder con el SDL.

5 Correspondencia de tipos ASN.1 definidos en módulos ASN.1 que utilizan objetos, clases y conjuntos de información

5.1 Introducción

La Recomendación X.681 proporciona la notación ASN.1 que permite definir clases de objetos de información así como objetos y conjuntos de información individuales y darles nombres de referencia. Una clase de objeto de información es una plantilla para un conjunto de información que establece los atributos de los miembros de esa clase. Los objetos de información proporcionan un mecanismo de tabla genérica dentro del lenguaje ASN.1. Esta tabla genérica define la asociación de conjuntos específicos de valores de campo o tipos. Esta característica sustituye a la anterior construcción MACRO (disponible en ASN.1: 1990) y se utiliza principalmente para rellenar intervalos en una definición de tipo que depende de uno o más campos clave.

Esta cláusula supone que todas las construcciones ASN.1 definidas en las Recomendaciones X.681, X.682 y X.683 pueden ser utilizadas en los módulos ASN.1. Se identifica después la información contenida en clases de objeto de información, objetos de información y conjuntos de objetos de información ASN.1 que puede ser útil para la correspondencia con objetivos SDL apropiados. Se definen las correspondencias que son posibles y útiles. Cabe señalar que cierta información no será representada en SDL debido a las diferencias de naturaleza de los dos lenguajes.

5.2 Valor de campo de clase de objeto

Gramática ASN.1

```
ObjectClassFieldValue ::=
        OpenTypeFieldVal |
        FixedTypeFieldVal
OpenTypeFieldVal ::= Type ":" Value
FixedTypeFieldVal ::= BuiltinValue | ReferencedValue
```

Modelo

La especificación de clase de objeto ASN.1 nunca tiene correspondencia en el SDL. Sin embargo, la información contenida en la misma es esencial para hacer corresponder los elementos que son definidos por referencia a la clase.

En la especificación de un solo tipo ASN.1 cuyos campos son definidos por referencia a una clase, sólo se puede utilizar el valor de tipo fijo y los campos de conjunto de valores. La correspondencia con el SDL se efectúa de modo que el nombre de campo corresponda con el tipo que está siendo transformado y que el tipo de campo pueda ser hallado en el campo referenciado de la especificación de clase. Los valores de campo de tipo abierto no pueden tener correspondencia en el SDL. Sin embargo, las clases no fueron diseñadas principalmente para este uso (especificación de un solo tipo ASN.1) por lo que esto no debe plantear problemas en la práctica. La especificación de valores facultativos o por defecto tiene correspondencia en el SDL, como se indica en esta especificación.

Ejemplo

Si la ASN.1 contiene la siguiente especificación:

```
EXAMPLE-CLASS ::= CLASS {
&TypeField                                  OPTIONAL,    -- class field 1
      &fixedTypeValueField          INTEGER         OPTIONAL,    -- class field 2
      &variableTypeValueField       &TypeField      OPTIONAL,    -- class field 3
      &FixedTypeValueSetField       INTEGER         OPTIONAL,    -- class field 4
      &VariableTypeValueSetField    &TypeField      OPTIONAL,    -- class field 5
}
WITH SYNTAX {
      [TYPE-FIELD      &TypeField]
      [FIXED-TYPE-VALUE-FIELD                     &fixedTypeValueField]
      [VARIABLE-TYPE-VALUE-FIELD                  &variableTypeValueField]
      [FIXED-TYPE-VALUE-SET-FIELD                 &FixedTypeValueSetField]
      [VARIABLE-TYPE-VALUE-SET-FIELD              &VariableTypeValueSetField]
}
ExampleType ::= SEQUENCE {
      integerComponent1       EXAMPLE-CLASS.&fixedTypeValueField,       -- field 1
      integerComponent2       EXAMPLE-CLASS.&FixedTypeValueSetField     -- field 2
}
exampleValue ExampleType ::= {
      integerComponent1       123,                                      -- field 1
      integerComponent2       456                                       -- field 2
}
```

Los que pueden tener correspondencia en el SDL son ExampleType y exampleValue:

```
value type ExampleType
      integerComponent1       <<package Predefined>> Integer,       /* field 1 */
      integerComponent2       <<package Predefined>> Integer        /* field 2 */
endvalue type ExampleType;
synonym exampleValue ExampleType = (. 123, 456 .);
```

5.3 Objetos y conjuntos de objetos

Modelo

Las clases se utilizan predominantemente en la especificación de objetos basados en la clase y conjuntos de tales objetos. Todas estas construcciones se utilizan para definir un tipo que es una descripción genérica de una conjunto de tipos, a condición de que una especificación de constricción denomine al objeto por el cual el tipo es constreñido.

