UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Z.105

(10/2001)

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 notación de sintaxis abstracta uno

Recomendación UIT-T Z.105

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	**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 programación 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 especificaión y descripción combinado con módulos de notación de sintaxis abstracta uno

Resumen

La presente Recomendación define cómo se pueden utilizar módulos de notación de sintaxis abstracta uno (ASN.1) en combinación con el lenguaje de especificación y descripción (SDL). Este texto sustituye las correspondencias semánticas de ASN.1 a SDL definidas en la Rec. UIT-T Z.105 (1999). La utilización de notación ASN.1 insertada en SDL anteriormente definida en la Rec. UIT-T Z.105 (1995) no está definida en esta Recomendación.

El principal campo 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 sistemas de telecomunicaciones, así como los datos, mensajes y codificación de mensajes utilizados por estos sistemas.

Orígenes

La Recomendación UIT-T Z.105, revisada por la Comisión de Estudio 10 (2001-2004) del UIT-T, fue aprobada por el procedimiento de la Resolución 1 de la AMNT el 29 de octubre de 2001.

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 Asamblea Mundial de Normalización de las Telecomunicaciones (AMNT), 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 1 de la AMNT.

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

Introducción

- **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 SDL, mientras que los parámetros de los mensajes intercambiados se describan con ASN.1. Esta Recomendación define una correspondencia de construcciones de ASN.1 a construcciones de SDL ya existentes y contiene solamente una pequeña extensión de la Rec. UIT-T Z.100 para poder utilizar los módulos ASN.1.

- **Ámbito**

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

El uso de la notación ASN.1 incorporada en SDL se define en la Rec. UIT-T Z.107 [2].

- **Aplicación**

El principal campo 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, así como los datos, mensajes y codificación de mensajes utilizados por estos sistemas.

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 toda interpretación (que puede ser dinámica) de la especificación es 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 es conforme a la semántica del lenguaje.

- **Estatus/estabilidad**

La presente Recomendación sustituye las correspondencias semánticas de ASN.1 a SDL definidas en la Rec. UIT-T Z.105 (l999). El uso de la notación ASN.1 insertada en SDL anteriormente definido en la Rec. UIT-T Z.105 (l995) no está definido en la presente Recomendación.

Las modificaciones de las Recomendaciones UIT-T X.680 [3], X.681 [4], X.682 [5] y X.683 [6] o Z.100 [1] pueden requerir la introducción de modificaciones en la presente Recomendación.

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

- **Trabajo conexo**

– Rec. UIT.T Z.100 (1999), *Lenguaje de especificación y descripción*.

– Rec. UIT.T X.680 (1997), *ASN.1: Especificación de la notación básica*.

– Rec. UIT.T X.681 (1997), *ASN.1: Especificación de objetos de información*.

– Rec. UIT.T X.682 (1997), *ASN.1: Especificación de constricciones*.

– Rec. UIT.T X.683 (1997), *ASN.1: Parametrización de especificaciones ASN.1*.

– Rec. UIT.T Z.107 (1999), *Lenguaje de especificación y descripción con rotación en síntasis abstracta uno*.

Recomendación UIT-T Z.105

Lenguaje de especificación y descripción combinado con módulos de notación de sintaxis abstracta uno

1 Alcance

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 a la representación SDL interna utilizando construcciones SDL equivalentes y formando junto con el resto de la descripción SDL una especificación completa.

SDL es un lenguaje para la especificación y descripción de sistemas de telecomunicaciones. SDL tiene conceptos para:

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

ASN.1 es un lenguaje para la definición de datos. Guardan relación con 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 SDL, y los datos y mensajes con ASN.1. La codificación de estos mensajes puede ser descrita por referencia a las reglas de codificación pertinentes definidas para ASN.1.

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

1.2 Características de la combinación de SDL y ASN.1

Los sistemas descritos en 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 SDL

El uso de ASN.1, según se define en las Recomendaciones UIT-T X.680, X.681. X.682 y X.683, está soportado en combinación con el SDL, reconociendo que algunas construcciones ASN.1 no pueden corresponder satisfactoriamente a SDL (o al menos la correspondencia no ha sido identificada ni especificada en la presente Recomendación). Las construcciones que no pueden hacerse

corresponder a 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 Rec. UIT-T 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 los marcadores de extensión y de excepción ya no se utilizan en esta Recomendación. Algunas construcciones de la ASN.1 nunca son transformadas en SDL como tales, pero contienen información que puede ser usada en la transformación. Los ejemplos prominentes de estas construcciones son constricciones relacionales definidas en la Rec. UIT-T X.682, clases de objeto y conjuntos de objetos.

La utilización de SDL según se define en la Rec. UIT-T Z.100 [1] está soportada.

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 autosuficiente: la correspondencia definida en esta Recomendación se basa en la Rec. UIT-T Z.100 y en las Recomendaciones UIT-T X.680, X.681, X.682 y X.683. Se aplica el lenguaje definido en la Rec. UIT-T 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 Rec. UIT-T Z.100 para incorporar módulos ASN.1.

La cláusula 4 define la correspondencia de los tipos y valores ASN.1 de la Rec. UIT-T X.680 con datos de la Rec. UIT-T Z.100 para incorporar 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 construcciones X.682 se trata también en esta cláusula.

