UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T X.920

(12/97)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE X: REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

Procesamiento distribuido abierto

Tecnología de la información – Procesamiento distribuido abierto – Lenguaje de definición de interfaz

Recomendación UIT-T X.920

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES DE LA SERIE X DEL UIT-T

REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

REDES PÚBLICAS DE DATOS	
Servicios y facilidades	X.1–X.19
Interfaces	X.20–X.49
Transmisión, señalización y conmutación	X.50–X.89
Aspectos de redes	X.90–X.149
Mantenimiento	X.150–X.179
Disposiciones administrativas	X.180–X.199
INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Modelo y notación	X.200–X.209
Definiciones de los servicios	X.210–X.219
Especificaciones de los protocolos en modo conexión	X.220–X.229
Especificaciones de los protocolos en modo sin conexión	X.230–X.239
Formularios para declaraciones de conformidad de implementación de protocolo	X.240–X.259
Identificación de protocolos	X.260–X.269
Protocolos de seguridad	X.270–X.279
Objetos gestionados de capa	X.280–X.289
Pruebas de conformidad	X.290–X.299
INTERFUNCIONAMIENTO ENTRE REDES	
Generalidades	X.300–X.349
Sistemas de transmisión de datos por satélite	X.350–X.399
SISTEMAS DE TRATAMIENTO DE MENSAJES	X.400–X.499
DIRECTORIO	X.500–X.599
GESTIÓN DE REDES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS Y ASPECTOS DE SISTEMAS	
Gestión de redes	X.600–X.629
Eficacia	X.630–X.639
Calidad de servicio	X.640–X.649
Denominación, direccionamiento y registro	X.650–X.679
Notación de sintaxis abstracta uno	X.680–X.699
GESTIÓN DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Marco y arquitectura de la gestión de sistemas	X.700–X.709
Servicio y protocolo de comunicación de gestión	X.710–X.719
Estructura de la información de gestión	X.720–X.729
Funciones de gestión y funciones de arquitectura de gestión distribuida abierta	X.730–X.799
SEGURIDAD	X.800–X.849
APLICACIONES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Compromiso, concurrencia y recuperación	X.850–X.859
Procesamiento de transacciones	X.860–X.879
Operaciones a distancia	X.880–X.899
PROCESAMIENTO DISTRIBUIDO ABIERTO	**X.900–X.999**

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

NORMA INTERNACIONAL 14750

RECOMENDACIÓN UIT-T X.920

TECNOLOGÍA DE LA INFORMACIÓN – PROCESAMIENTO DISTRIBUIDO ABIERTO – LENGUAJE DE DEFINICIÓN DE INTERFAZ

Resumen

En esta Recomendación | Norma Internacional se describe un lenguaje de definición de interfaz (IDL, *interface definition languaje*) para las especificaciones que están en conformidad con el lenguaje informático definido en la arquitectura del modelo de referencia del procesamiento distribuido abierto (ODP) (véase la Rec. UIT-T X.903 | ISO/CEI 10746-3). El IDL permite describir las interfaces de objeto, junto con sus operaciones y parámetros conexos. Este lenguaje está totalmente en consonancia con el IDL CORBA desarrollado por el grupo de gestión de objeto (OMG, *object management group*).

Orígenes

El texto de la Recomendación UIT-T X.920 se aprobó el 12 de diciembre de 1997. Su texto se publica también, en forma idéntica, como Norma Internacional ISO/CEI 14750.

La Recomendación UIT-T X.920 resulta de la adopción del texto de las especificaciones IDL del OMG para las cuales la propiedad de distribución a nivel mundial y los derechos sobre trabajos derivados son propiedad del grupo de gestión de objeto (OMG, *object management group*)

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

Introducción

El rápido crecimiento del procesamiento distribuido ha llevado a la necesidad de un marco de coordinación para la normalización de procesamiento distribuido abierto (ODP, *open distributed processing*). El modelo de referencia del procesamiento distribuido abierto (RM-OPD, *reference model of open distributed processing*) proporciona este tipo de marco. Define una arquitectura en la que puede integrarse soporte de distribución de interoperabilidad y de portabilidad.