Para la transformación en el SDL, se ejecutan los siguientes pasos:

1) el nombre del conjunto de objetos constriñentes se utiliza para identificar los objetos que tendrán correspondencia en el SDL,
2) para cada objeto se crea un tipo de valor en SDL,
3) cada tipo de valor tiene tantos campos como campos haya en la especificación de objeto,
4) los nombres de campo se derivan de los nombres de los campos concordantes en la especificación del tipo constreñido,
5) la especificación de tipo para cada campo se deriva de la siguiente manera: si el campo es un campo de valor de tipo fijo o de conjunto de tipos, el tipo SDL se deriva del tipo de campo concordante en la clase de objeto referenciada con una especificación de subgama derivada del campo concordante de la especificación de objeto. Si el tipo de campo es abierto en la especificación de clase, el tipo SDL se deriva del tipo de campo dado en la especificación de objeto. Si en la especificación de objeto no se menciona el campo, éste no tiene correspondencia con el tipo SDL.

Son posibles varios niveles indirectos para hacer esto, porque los tipos de campo pueden ser especificados por referencia a alguna otra clase de objeto. También se puede utilizar la parametrización en la especificación ASN.1 y tiene que ser resuelta antes de efectuar la correspondencia con el SDL.

Ejemplo

Supóngase que la siguiente definición de clase de objeto de información viene dada en un módulo ASN.1:

```
ADDRESS-CLASS-TEMPLATE::= CLASS {
&whichType    INTEGER(0..3),
&OptType
}
WITH SYNTAX {
        WHICH &whichType
        OPT &OptType
}
```

Supóngase también que los siguientes objetos de información vienen dados en el módulo ASN.1:

```
gssi-object         ADDRESS-CLASS-TEMPLATE::=
{
        WHICH                   0
        OPT             Group-Short-Subscriber-Identity
}
gssi-ae-object      ADDRESS-CLASS-TEMPLATE::=
{
        WHICH                   1
        OPT             GSSI-AE
}
vgssi-object        ADDRESS-CLASS-TEMPLATE::=
{
        WHICH                   2
        OPT             Visitor-Group-Short-Subscriber-Identity
}
all-object          ADDRESS-CLASS-TEMPLATE::=
{
        WHICH                   3
        OPT             ALL-TYPE
}
```

Para que el ejemplo sea más completo, se especifican también los tipos referenciados en definiciones de objetos:

```
GSSI-AE        ::= SEQUENCE
{
        gssi            Group-Short-Subscriber-Identity,
        ae              Address-Extension
}
ALL-TYPE      ::= SEQUENCE
{
        gssi            Group-Short-Subscriber-Identity,
        ae              Address-Extension,
        vgssi           Visitor-Group-Short-Subscriber-Identity
}
```

Supóngase también que los objetos son especificados para formar un conjunto de objetos:

```
Address-Class-Instance-Set      ADDRESS-CLASS-TEMPLATE::=
{ gssi-object | gssi-ae-object | vgssi-object | all-object }
```

Las definiciones anteriores se utilizan para especificar una secuencia que sigue:

```
Group-Identity-Uplink ::= SEQUENCE
{
group-Identity-Address-Type    ADDRESS-CLASS-TEMPLATE.&whichType
({Address-Class-Instance-Set }),
opt             ADDRESS-CLASS-TEMPLATE.&OptType
({Address-Class-Instance-Set }{@.group-Identity-Address-Type })
}
```