La cláusula 6 define la correspondencia de tipos ASN.1 parametrizados con datos de la Rec. UIT-T Z.100 para incorporar tipos de datos ASN.1 parametrizados.

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

1.5 Convenios utilizados en la presente Recomendación

Se aplican normalmente los convenios de la Rec. UIT-T 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.

Para las producciones de gramática ASN.1 se dan referencias a documentos originales. Sin embargo, las producciones de gramática ASN.1 se han copiado también en esta Recomendación en los lugares en que se ha considerado necesario para facilitar su lectura. En caso de conflicto, tendrán precedencia las producciones ASN.1 originales.

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 rotación de síntaxis abstracta uno incorporada*.

[3] Recomendación UIT-T X.680 (1997) | ISO/CEI 8824-1:1998, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de la notación básica,* más enmiendas 1 y 2 (1999) y corrigendum 1 (1999).

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

Grammaire ASN.1

`ModuleDefinition` se define en la cláusula 12.1 de [3].

```
ModuleDefinition ::=
              ModuleIdentifier
              DEFINITIONS
              TagDefault
              "::="
              BEGIN
              ModuleBody
              END
ModuleIdentifier ::=
              modulereference
              DefinitiveIdentifier
DefinitiveIdentifier ::=
              "{" DefinitiveObjIdComponentList "}" | empty
```

Modelo

La producción <package> se extiende como sigue:

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

<module definition> ::=

ModuleDefinition

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

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

- **`ModuleIdentifier`** (sin ningún **`DefinitiveIdentifier`**) corresponde al <package name>;
- **`Imports`** corresponde a las <package use clause>;
- **`Exports`** corresponde a la <interface>.

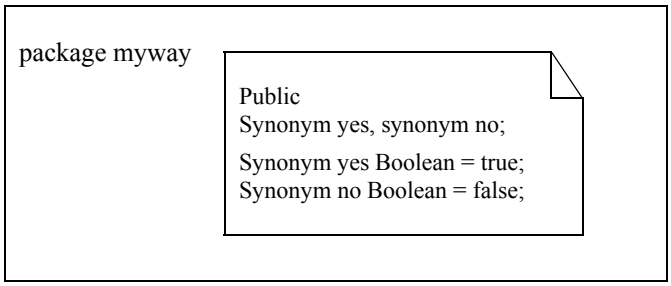
Un lote ASN.1 es transformado en el SDL equivalente, antes de que sea considerado como un lote, y antes de cualesquiera transformaciones Z.100. En esta transformación, los nombres son transformados en identificadores plenamente calificados cuando el SDL requiere o permite un identificador en vez de un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



De manera similar, cuando el lote es usado en el **`imports`** de otro lote:

```
IMPORTS yes FROM myway;
```

Esto es igual que la <package reference clause>:

use myway/yes;

NOTA – Como el SDL no soporta valores de identificador de objeto para identificación de lotes, módulos ASN.1 con la misma **`modulereference`** pero diferentes **`DefinitiveIdentifiers`** pueden causar 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

Grammaire ASN.1

Los nombres ASN.1 pueden contener caracteres guión ("-"). Si tal carácter se utiliza en SDL, se interpreta como el operador menos.

Modelo

Los nombres ASN.1 que contienen guiones corresponden a nombres SDL léxicamente iguales, salvo que los guiones se convierten en caracteres de subrayado.

Ejemplo

El nombre ASN.1 **`my-example-name`** corresponde a my_example_name en SDL.

4.2 Definiciones de variables y datos

4.2.1 Asignación de tipo (Type assignment)

Grammaire ASN.1

`TypeAssignment` se define en la cláusula 15.1 de [3].

```
TypeAssignment     ::=  typereference "::=" Type
```

Modelo

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

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

Si **`Type`** no es una **`typereference`** ni un **`constrainedType`**, la **`TypeAssignment`** se representa por una <partial type definition> donde <properties expression> está vacía y 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) }.

4.2.2 Asignación de valor (Value assignment)

Grammaire ASN.1

ValueAssignment se define en la cláusula 15.2 de [3].

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

Modelo

Una **ValueAssignment** se representa 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 Sequence (Secuencia)

Grammaire ASN.1

SequenceType se define en la cláusula 24.1 de [3]. **SetType** se define en la cláusula 26.1 de [3].

```
SequenceType ::=
                  SEQUENCE "{" "}" |
                  SEQUENCE "{" ExtensionAndException "}" |
                  SEQUENCE "{" ComponentTypeLists "}"

ExtensionAndException ::=   "..." | "..." ExceptionSpec

ComponentTypeLists ::=
              RootComponentTypeList |
              RootComponentTypeList   ","  ExtensionAndException |
              RootComponentTypeList   ","  ExtensionAndException  ","
              AdditionalComponentTypeList |
              ExtensionAndException   ","  AdditionalComponentTypeList
RootComponentTypeList       ::=  ComponentTypeList
AdditionalComponentTypeList ::=  ComponentTypeList
ComponentTypeList  ::=
              ComponentType   |
              ComponentTypeList "," ComponentType
ComponentType ::=
              NamedType    |
              NamedType OPTIONAL   |
              NamedType DEFAULT Value   |
              COMPONENTS OF Type
NamedType     ::=  identifier Type
```

Modelo

Un **SequenceType** se representa 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 <constant expression> después de **default**.

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

Las incidencias de **ExtensionAndException** en **SequenceType** se ignoran en la transformación.