Uno de los componentes de la arquitectura (descrito en la parte 3 de RM-OPD: Arquitectura) (véase la Rec. UIT-T X.903 | ISO/CEI 10746-3) es un lenguaje que resulta adecuado para describir la firma de interfaces informáticas de operación. Esta Recomendación | Norma Internacional incluye este tipo de lenguaje de definición de interfaz, denominado ODP IDL.

NOTA – Esta Recomendación | Norma Internacional concuerda técnicamente con la especificación de lenguaje de definición de interfaz CORBA.

El anexo A es normativo y proporciona un conjunto normalizado de excepciones para una determinada infraestructura de distribución ODP.

El anexo B es informativo y proporciona la codificación CORBA de un tipo denominado TypeCode que representa descripciones de tipo.

NORMA INTERNACIONAL

RECOMENDACIÓN UIT-T

TECNOLOGÍA DE LA INFORMACIÓN – PROCESAMIENTO DISTRIBUIDO ABIERTO – LENGUAJE DE DEFINICIÓN DE INTERFAZ

1 Alcance

Esta Recomendación | Norma Internacional está concebida con el fin de proporcionar el modelo de referencia ODP (véase la Rec. UIT-T X.902 | ISO/CEI 10746-2 y la Rec. UIT-T X.903 | ISO/CEI 10746-3) con un lenguaje y una notación neutral de entorno para describir las firmas de interfaz de operaciones informáticas. El empleo de esta notación no entraña la utilización de mecanismos y protocolos específicos de apoyo.

2 Referencias normativas

Las siguientes Recomendaciones y Normas Internacionales contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones de la presente Recomendación | Norma Internacional. Al efectuar esta publicación, estaban en vigor las ediciones indicadas. Todas las Recomendaciones y Normas son objeto de revisiones, por lo que se preconiza que los participantes en acuerdos basados en la presente Recomendación | Norma Internacional investiguen la posibilidad de aplicar las ediciones más recientes de las Recomendaciones y las Normas citadas a continuación. Los miembros de la CEI y de la ISO mantienen registros de las Normas Internacionales actualmente vigentes. La Oficina de Normalización de las Telecomunicaciones de la UIT mantiene una lista de las Recomendaciones UIT-T actualmente vigentes.

2.1 Recomendaciones | Normas Internacionales idénticas

- Recomendación UIT-T X.902 (1995) | ISO/CEI 10746-2:1996, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Fundamentos.*
- Recomendación UIT-T X.903 (1995) | ISO/CEI 10746-3:1996, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Arquitectura.*

2.2 Referencias adicionales

- ISO/CEI 646:1991, *Information technology – ISO 7-bit coded character set for information interchange.*
- ISO/CEI 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet N.º 1.*

3 Definiciones

A los efectos de esta Recomendación | Norma Internacional se aplican las definiciones siguientes:

En esta Recomendación | Norma Internacional se utilizan los siguientes términos definidos en la Rec. UIT-T X.902 | ISO/CEI 10746-2:

- objeto;
- interfaz;
- firma de interfaz.

En esta Recomendación | Norma Internacional se usa el siguiente término definido en la Rec. UIT-T X.903 | ISO/CEI 10746-3:

- operación.

4 Sintaxis y semántica ODP IDL

El ODP IDL (lenguaje de definición de interfaz) es el lenguaje utilizado para describir las firmas de interfaz para interfaces que invocan los objetos del cliente y proporcionan las implementaciones de objeto. Una definición de interfaz escrita en ODP IDL define totalmente la firma de interfaz y especifica completamente cada uno de los parámetros de la operación.