Como el conjunto de objetos se menciona en la secuencia, esto significa realmente que se puede derivar un tipo de datos SDL del módulo ASN.1 para cada objeto del conjunto:

```
value type gssi_object STRUCT
        group_Identity_Address_Type         <<package Predefined>> Integer constants (0);
        opt                 Group_Short_Subscriber_Identity;
endvalue type gssi_object;
/* */

value type gssi_ae_object STRUCT
        group_Identity_Address_Type         <<package Predefined>> Integer constants (1);
        opt                 GSSI_AE;
endvalue type gssi_ae_object;
/* */

value type vgssi_object STRUCT
        group_Identity_Address_Type         <<package Predefined>> Integer constants (2);
        opt                 Visitor_Group_Short_Subscriber_Identity;
endvalue type vgssi_object;
/* */

value type all_object STRUCT
        group_Identity_Address_Type         <<package Predefined>> Integer constants (3);
        opt                 ALL_TYPE;
endvalue type all_object;
```

6 Correspondencia de tipos ASN.1 parametrizados

La Recomendación X.683 [6] define la manera de parametrizar los tipos ASN.1. Todos los conceptos ASN.1 de 1997 pueden ser parametrizados. Esto permite la especificación parcial de tipos o valores dentro de un módulo ASN.1 completando la especificación con la adición de los parámetros reales en el momento de la creación.

La Recomendación Z.100 define un concepto equivalente de parámetros de contextos formales.

El principio es que los tipos ASN.1 parametrizados correspondan con tipos Z.100 que tienen parámetros de contextos formales que permiten que existan especificaciones parciales sin parámetros reales y analizados formalmente.

Hay declaraciones de asignaciones parametrizadas que corresponden con cada una de las declaraciones de asignaciones especificadas en las Recomendaciones X.680 y X.681. La construcción "ParameterizedAssignment" es:

ParameterizedAssignment ::=
ParameterizedTypeAssignment |
ParameterizedValueAssignment|
ParameterizedValueSetTypeAssignment |
ParameterizedObjectClassAssignment|
ParameterizedObjectAssignment |
ParameterizedObjectSetAssignment

Se admite el uso de todas las asignaciones parametrizadas dentro de módulos ASN.1.

Los tipos y valores parametrizados pueden corresponder con el SDL, como se define en 6.1 y 6.2.

Las asignaciones parametrizadas que no pueden corresponder con tipos o valores SDL con parámetros de contexto pueden ser utilizadas en módulos ASN.1 para definir otros tipos o valores ASN.1 que pueden tener correspondencia en SDL, como se define en la cláusula 4.

6.1 Asignación de tipos parametrizados

Gramática ASN.1

ParameterizedTypeAssignment ::=
typereference
ParameterList
"::="
Type

Modelo

La diferencia entre tipos ASN.1 ordinarios y parametrizados es que **ParameterList** sigue a **typereference** y los parámetros formales contenidos en **ParameterList** se utilizan en la definición de **Type**.

Un **Type** definido en ASN.1 que utiliza parámetros de la **ParameterList** corresponde con el tipo SDL apropiado (definido en 4.2.1) a condición de que los parámetros ASN.1 sean parámetros de valor o de tipo. Estos parámetros corresponden con <formal context parameters> del tipo SDL. El parámetro de tipo ASN.1 corresponde con <sort context parameter> de SDL y el parámetro de valor ASN.1 corresponde con <synonym context parameter>de SDL. Los tipos parametrizados ASN.1 que tienen parámetros diferentes han de ser ejemplificados en módulos ASN.1, después de lo cual el tipo o valor resultante puede corresponder con el SDL.

Ejemplo

La definición de tipo ASN.1

```
TemplateMessage {INTEGER : minSize, INTEGER : maxSize, IndicatorType } ::= SEQUENCE
{
        asp             INTEGER,
        pdu             OCTET STRING(SIZE(minSize..maxSize)),
        indicator       IndicatorType
}
```

corresponde con el tipo SDL

```
value type TemplateMessage
<synonym minSize <<package Predefined>> Integer; synonym maxSize <<package Predefined>>
Integer; value type IndicatorType>
        asp             Integer;
        pdu             <<package Predefined>>Octetstring (SIZE(minSize:maxSize));
        indicator       IndicatorType;
endvalue type;
```

6.2 Asignación de valores parametrizados

Gramática ASN.1

```
ParameterizedValueAssignment ::=
        valuereference
        ParameterList
        Type
        "::="
        Value
```

Modelo

Una **ParameterizedValueAssignment** es representada por un <synonym definition item> con ítems de la **ParameterList** que corresponden con sus <formal context parameters>. Para los parámetros se aplican las condiciones indicadas en 6.1.