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';
}

NOTA 1 – No se distingue entre el uso de las palabras clave **SEQUENCE** y **SET**. Ésta es una mitigación en comparación con Rec. UIT-T 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 Sequenceof (Secuencia de)

Grammaire ASN.1

SequenceOfType se define en la cláusula 25.1 de [3].

```
SequenceOfType::=  SEQUENCE OF Type
```

Modelo

La especificación de una **SequenceOfType** es igual que la especificación del género String 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.

Si se especifica una constricción de tamaño ASN.1 para **Type**, la **SequenceOfType** es un syntype que tiene la constricción de tamaño transformado como una <range condition> (véase 4.4). El género progenitor del syntype es la **SequenceOfType** 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 incidencia de la **SequenceOfType**.

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)
}
syntype S2 = <<**package** Predefined>> Integer **constants** (0..9) endsyntype;
syntype phonenumber = S1 **constants** size (8) endsyntype phonenumber;

4.3.3 Choice (Elección)

Grammaire ASN.1

ChoiceType se define en la cláusula 28.1 de [3].

```
ChoiceType                    ::=  CHOICE "{" AlternativeTypeLists "}"

AlternativeTypeLists          ::=
                    RootAlternativeTypeList |
                    RootAlternativeTypeList "," ExtensionAndException |
                    RootAlternativeTypeList "," ExtensionAndException  ","
                                AdditionalAlternativeTypeList
RootAlternativeTypeList       ::=  AlternativeTypeList
AdditionalAlternativeTypeList ::=  AlternativeTypeList
AlternativeTypeList           ::=
                    NamedType |
                    AlternativeTypeList "," NamedType
```

Modelo

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

Las incidencias de **ExtensionAndException** en **ChoiceType** se ignoran en la transformación.

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;
}

4.3.4 Enumerated (Enumerado)

Grammaire ASN.1

EnumeratedType se define en la cláusula 19.1 de [3].

```
EnumeratedType        ::=   ENUMERATED "{" Enumerations "}"
```

```
Enumerations          ::=
              RootEnumeration |
              RootEnumeration  ","   "..." |
              RootEnumeration  ","   "..."   ","  AdditionalEnumeration

RootEnumeration       ::=   Enumeration

AdditionalEnumeration ::=   Enumeration

Enumeration           ::=   EnumerationItem | EnumerationItem "," Enumeration

EnumerationItem       ::=   identifier | NamedNumber

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

Modelo

Un **EnumeratedType** se representa mediante una <partial type definition> donde <properties expression> está vacía y donde <formal context parameters> se omite. 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 transformada SDL del **SignedNumber** (o **DefinedValue** respectivamente).

Los ejemplares de ". . ." en **EnumeratedType** se ignoran en la transformación a SDL.

La definición ASN.1:

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

es igual que:

value type colours **{**
literals blue = 3, red, yellow = 0
}

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

Grammaire ASN.1

IntegerType se define en la cláusula 18.1 de [3]. **BitStringType** se define en la cláusula 21.1 de [3].

```
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 circundante más cercana con un <synonym definition item> para cada **NamedNumber** (o **NamedBitList** respectivamente). 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 <constant 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)
}
synonym z100 Integer = 0;
synonym x680 Integer = 1,
synonym z10x Integer = 2;

4.3.6 ValueRange (Gama de valores)

Grammaire ASN.1

`ValueRange` se define en la cláusula 48.4.1 de [3].

```
ValueRange ::= LowerEndpoint ".." UpperEndpoint
```

Modelo

La especificación de una restricción de gama de valores ASN.1 se representa como la especificación del <sort> contenido y la adición de la representación de la restricción de gama de valores ASN.1 después de la palabra clave **constants** en el <syntype>.

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 BitString (Cadena de bits)

Grammaire ASN.1

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

Modelo

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

4.3.8 OctetString (Cadena de octetos)

Grammaire ASN.1

```
OctetStringType ::= OCTET STRING
```

Modelo

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

4.3.9 Setof (Conjunto de)

Grammaire ASN.1

`SetOfType` se define en la cláusula 27.1 de [3].

```
SetOfType          ::=  SET OF Type
```

Modelo

La especificación de un **`SetOfType`** es igual que la especificación del género Bag <<**package** Predefined>> que tiene la transformada SDL de **`Type`** como el primer <actual context parameter> y el nombre Emptybag definido como el nombre literal para el bag vacío.

Si se especifica una constricción de tamaño ASN.1 para **`Type`**, el **`SetOfType`** es un syntype que tiene la constricción de tamaño transformado como una <range condition> (véase 4.4). El género progenitor del syntype es el **`SetOfType`** 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 incidencia de **`SetOfType`**.

4.4 Range condition (Condición de intervalo)

Modelo

Una condición de intervalo define un conjunto de valores. Se utiliza para definir un syntype. Tiene un género progenitor asociado, que es el género especificado en la definición de syntype. 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 syntype sin una condición de intervalo, el resultado del operador es true.