Una especificación ODP IDL está constituida de forma lógica por uno o más ficheros. Un fichero se traduce conceptualmente en varias fases. La primera fase de preprocesamiento realiza la inclusión del fichero y su sustitución macro. El preprocesamiento se controla mediante directrices introducidas por líneas que tienen # como primer carácter distinto de espacio en blanco. El resultado del preprocesamiento es una secuencia de testigos. Dicha secuencia de testigos, que es un fichero después de preprocesamiento, se denomina una unidad de traslación.

El ODP IDL obedece a las mismas reglas léxicas que C++[1)], aunque se introducen nuevas palabras clave para soportar los conceptos de distribución. El ODP IDL proporciona asimismo soporte total para las características de preprocesamiento C++. Se prevé que la especificación ODP IDL rastreará los cambios pertinentes introducidos en el C++ como resultado de los esfuerzos de normalización de ISO/CEI.

La subcláusula 4.1 contiene una descripción de los convenios léxicos del ODP IDL. La subcláusula 4.2 contiene una descripción del preprocesamiento ODP IDL. En la subcláusula 4.13 y siguientes se describen las reglas de ámbito de aplicación de los identificadores en una especificación ODP IDL.

La gramática ODP IDL es un subconjunto de ISO/CEI C++ con constructivos adicionales para soportar el mecanismo de invocación de operaciones. El ODP IDL es un lenguaje descriptivo; soporta la sintaxis C++ para las declaraciones de constante, tipo y operación; no incluye ninguna variable o estructura algorítmica. La subcláusula 4.3 versa sobre la gramática.

En esta cláusula se describe la semántica ODP IDL y se proporciona la sintaxis para los constructivos gramaticales ODP IDL. Para la descripción de la gramática ODP IDL se utiliza una notación sintáctica que es similar a la forma ampliada Backus-Naur (EBNF, *extended Backus-Naur format*); en el cuadro 1 se enumeran los símbolos utilizados en este formato y se aclara su significado.

Cuadro 1 – Formato EBNF del ODP IDL

Símbolo	Significado
::=	Se define como
|	De no ser así/por otra parte
<text>	No terminal
"text"	Literal
*	La unidad sintáctica precedente puede repetirse cero o más veces
+	La unidad sintáctica precedente puede repetirse cero o más veces
{}	Las unidades sintácticas encerradas entre llaves están agrupadas como una sola unidad sintáctica
[]	La unidad sintáctica encerrada entre llaves es facultativa – puede ocurrir cero o más veces

4.1 Convenios léxicos

En esta subcláusula[2)] se presentan los convenios léxicos del ODP IDL. Se definen testigos en una especificación ODP IDL y se describen comentarios, identificadores, palabras clave y literales – entero, carácter, constantes de coma flotante y literales en cadena.

---

1) Ellis, Margaret A. y Bjarne Stroustrop. *The Annotated C++ Reference Manual*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990, ISBN 0-201-51459-1.

2) El texto de esta subcláusula es una adaptación de *The Annotated C++ Reference Manual*, capítulo 2; difiere del mismo en la lista de palabras clave legales y en la puntuación.

El ODP IDL utiliza el conjunto de caracteres ISO/CEI Latin-1 (ISO/CEI 8859-1). Este conjunto de caracteres está dividido en caracteres alfabéticos (letras), dígitos, caracteres gráficos, el carácter de espacio ("blank") y los caracteres de formateo. En el cuadro 2 se muestran los caracteres alfabéticos ODP IDL; las equivalencias en mayúscula y minúscula se agrupan en pares. El cuadro 3 muestra los diálogos y el cuadro 4 los caracteres gráficos.

En el cuadro 5 se muestran los caracteres de formateo.

4.1.1 Testigos

Hay cinco tipos de testigos: identificadores, palabras clave, literales, operadores y otros separadores. Los espacios, tabuladores verticales y horizontales, nuevas líneas, retornos del carro y comentarios (colectivo, "espacio blanco"), según se describen a continuación, se ignoran, salvo que sirvan como testigos separados. Se requiere cierto espacio en blanco para separar identificadores, palabras clave y constantes que de otro modo serían adyacentes.