Ejemplo

La asignación de valor ASN.1:

```
genericBirthdayGreeting { IA5String : name } IA5String ::= { "Happy birthday, ", name, "!!" }
```

corresponde con

```
synonym genericBirthdayGreeting <synonym name <<package Predefined>> IA5String >

<<package Predefined>> IA5String = 'Happy birthday,'//name//'!!';
```

6.3 Definiciones parametrizadas ASN.1 de referencia

Modelo

En ASN.1 se utilizan tipos y valores parametrizados para definir tipos y valores ASN.1 simples proporcionando una **ActualPameterList.** Los tipos y valores resultantes puede corresponder con el SDL como se define en la cláusula 3. Si fue posible hacer corresponder la definición parametrizada con el SDL, las referencias ASN.1 a estas definiciones pueden corresponder con ejemplos SDL del tipo con parámetros de contexto, de modo que elementos de **ActualParameterList** correspondan con <actual context parameters>.

Ejemplo

Se puede emplear el tipo parametrizado utilizado en el ejemplo en 6.1 para definir un tipo ASN.1 simple, como sigue:

ActualMessage ::= TemplateMessage{10, 20, BOOLEAN}

Esto puede corresponder con el tipo SDL

value type **ActualMessage : TemplateMessage < 10, 20, <<**package **Predefined>> Boolean >**

El valor parametrizado genericBirthdayGreeting puede ser ejemplificado en ASN.1 de la siguiente manera:

greeting1 IA5String ::= genericBirthdayGreeting { "John" }, which can be mapped to SDL as

synonym greeting1 <<**package** Predefined>> IA5String = 'John'

6.4 Definición de objeto parametrizado junto con una clase de objeto de información

A continuación se da un ejemplo de la definición de objeto parametrizado junto con una clase de objeto de información y su correspondencia con el SDL.

```
MESSAGE-PARAMETERS ::= CLASS {
        &maximum-priority-level             INTEGER,
        &maximum-message-buffer-size        INTEGER
}
WITH SYNTAX {
        THE MAXIMUM PRIORITY LEVEL IS           &maximum-priority-level
        THE MAXIMUM MESSAGE BUFFER SIZE IS  &maximum-message-buffer-size
}
Message-PDU { MESSAGE-PARAMETERS : param } ::= SEQUENCE {
        priority-level      INTEGER (0..param.&maximum-priority-level),
        message             BMPString (SIZE (0..param.&maximum-message-buffer-size))
}
my-message-parameters MESSAGE-PARAMETERS ::= {
        THE MAXIMUM PRIORITY LEVEL IS 10
        THE MAXIMUM MESSAGE BUFFER SIZE IS  2000
}
MY-Message-PDU ::= Message-PDU { my-message-parameters }
```

El resultado del tipo SDL sería el siguiente:

```
value type MY_Message_PDU STRUCT
        priority_level      <<package Predefined>> INTEGER (0..10);
        message     <<package Predefined>> BMPString (SIZE (0..2000));
end value type;
```