En la siguiente explicación de cómo cada Range contribuye al resultado del 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 un **`ClosedRange`** contribuye con:

  E1 rel1 V and V rel2 E2

donde E1 es el valor de **LowerEndValue** y E2 es el valor 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 clave **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**; o

  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 **NamedConstraints** 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 Value expressions (Expresiones de valor)

4.5.1 Choice Value (Valor de elección)

Grammaire ASN.1

ChoiceValue se define en la cláusula 28.8 de [3].

```
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 Composite primary (Primario compuesto)

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

4.5.2.1 Sequence value (Valor de secuencia)

Grammaire ASN.1

SequenceValue se define en la cláusula 24.16 de [3].

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

NOTA – No se distingue entre **SetValue** y **SequenceValue**. Ésta es una mitigación en comparación con la Rec. UIT-T Rec. X.680.

Modelo

La especificación de valor de secuencia en ASN.1 corresponde a 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 el 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, c y d de myValue tienen un valor asignado y los campos, e y f no tienen valor asignado.

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

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

4.5.2.2 Sequence of value (Secuencia de valor)

Grammaire ASN.1

`SequenceofValue` se define en la cláusula 25.3 de [3].

```
SequenceOfValue    ::=  "{" ValueList "}" | "{"  "}"
ValueList          ::=  Value | ValueList "," Value
```

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 al literal Emptystring, respectivamente.

4.5.2.3 Object identifier value (Valor de identificador de objeto)

Grammaire ASN.1

`ObjectIdentifierValue` se define en la cláusula 31.3 de [3].

Modelo

`ObjectIdentifierValue` se ignora en la transformación a SDL.

`ObjectIdentifierValue` se utiliza en ASN.1 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 hacerse corresponder de manera única a 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 Real value (Valor real)

Grammaire ASN.1

RealValue se define en la cláusula 20.6 de [3].

```
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 a un valor de género real de SDL, con el valor obtenido efectivamente en el cálculo, 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, porque 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 a partir 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 String primary (Primario de cadena)

Grammaire ASN.1

La cadena de caracteres en ASN.1 se define en la cláusula 11.11 de [3].

BitStringValue se define en la cláusula 21.9 de [3].

```
     BitStringValue ::=
              bstring                 |
              hstring                 |
              "{" IdentifierList "}"  |
              "{" "}"
     IdentifierList ::=
              identifier    |
              IdentifierList "," identifier
```

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 **IA5Type** definido por la presente Recomendación.

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

4.5.4 Especificación de conjunto de elementos

Grammaire ASN.1

ElementSetSpecs se define en la cláusula 46.1 de [3].

```
ElementSetSpecs             ::=
         RootElementSetSpec |
         RootElementSetSpec  ","   "..."  |
         "..."  ","  AdditionalElementSetSpec |
         RootElementSetSpec  ","  "..."  ","  AdditionalElementSetSpec

RootElementSetSpec          ::=  ElementSetSpec
AdditionalElementSetSpec    ::=  ElementSetSpec
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 a SDL.

Los ejemplares de "..." en **ElementSetSpecs** se ignoran en la transformación a SDL.

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

5.1 Introducción

La Rec. UIT-T X.681 proporciona la notación ASN.1 que permite definir clases de objetos de información así como objetos de información individuales y conjuntos de éstos, y darles nombres de referencia. Una clase de objeto de información es una plantilla para una colección de informaciones 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 llenar lagunas 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 UIT-T X.681, X.682 y X.683 pueden ser utilizadas en módulos ASN.1. Identifica por tanto la información contenida en clases de objeto de información, objetos de información y conjuntos de objetos de información ASN.1 que puede ser útil cuando se hace corresponder a 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 Definición y asignación de clases de objeto de información

Grammaire ASN.1

`ObjectClassAssignment` se define en la cláusula 9.1 de [4].

Modelo

Las definiciones de **`ObjectClass`** en ASN.1 no tienen correspondencia directa en SDL.

5.3 Object class field type (Tipo de campo clase de objeto)

Grammaire ASN.1

`ObjectClassFieldType` se define en la cláusula 14.1 de [4].

```
ObjectClassFieldType ::= DefinedObjectClass "." FieldName

FieldSpec ::=
     TypeFieldSpec |
     FixedTypeValueFieldSpec |
     VariableTypeValueFieldSpec |
     FixedTypeValueSetFieldSpec |
     VariableTypeValueSetFieldSpec |
     ObjectFieldSpec |
     ObjectSetFieldSpec
```

Modelo

Pueden definirse tipos ASN.1 utilizando la notación **`ObjectClassFieldType`** para extraer información de los campos de especificaciones de clase sin la presencia de constricciones de tabla. Tales tipos ASN.1 pueden hacerse corresponder a SDL, siempre que en su definición se utilicen sólo **`FixedTypeValueFieldSpec`** o **`FixedTypeValueSetFieldSpec`**. La correspondencia a un tipo de valor SDL se efectúa como se define en 4.3 una vez que se determina el significado de **`FixedTypeValueFieldSpec`** o **`FixedTypeValueSetFieldSpec`** a partir de las especificaciones de clase a que se hace referencia.

También se utiliza la notación **`ObjectClassFieldType`** en relación a las constricciones de tabla como se define en 5.5.2.

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
}
```

Cosas que pueden hacerse corresponder a SDL son **`ExampleType`** y **`exampleValue`**:

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

5.4 Definición y asignación de objetos de información

Grammaire ASN.1

`ObjetAssignment` se define en la cláusula 11.1 de [4].

Modelo

Las definiciones de **`Object`** en el módulo ASN.1 no tienen correspondencia equivalente en SDL.