Si el tren de entrada ha sido descompuesto en testigos hasta un carácter dado, se considera que el testigo siguiente es la cadena más larga de caracteres que podría posiblemente constituir un testigo.

4.1.2 Comentarios

Los caracteres /* inician un comentario, que termina con los caracteres */. Estos comentarios no jerarquizan. Los caracteres // inician un comentario que termina al final de la línea en el que ocurren. Los caracteres de comentario //, /* y */ no tienen un significado especial dentro de un comentario // y se tratan exactamente igual que los otros caracteres. Análogamente, los caracteres de comentario // y /* no tienen un significado especial dentro de un comentario /*. Los comentarios pueden contener caracteres de alfabeto, dígito, gráfico, espacio, tabulación horizontal, tabulación vertical, página siguiente y nueva línea.

4.1.3 Identificadores

Un identificador es una secuencia arbitrariamente larga de caracteres alfabéticos, dígitos y subrayado ("_"). El primer carácter debe ser un carácter alfabético. Todos los caracteres son significantes.

Los identificadores que difieren únicamente en el tamaño de la letra (mayúscula/minúscula) colisionan y generan un error de compilación. Para una definición, un identificador debe deletrearse coherentemente (en relación con las mayúsculas/minúsculas) a lo largo de una especificación.

Cuando se comparan dos identificadores para observar si colisionan:

- Las letras en mayúscula y minúscula se tratan como la misma letra. En el cuadro 2 se definen las correspondencias de equivalencia entre letras mayúsculas y minúsculas.
- En la comparación *no* se tienen en cuenta las equivalencias entre dígrafos y pares de letras (por ejemplo "æ" y "ae" no se consideran equivalentes) o las equivalencias entre letras acentuadas y no acentuadas (por ejemplo, "à" y "a" no se consideran equivalentes).
- Todos los caracteres son significantes.

Para los identificadores ODP IDL hay un solo espacio de nombre. La utilización del mismo identificador para una constante y una interfaz, por ejemplo, genera un error de compilación.

4.1.4 Palabras clave

Los identificadores enumerados en el cuadro 6 están reservados para usarlos como palabras clave, y no se pueden utilizar de otro modo. La palabra clave `Object` en el ODP IDL se utiliza para representar un tipo de interfaz, mientras que la palabra clave `interface` se utiliza para indicar el comienzo de una declaración de interfaz en una plantilla de firma de interfaz. La palabra clave `Object` puede utilizarse como especificador de tipo. La palabra clave `attribute` define un método que proporciona acceso a una porción del estado de un objeto. Una definición de atributo es lógicamente equivalente a declarar un par de métodos de acceso; uno para recuperar el valor del atributo y otro para fijar el valor del mismo.

La palabra clave Exception se utiliza para representar el concepto ODP de terminación denominada sin éxito, siendo el nombre de excepción el nombre de terminación.

Cuadro 2 – Los 114 caracteres alfabéticos (letras)