7 Adiciones al lote Predefined

Se añadirán las siguientes definiciones al lote Predefined con el fin de sustentar la combinación de módulos ASN.1 con SDL.

```
syntype NumericChar = Character constants
' ',   '0',  '1',  '2',  '3',  '4',  '5',  '6',
'7',   '8',  '9' endsyntype;
/* */
```

```
/*          NumericString sort          */
    /*      Definition          */
    value type NumericString
        inherits String < NumericChar > ( " = emptystring )
        adding
            operators ocs in nameclass
                "" ( ('0':'9') or """ or (' ') )+ "" -> this NumericString;
    /* character strings of any length of any characters from a space ' ' to a '9' */
    axioms
        for all c in NumericChar 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 NumericString;
    /* */
   syntype PrintableChar = Character constants
    ' ',   '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',  "",
    '(',   ')',  '+',  ',',  '-',  '.',  '/',  ':',
    '=',   '?'
    constants;
    /* */

/*          PrintableString sort          */
    /*      Definition          */
    value type PrintableString
        inherits String < PrintableChar > ( " = emptystring )
        adding
            operators ocs in nameclass
                "" ( (' ':'&') or """ or ('(': '?') )+ "" -> this PrintableString;
    /* character strings of any length of any characters from a space ' ' to a '?' */
    axioms
        for all c in PrintableChar 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 PrintableString;
/* */
    syntype TeletexChar = Character constants
/* characters specified in X.680 clause 34.1 table 3 */ endsyntype;
/* */

/*          TeletexString sort          */
    /*      Definition          */
```

```
    value type TeletexString
        inherits String < TeletexChar > ( " = emptystring )
        adding
            operators ocs in nameclass
                /* characters specified in X.680 clause 34.1 table 3  */ -> this TeletexString;
    axioms
        for all c in TeletexChar 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 TeletexString;
    syntype VideotexChar = Character constants
/* characters specified in X.680 clause 34.1 table 3 */ endsyntype;
/* */

/*          VideotexString sort          */
    /*      Definition              */
    value type VideotexString
        inherits String < VideotexChar > ( " = emptystring )
        adding
            operators ocs in nameclass
            /* characters specified in X.680 clause 34.1 table 3 */ -> this VideotexString;
    axioms
        for all c in VideotexChar 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 VideotexString;

syntype IA5Char = Character endsyntype;

syntype IA5String = Charstring endsyntype;

value type GeneralChar
  literals /* All G and all C sets + SPACE + DELETE X.680 clause 34.1 table 3*/
    operators
        gchr      ( Integer ) -> this GeneralChar;
endvalue type;

value type UniversalChar
  literals /* see X.680 clause 34.6 */
    operators
        uchr      ( Integer ) -> this UniversalChar;
endvalue type;
/* */

/*          UniversalCharString sort          */

    /*     Definition              */
```

```
    value type UniversalCharString
        inherits String < UniversalChar > ( " = emptystring )
        adding
            operators ocs in nameclass
                                /* see X.680 clause 34.6 */ -> this UniversalCharString;
    axioms
        for all c in UniversalChar 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 UniversalCharString;
    /* */

/*      UTF8String sort                 */
    syntype UTF8String = UniversalCharString endsyntype;
/* */

/*      GeneralCharString sort          */
    /*      Definition              */
    value type GeneralCharString
        inherits String < GeneralChar > ( " = emptystring )
        adding
            operators ocs in nameclass
            /* All G and all C sets + SPACE + DELETE X.680 clause 34.1 table 3 */
                        -> this GeneralCharString;
    /* character strings of any length of any characters from a space ' ' to a '?' */
   axioms
        for all c in GeneralChar 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 GeneralCharString;
/* */
syntype GraphicChar = GeneralChar constants
/*All G+SPACE+DELETE as specified in X.680 clause 34.1 table 3 */
endsyntype;
/* */

/*          GraphicCharString sort              */
    /*      Definition              */
    value type GraphicCharString
        inherits String < GraphicChar > ( " = emptystring )
        adding
            operators ocs in nameclass
            /* All G + SPACE + DELETE as specified in X.680 clause 34.1 table 3*/
     -> this GraphicCharString;
    axioms
        for all c in GraphicChar nameclass (
        for all cs, cs1, cs2 in ocs nameclass (
        spelling(cs) == spelling(c)                                    ==> cs   == mkstring(c);
        spelling(cs) == spelling(cs1) // spelling(cs2),
```

```
        length(spelling(cs2)) == 1                                  ==> cs   == cs1 // cs2;
        ));
    endvalue type GraphicCharString;

syntype VisibleChar = Character constants
/* characters specified in X.680 clause 34.1 table 3 */
endsyntype;
/* */

/*      VisibleString sort              */
    /*      Definition              */
    value type VisibleString
        inherits String < VisibleChar > ( " = emptystring )
        adding
            operators ocs in nameclass
                /* characters specified in X.680 clause 34.1 table 3 */
         -> this VisibleString;
    axioms
        for all c in VisibleChar 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 VisibleString;

syntype BMPChar = UniversalChar CONSTANTS /* see X.680 clause 34.12 */
endsyntype;
/* */

/*      BMPCharString sort              */
    /*   Definition             */
    value type BMPCharString
        inherits String < BMPChar > ( " = emptystring )
        adding
            operators ocs in nameclass
                        /* see X.680 clause 34.12 */ -> this BMPCharString;
    axioms
        for all c in BMPChar 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 BMPCharString;
/* */

    value type NULL
    literals NULL

    endvalue type;
```

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
Serie Z	**Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación**

18099

Impreso en Suiza
Ginebra, 2000