5.5 Información procedente de objetos

Grammaire ASN.1

`InformationFromObjects` se define en la cláusula 15.1 de [4].

Modelo

Para hacer referencia a la información de la columna de la tabla asociada para un objeto o un conjunto de objetos se pueden utilizar los diversos casos de la notación **`InformationFromObjects`**.

En el módulo ASN.1, un tipo ASN.1 se puede especificar con campos definidos utilizando la notación **`InformationFromObjects`**. Tal tipo ASN.1 puede hacerse corresponder a SDL, a condición de que todas las incidencias de la notación **`InformationFromObjects`** puedan extenderse a un valor o a un tipo. El tipo ASN.1 como tal se hace corresponder como se especifica en 4.3, mientras que la semántica de la extensión **`InformationFromObjects`** sigue la semántica de ASN.1.

5.6 Especificación de constricción

Grammaire ASN.1

`GeneralConstraint` se define en la cláusula 8.1 de [5].

```
GeneralConstraint ::=
     UserDefinedConstraint |
     TableConstraint
```

Modelo

Los tipos especificados utilizando **`TableConstraint`** se hacen corresponder a SDL de acuerdo con las reglas dadas en 5.5.2. Los tipos especificados utilizando **`UserDefinedConstraint`** no pueden hacerse corresponder a SDL.

5.6.1 Constricciones definidas por el usuario

Grammaire ASN.1

`UserDefinedConstraint` se define en la cláusula 9.1 de [5].

```
UserDefinedConstraint ::=
     CONSTRAINED BY "{" UserDefinedConstraintParameter ","  *  "}"
```

Modelo

Esta forma de especificación de constricción puede considerarse como una forma especial de comentario ASN.1, ya que no puede ser tratada totalmente por la máquina. Por lo tanto, las especificaciones de tipo ASN.1 que utilizan **`UserDefinedConstraint`** no pueden hacerse corresponder a SDL.

5.6.2 Constricciones de tabla

Grammaire ASN.1

`TableConstraint` se define en la cláusula 10.3 de [5].

```
TableConstraint ::=
     SimpleTableConstraint |
     ComponentRelationConstraint
SimpleTableConstraint ::= ObjectSet
ComponentRelationConstraint ::=
     "{" DefinedObjectSet "}"   "{" AtNotation ","  +  "}"
AtNotation ::=
     "@" ComponentIdList |
     "@." ComponentIdList
ComponentIdList             ::= identifier  "."  +
```

Modelo

La notación de constricción puede aparecer (entre paréntesis) después de cualquier utilización de la construcción sintáctica "Type". Los diseñadores de aplicaciones pueden utilizar esta notación para definir un tipo de datos estructurado con ulteriores constricciones sobre los valores de sus campos. Ejemplos de tales constricciones son la limitación de la gama de uno o más componentes, o la especificación de una relación entre componentes. La primera es una **`SimpleTableConstraint`** y la última es una **`ComponentRelationConstraint`**.

Para los tipos con **`SimpleTableConstraint`** se aplican las siguientes reglas de transformación.

Antes de que el tipo constreñido pueda hacerse corresponder a SDL es necesario construir algunos tipos de valor SDL a partir de la especificación de clase y de la especificación del conjunto constriñente de la siguiente manera:

a) Por cada conjunto de objetos se crea cierto número de tipos de valor SDL. Los tipos se generan de manera que para cada campo de la CLASS asociada con el conjunto de objetos se genere un tipo de valor SDL. El nombre del tipo es la concatenación del nombre del conjunto de objetos, un carácter de subrayado ('_') y el nombre del campo clase concordante.

b) Si el campo clase es un **`FixedTypeValueFieldSpec`** se construye un syntype SDL. El syntype tiene una constricción de intervalo que es una unión de valores especificados por el campo concordante de cada objeto en el conjunto de objetos.

c) Si el campo clase es una **`VariableTypeValueFieldSpec`** se construye un tipo elección SDL. El tipo elección se construye de tal manera que todos los tipos encontrados en el campo concordante de todos los objetos que pertenecen al conjunto de objetos constriñente estén incluidos en la elección. Los nombres del campo elección se derivan como equivalentes en minúsculas de los tipos concordantes.

El tipo ASN.1 constreñido puede ahora hacerse corresponder a SDL. El tipo como tal se hace corresponder como se define en 4.3. Los nombres del campo SDL son los mismos que los nombres de campo ASN.1. La especificación ASN.1 de opcionalidad se conserva en la transformación. Por cada campo ASN.1 constreñido por un conjunto de objetos, el tipo SDL se especifica como un tipo construido a partir de la especificación de clase y la especificación del conjunto constriñente (items a) a c)).

Para las especificaciones de tipo ASN.1 que utilizan **`ComponentRelationConstraint`** se aplican las mismas reglas de transformación de tipo. Además de esto, por cada tipo ASN.1 con **`ComponentRelationConstraint`** se genera también un método de comprobación que atraviesa el objeto y comprueba las constricciones. El método de comprobación da verdadero "true" si se respetan todas las constricciones relacionales y falso "false" si se viola cualquiera de las constricciones relacionales.