Carácter	Descripción	Carácter	Descripción
Aa	A mayúscula/a minúscula	Àà	A mayúscula/a minúscula con acento grave
Bb	B mayúscula/b minúscula	Áá	A mayúscula/a minúscula con acento agudo
Cc	C mayúscula/c minúscula	Ââ	A mayúscula/a minúscula con acento circunflejo
Dd	D mayúscula/d minúscula	Ãã	A mayúscula/a minúscula con tilde
Ee	E mayúscula/e minúscula	Ää	A mayúscula/a minúscula con diéresis
Ff	F mayúscula/f minúscula	Åå	A mayúscula/a minúscula con cero volado
Gg	G mayúscula/g minúscula	Ææ	Diptongo de A con E mayúscula/minúscula
Hh	H mayúscula/h minúscula	Çç	C mayúscula/c minúscula con cedilla
Ii	I mayúscula/i minúscula	Èè	E mayúscula/e minúscula con acento grave
Jj	J mayúscula/j minúscula	Éé	E mayúscula/e minúscula con acento agudo
Kk	K mayúscula/k minúscula	Êê	E mayúscula/e minúscula con acento circunflejo
Ll	L mayúscula/l minúscula	Ëë	E mayúscula/e minúscula con diéresis
Mm	M mayúscula/m minúscula	Ìì	I mayúscula/i minúscula con acento grave
Nn	N mayúscula/n minúscula	Íí	I mayúscula/i minúscula con acento agudo
Oo	O mayúscula/o minúscula	Îî	I mayúscula/i minúscula con acento circunflejo
Pp	P mayúscula/p minúscula	Ïï	I mayúscula/i minúscula con diéresis
Qq	Q mayúscula/q minúscula	Ðð	eth islandesa mayúscula/minúscula
Rr	R mayúscula/r minúscula	Ññ	N mayúscula/n minúscula con tilde
Ss	S mayúscula/s minúscula	Òò	O mayúscula/o minúscula con acento grave
Tt	T mayúscula/t minúscula	Óó	O mayúscula/o minúscula con acento agudo
Uu	U mayúscula/u minúscula	Ôô	O mayúscula/o minúscula con acento circunflejo
Vv	V mayúscula/v minúscula	Õõ	O mayúscula/o minúscula con tilde
Ww	W mayúscula/w minúscula	Öö	O mayúscula/o minúscula con diéresis
Xx	X mayúscula/x minúscula	Øø	O mayúscula/o minúscula con barra oblicua
Yy	Y mayúscula/y minúscula	Ùù	U mayúscula/u minúscula con acento grave
Zz	Z mayúscula/z minúscula	Úú	U mayúscula/u minúscula con acento agudo
		Ûû	U mayúscula/u minúscula con acento circunflejo
		Üü	U mayúscula/u minúscula con diéresis
		Ýý	Y mayúscula/y minúscula con acento agudo
		Þþ	thorn islandesa mayúscula/minúscula
		ß	Doble S alemana minúscula
		ÿ	Y minúscula con diéresis

Cuadro 3 – Dígitos decimales

0 1 2 3 4 5 6 7 8 9

Cuadro 4 – Los 65 caracteres gráficos

Carácter	Descripción	Carácter	Descripción
!	Signo de admiración	¡	Signo de admiración invertido
"	Comillas dobles	¢	Signo de centavo
#	Signo de número	£	Signo de libra
$	Signo de dólar	¤	Signo de moneda
%	Signo de tanto por ciento	¥	Signo de yen
&	y comercial	¦	Barra interrumpida
’	Apóstrofo	§	Signo de párrafo
(	Paréntesis izquierdo	¨	Diéresis
)	Paréntesis derecho	©	Signo de derechos de autor
*	Asterisco	ª	Indicador ordinal femenino
+	Signo más	«	Comillas angulares izquierdas
’	Coma	¬	Signo de negación
-	Guión, signo menos	–	Guión de corte programable
.	Punto	®	Signo de marca registrada
/	Barra oblicua	¯	Signo de vocal larga (macrón)
:	Dos puntos	°	Cero volado, signo de grado
;	Punto y coma	±	Signo más menos
<	Signo menor que	²	Exponente dos
=	Signo igual	³	Exponente tres
>	Signo mayor que	´	Acento agudo
?	Signo de interrogación (final)	µ	Signo de micro
@	«a» comercial	¶	Signo de calderón
[	Corchete izquierdo	·	Punto central
\	Barra oblicua inversa	¸	Cedilla
]	Corchete derecho	¹	Exponente uno
^	Acento circunflejo	º	Indicador ordinal masculino
_	Línea baja de subrayado	»	Comillas angulares derechas
`	Acento grave	¼	Fracción vulgar un cuarto
{	Llave de imprenta izquierda	½	Fracción vulgar un medio
|	Línea vertical	¾	Fracción vulgar tres cuartos
}	Llave de imprenta derecha	¿	Signo de interrogación (inicial)
~	Tilde	×	Signo de multiplicación
		÷	Signo de división