El método de comprobación comprende los siguientes pasos:

Por cada elemento que interviene en la constricción relacional (es decir, que tiene vinculado un **`ComponentRelationConstraint`** o se menciona en cualquier **`ComponentRelationConstraint`**) se genera una declaración variable de prueba local. La generación sigue el siguiente esquema:

'dcl <test var name> <field type>; <test var name> := <field ref>;'

donde <test var name> es un nombre único para cada variable de prueba, <field type> es el tipo del elemento, <field ref> es una referencia al elemento. Si el elemento está presente, la variable se inicializa al valor del campo correspondiente del objeto.

Para cada constricción relacional se genera una prueba para cada combinación de valores o tipos constriñentes en la definición del conjunto de objetos. Cada prueba se genera utilizando el siguiente esquema:

'if (<test expr> and not (<value test>) then { return False; }

donde <test expr> es el resultado de la combinación de una prueba para cada valor o tipo constriñente utilizando el operador 'and'. Para los valores constriñentes, la prueba se define como:

'<test var name> = <test value>'

donde <test var name> es el nombre de la variable de prueba antes descrita y <test value> es el valor correspondiente tomado de la definición del conjunto de objetos.

Para los tipos constriñentes, la prueba se define como:

'<test var name>.<ispresent method>'

donde <test var name> es el nombre de la variable de prueba antes descrita y el <ispresent method> es el método que comprueba que el tipo correspondiente está presente.

El <value test> es el resultado de la combinación de una prueba para cada valor o tipo del elemento constreñido, en la definición del conjunto de objetos, que corresponde a los valores y tipos en el <test expr> anterior, utilizando el operador 'or'. Para los valores, cada prueba se da como:

'<test var name> = <value>'

donde el <test var name> es el nombre de la variable que corresponde al campo constreñido y <value> es un valor tomado de la definición del conjunto de objetos.

Para los tipos, la prueba se define como:

'<test var name>.<ispresent method>'

donde <test var name> es el nombre de la variable que corresponde al campo constreñido y el <ispresent method> es el método que comprueba que el tipo correspondiente está presente.

Por cada campo string en el tipo se genera un bucle de acuerdo con el siguiente esquema:

'loop(dcl <loop var> Integer := 1; <loop var> <= length(<string field>);
<loop var> := <loop var> + 1) { <loop body> }'

donde <loop var> es un nombre de variable único, <string field> es una referencia al campo string tratado y <loop body> es el resultado de aplicar los pasos de transformación indicados en esta cláusula a los elementos de la cadena.

Ejemplo 1

Un ejemplo de un tipo con **`SimpleTableConstraint`**:

```
ErrorReturn ::= SEQUENCE
{
  errorCategory ERROR-CLASS.&category({ErrorSet}) OPTIONAL,
  errors SEQUENCE OF SEQUENCE
  {
    errorCode ERROR-CLASS.&code
                        ({ErrorSet}),
    errorInfo ERROR-CLASS.&Type
                        ({ErrorSet})
  } OPTIONAL
}
```

Siempre y cuando las especificaciones de clase y el conjunto de objetos sean:

```
ERROR-CLASS ::= CLASS
{
  &category PrintableString (SIZE(1)),
  &code     INTEGER,
  &Type
}
WITH SYNTAX {&category &code &Type }

ErrorSet ERROR-CLASS ::=
{
  { "A" 1 INTEGER } |
  { "A" 2 REAL } |
  { "B" 1 CHARACTER STRING } |
  { "B" 2 GeneralString }
}
```

Los tipos SDL derivados de la especificación de constricción serían:

```
syntype ErrorSet_category = PrintableString (SIZE(1))
  constants 'A', 'B'
endsyntype;

syntype ErrorSet_code = <<package Predefined>> Integer
  constants 1, 2
endsyntype;

value type ErrorSet_Type { choice
  integer         <<package Predefined>> Integer;
  real            <<package Predefined>> Real;
  characterString <<package Predefined>> CharacterString;
  generalString   <<package Predefined>> GeneralString;
}
```

El tipo SDL construido sería el siguiente:

```
value type ErrorReturn { struct
  errorCategory ErrorSet_category optional;
  errors String <
    { struct
      errorCode ErrorSet_code,
      errorInfo ErrorSet_Type } > optional;
}
```

No se generaría ningún método de comprobación.

Ejemplo 2

Un ejemplo de un tipo con **`ComponentRelationConstraint`**.

```
ErrorReturn ::= SEQUENCE
{
  errorCategory ERROR-CLASS.&category({ErrorSet}) OPTIONAL,
  errors SEQUENCE OF SEQUENCE
  {
    errorCode ERROR-CLASS.&code
                        ({ErrorSet}{@errorCategory}),
    errorInfo ERROR-CLASS.&Type
                        ({ErrorSet}{@errorCategory,@.errorCode})
  } OPTIONAL
}
```

El tipo SDL correspondiente sería el siguiente:

```
value type ErrorReturn {
struct
  errorCategory ErrorSet_category optional;
  errors String <
    { struct
      errorCode ErrorSet_code,
      errorInfo ErrorSet_Type } > optional;
  method Check() -> Boolean
  {
     dcl t1 ErrorSet_category;
     dcl p1 Boolean;
     p1 := this.errorCategoryPresent();
     if (p1 = True)
{
          t1 := this.errorCategory;
     }
     if ((p1 = False) and (this.errorsPresent() =  True))
     {
          return False;
     }
     loop (dcl i1 Integer := 1; I <=length(errors); i1 := i1+1)
     {
          dcl t2 ErrorSet_code, t3 ErrorSet_Type;
          t2 := this.errors[i1].errorCode;
          t3 := this.errors[i1].errorInfo ;
          if (t1="A" and not( t2=1 or t2=2))
          {
               return False;
          }
          if (t1="B" and not( t2=1 or t2=2))
          {
               return False;
          }
          if (t1="A" and t2=1 and not (t3.integerPresent()))
          {
```

```
            return False;
        }
        if (t1="A" and t2=2 and not (t3.realPresent()))
        {
            return False;
        }
        if (t1="B" and t2=1 and not (t3.characterStringPresent()))
{
            return False;
        }
        if (t1="B" and t2=2 and not (t3.generalStringPresent))
{
            return False;
        }
         }
    }
}
```

6 Correspondencia de especificaciones ASN.1 parametrizadas

La Rec. UIT-T X.683 [6] define la manera de parametrizar una especificación ASN.1. Todos los conceptos ASN.1:1997 pueden parametrizarse. Esta característica permite la especificación parcial de tipos o valores dentro de un módulo ASN.1 completándose la especificación mediante la adición de los parámetros reales en el momento de la ejemplificación.

La Rec. UIT-T Z.100 define un concepto equivalente de parámetros de contexto formal.

6.1 Asignación parametrizada

Grammaire ASN.1

Hay enunciados de asignación parametrizada que corresponden a cada uno de los enunciados de asignación especificados en las Recomendaciones UIT-T X.680 y X.681. La construcción "ParameterizedAssignment" es:

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

Modelo

La utilización de todas las formas de **ParameterizedAssignment** está soportada en módulos ASN.1.

ParameterizedTypeAssignment puede hacerse corresponder a SDL como se define en 6.2 con base en los mecanismos de los parámetros de contexto formal SDL.

ParameterizedValueSetTypeAssignment, ParameterizedObjectClassAssignment, ParameterizedObjectAssignment, ParameterizedObjectSetAssignment se pueden utilizar en módulos ASN.1 para destinarlas a otras especificaciones ASN.1 pero no se hacen corresponder a sí mismas en SDL.

6.2 Asignación de tipo parametrizado

Grammaire ASN.1

```
ParameterizedTypeAssignment ::=
     typereference
     ParameterList
     "::="
     Type
ParameterList ::= "{" Parameter "," + "}"
Parameter ::= ParamGovernor ":" DummyReference | DummyReference
ParamGovernor ::= Governor | DummyGovernor
Governor ::= Type | DefinedObjectClass
DummyGovernor ::= DummyReference
DummyReference ::= Reference
```

Modelo

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

Un **`Type`** definido en ASN.1 utilizando parámetros de la **`ParameterList`** corresponde al tipo SDL apropiado (definido en 4.2.1) siempre que los parámetros ASN.1 sean parámetros de valor o de tipo. Tales parámetros se hacen corresponder a <formal context parameters> del tipo SDL. Un parámetro de tipo ASN.1 se hace corresponder a <sort context parameter> SDL y un parámetro de valor ASN.1 se hace corresponder a <synonym context parameter> a SDL.

Los tipos parametrizados ASN.1 que tienen parámetros que no son tipos o valores no pueden hacerse corresponder a SDL directamente. Sin embargo, si los parámetros pueden ser expandidos primero a tipos o valores, el tipo o valor ASN.1 resultante puede hacerse corresponder a SDL como se define en 6.3.

Ejemplo

La definición de tipo ASN.1:

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

Se hace corresponder al tipo SDL:

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

6.3 Referencias a definiciones de tipo parametrizadas ASN.1

Grammaire ASN.1

```
ParameterizedType ::=
         SimpleDefinedType
         ActualParameterList
ActualParameterList ::=
         "{" ActualParameter "," + "}"
ActualParameter ::=
     Type |
     Value    |
     ValueSet  |
     DefinedObjectClass |
     Object   |
     ObjectSet
```

Modelo

En ASN.1 se utilizan referencias a **ParameterizedType** y **ParameterizedValue** para definir nuevos tipos y valores ASN.1 proporcionando una **ActualParameterList**.

Si la definición de **ParameterizedType** era tal que fue posible hacerla corresponder a SDL y la **ActualParameterList** contiene sólo parámetros **Type** y **Value**, las referencias ASN.1 a tales definiciones pueden hacerse corresponder a ejemplificaciones SDL del tipo con parámetros de contexto de manera que elementos de **ActualParameterList** correspondan a <actual context parameters>. El ejemplo 1 ilustra tal correspondencia.

Si de acuerdo con 6.2 la definición de **ParameterizedType** no pudiera hacerse corresponder a la definición de tipo SDL con parámetros de contexto, las referencias a tales definiciones de **ParameterizedType** podrían hacerse corresponder a SDL de manera que el significado de tal tipo esté completamente expandido al nivel de tipos definidos en la cláusula 4 antes de que se efectúe una correspondencia a SDL.

Si la **ActualParameterList** contiene **ValueSet**, **DefinedObjectClass**, **Object** u **ObjectSet**, la correspondencia de tal tipo a SDL se efectúa de tal manera que el significado de tal tipo se expande completamente al nivel de los tipos definidos en la cláusula 4 antes de que se efectúe una correspondencia a SDL. El ejemplo 2 ilustra tal correspondencia.