Cuadro 5 – Caracteres de formateo

Descripción	Abreviatura	Valor octal ISO/CEI 646
Aviso	BEL	007
Retroceso	BS	010
Tabulación horizontal	HT	011
Nueva línea	NL, LF	012
Tabulación vertical	VT	013
Página siguiente	FF	014
Retorno del carro	CR	015

Las palabras clave obedecen a las reglas aplicables a los identificadores (véase 1.3) y deben escribirse exactamente como se indica en la lista que figura más arriba. Por ejemplo, "`boolean`" es correcto; "`Boolean`" es incorrecto. Las especificaciones ODP IDL utilizan los caracteres que se indican como puntuación en el cuadro 7.

Además, el preprocesador utiliza los testigos enumerados en el cuadro 8.

Cuadro 6 – Palabras clave

any	default	in	oneway	struc	wchar
attribute	double	inout	out	switch	wstring
boolean	enum	interface	raises	TRUE	
case	exception	long	readonly	typedef	
char	FALSE	module	sequence	unsigned	
const	fixed	Object	short	union	
context	float	octet	string	void	

Cuadro 7 – Testigos de puntuación

;	{	}	:	::	,	=	+	-	(	)	<	>	[	]
’	"	\	|	^	&	*	/	%	~	<<	>>				

Cuadro 8 – Testigos de preprocesador

#	##	!	||	&&	include	pragma	define

4.1.5 Literales

Literales de carácter y de cadena amplios se especifican exactamente igual que literales de carácter y de cadena. Todos los literales y de cadena amplios y no amplios, sólo pueden especificarse (de manera portable) utilizando los caracteres que se encuentra en el conjunto de caracteres de ISO/CEI 8859-1. Hay que destacar que estas extensiones para caracteres internacionales sólo afectan la especificación de literales en el ODP IDL y no al resto de ficheros fuente ODP IDL. Es decir, los nombres de interfaz, nombres de operación, nombres de tipo, etc., continuarán estando limitados por el conjunto de caracteres de ISO/CEI 8859-1.

Los literales y los nuevos tipos entero y de coma flotante se especifican como se indica en esta subcláusula (*Literales enteros* y *Literales en coma flotante*).

4.1.5.1 Literales enteros

Se considera que un literal entero consistente en una secuencia de dígitos es decimal (base diez) a menos que empiece con 0 (dígito cero). Se considera que una secuencia de dígitos que comienza con 0 es un entero octal (base ocho). Los dígitos 8 y 9 no son dígitos octales. Se considera que una secuencia de dígitos precedida por 0x o 0X es un entero hexadecimal (base dieciséis). Los dígitos hexadecimales incluyen una a o A a través de f o F con valores decimales de 10 a 15, respectivamente. Por ejemplo, el número doce puede escribirse 12, 014 o 0XC.

4.1.5.2 Literales de carácter

Un literal de carácter es uno o más caracteres encerrados en comillas simples, como en 'x'. Los literales de carácter tienen el tipo `char`.

Un carácter es una cantidad de 8 bits con un valor numérico entre 0 y 255 (decimal). El valor de un literal de carácter de espacio, alfabético, dígito o gráfico es el valor numérico del carácter definido en la norma de juego de caracteres ISO/CEI Latin-1 (ISO/CEI 8859-1) (véanse los cuadros 2, 3 y 4). El valor de un nulo es 0. El valor de un literal de carácter de formateo es el valor numérico del carácter tal como se define en ISO/CEI 646 (véase el cuadro 5). El significado de todos los otros caracteres es dependiente de la implementación.

Los caracteres no gráficos deben representarse utilizando secuencias de escape como las que se definen a continuación en el cuadro 9. Obsérvese que las secuencias de escape deben usarse para representar caracteres de comilla simple y barra oblicua inversa en literales de carácter.

Si el carácter que sigue a una barra oblicua inversa no es uno de los especificados, el comportamiento no es definido. Una secuencia de escape especifica un carácter único.