Los tipos y valores ASN.1 derivados de definiciones parametrizadas ASN.1 referenciadas pueden hacerse corresponder a SDL como se define en la cláusula 4.

Ejemplo 1

El tipo parametrizado utilizado en el ejemplo de 6.2 puede utilizarse para definir una ASN.1 simple como sigue:

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

Esto puede hacerse corresponder a tipo SDL:

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

Ejemplo 2

Lo que sigue es un ejemplo de la definición de tipo ASN.1 derivada utilizando un parámetro que es un objeto de información. Los módulos ASN.1 necesitan contener la definición de clase de objeto de información pertinente con asignación parametrizada que tiene objeto de esa clase como referencia del parámetro ficticio, asignación de valor de objeto de información y referencia a definición de tipo parametrizada.

```
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 }
```

La semántica de ASN.1 es que con tal definición de clase, definición de tipo parametrizada y definición de valor de objeto, el tipo resultante de la transformación de MY-Message-PDU es equivalente a:

```
MY-Message-PDU ::=      SEQUENCE {
     priority-level     INTEGER  (0..10),
     message            BMPString (SIZE    (0..2000))
}
```

El tipo ASN.1 resultante puede hacerse corresponder a tipo SDL como:

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

6.4 Referencias a definiciones de valor parametrizado ASN.1

Grammaire ASN.1

```
ParameterizedValue ::=
     SimpleDefinedValue
     ActualParameterList

SimpleDefinedValue ::=
     Externalvaluereference |
     valuereference

ActualParameterList ::=
          "{" ActualParameter "," + "}"

ActualParameter ::=
     Type      |
     Value     |
     ValueSet  |
     DefinedObjectClass |
     Object    |
     ObjectSet
```

Modelo

Se utilizan referencias **`ParameterizedValue`** en ASN.1 para definir nuevos valores ASN.1 proporcionando una **`ActualParameterList`**.

Las referencias **ParameterizedValue** se hacen corresponder a SDL de tal manera que el significado de tal especificación de valor está completamente expandido al nivel de asignaciones de valor definidas en la cláusula 4 antes de que se establezca una correspondencia a SDL.

6.5 Referencias a otras definiciones parametrizadas ASN.1

```
ParameterizedValueSetType ::=
     SimpleDefinedType
     ActualParameterList

ParameterizedObjectClass ::=
     DefinedObjectClass
     ActualParameterList

ParameterizedObjectSet ::=
     DefinedObjectSet
     ActualParameterList

ParameterizedObject ::=
     DefinedObject
     ActualParameterList

ActualParameterList ::=
          "{" ActualParameter "," + "}"

ActualParameter ::=
     Type |
     Value     |
     ValueSet  |
     DefinedObjectClass |
     Object    |
     ObjectSet
```

Los módulos ASN.1 pueden contener la especificación de conjuntos de valores, clases de objetos, conjuntos de objetos y objetos definidos mediante referencia al **SimpleDefinedType** con **ActualParameterList**. Tales especificaciones no se hacen corresponder a SDL.

7 Adiciones al lote Predefined

Se añadirán las siguientes definiciones al lote Predefined con miras al soporte de 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 ITU-T Rec. X.680 clause 36.1 Table 3 */ endsyntype;
/* */

/*  TeletexString sort    */
/*  Definition            */
value type TeletexString
    inherits String < TeletexChar > ( '' = emptystring )
    adding
        operators ocs in nameclass
/*  characters specified in ITU-T Rec. X.680 clause 36.1 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 ITU-T Rec. X.680 clause 36.1 Table 3 */ endsyntype;
/* */

/*  VideotexString sort       */
/*  Definition                */
value type VideotexString
    inherits String < VideotexChar > ( '' = emptystring )
    adding
        operators ocs in nameclass
/*  characters specified in ITU-T Rec. X.680 clause 36.1 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 ITU-T Rec. X.680 clause 36.1
Table 3 */
operators
    gchr         ( Integer ) -> this GeneralChar;
endvalue type;

value type UniversalChar
    literals /*  see ITU-T Rec. X.680 clause 36.6    */
operators
    uchr         ( Integer ) -> this UniversalChar;
endvalue type;
/* */

/*  UniversalCharString sort     */
/*  Definition                   */

value type UniversalCharString
    inherits String < UniversalChar > ( '' = emptystring )
    adding
        operators ocs in nameclass
/*  see ITU-T Rec. X.680 clause 36.6    */ -> 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 ITU-T Rec. X.680 clause 36.1 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 ITU-T Rec. X.680 clause 36.1 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 ITU-T Rec. X.680 clause 36.1 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 ITU-T Rec. X.680 clause 36.1 Table 3 */
    endsyntype;
    /* */

    /*   VisibleString sort    */
    /*   Definition            */
    value type VisibleString
        inherits String < VisibleChar > ( " = emptystring )
        adding
            operators ocs in nameclass
    /*   characters specified in ITU-T Rec. X.680 clause 36.1 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 ITU-T Rec. X.680 clause 36.12 */
    endsyntype;
    /* */
    /*   BMPCharString sort      */
    /*    Definition             */
    value type BMPCharString
        inherits String < BMPChar > ( " = emptystring )
        adding
            operators ocs in nameclass
    /*   see ITU-T Rec. X.680 clause 36.12   */ -> 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	Redes de cable y transmisión de programas radiofónicos y televisivos, 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**

Ginebra, 2002