El escape \000 consiste en la barra oblicua inversa seguida de uno, dos o tres dígitos octales que se considera que especifican el valor del carácter deseado. El escape \xhh consiste en la barra oblicua inversa seguida de x seguida de uno o dos dígitos hexadecimales que se considera que especifican el valor del carácter deseado. Una secuencia de dígitos octales o hexadecimales termina en el primer carácter que no es un dígito octal o un dígito hexadecimal, respectivamente. El valor de una constante de carácter es dependiente de la implementación si es superior al valor del carácter (char) más largo.

Cuadro 9 – Secuencias de escape

Descripción	Secuencia de escape
Nueva línea	\n
Tabulación horizontal	\t
Tabulación vertical	\v
Retroceso	\b
Retorno de carro	\r
Página siguiente	\f
Aviso	\a
Barra oblicua inversa	\\
Signo de interrogación	\?
Comilla simple	\'
Comilla doble	\"
Número octal	\ooo
Número hexadecimal	\xhh

4.1.5.3 Literales de coma flotante

Un literal de coma flotante consiste en una parte entera, un punto decimal, una parte fraccionaria, una e o E y un exponente entero facultativamente con signo. Las partes entera y fraccionaria consisten en una secuencia de dígitos decimales (base diez). Puede faltar tanto la parte entera como la parte fraccionaria (pero no ambas); puede faltar el punto decimal o la letra e (o E) y el exponente (pero no ambos).

4.1.5.4 Literales de punto fijo

Un literal de punto decimal fijo consiste en una parte entera, un punto decimal, una parte fraccionaria y una d o D. Las partes entera y fraccionaria consisten en una secuencia de dígitos decimales (base 10). Puede faltar tanto la parte entera como la parte fraccionaria (pero no ambas); puede faltar el punto decimal [pero no la letra d (o D)].

4.1.5.5 Literales de cadena

Un literal de cadena es una secuencia de caracteres (según se define en 4.1.5.2) encerrada entre comillas dobles, como en "...".

Los literales de cadena adyacente están concatenados. Los caracteres en cadenas concatenadas se mantienen distintos. Por ejemplo,

"\xA" "B" contienen los dos caracteres '\xA' y 'B' después de la concatenación (y no el único carácter hexadecimal '\xAB').

El tamaño de un literal de cadena es el número de literales de carácter encerrados entre comillas, después de la concatenación. El tamaño del literal está asociado con el literal.

4.2 Preprocesamiento

Se puede utilizar una notación de preprocesamiento como una notación de módulo, con miras a organizar las especificaciones y poder referirse a partes de una especificación en una especificación dada. Por lo tanto, la inclusión del fichero fuente #include debe interpretarse como una forma genérica de incluir un módulo de especificación dado y no está vinculado con ningún sistema particular de archivado.

El preprocesamiento ODP IDL especificado en la *Norma ANSI/ISO C++* proporciona macrosustitución, compilación condicional e inclusión de fichero fuente. Además, se proporcionan directivas para controlar la numeración de línea en diagnósticos y para la depuración simbólica, para generar un mensaje de diagnóstico con determinada secuencia de testigos, y para realizar acciones dependientes de la implementación (la directiva pragma #). Se dispone de ciertos nombres predefinidos. Un preprocesador trata conceptualmente estas facilidades, que pueden o no ser realmente implementadas como un proceso separado.

Las líneas que comienzan con # (también llamadas "directivas") se comunican con este preprocesador. Puede aparecer un espacio en blanco antes de #. Estas líneas tienen una sintaxis independiente del resto del ODP IDL; pueden aparecer en cualquier sitio y sus efectos duran (independientemente de las reglas de fijación de ámbito ODP IDL) hasta el final de la unidad de traducción. La ubicación textual de pragmas específicos del ODP IDL puede ser constreñida semánticamente.

Una directiva (o cualquier línea) de preprocesamiento puede ser continuada en la siguiente línea en un fichero fuente mediante la ubicación de un carácter de barra oblicua invertida ("\") inmediatamente antes de la nueva línea al final de la línea que se va a continuar. El preprocesador efectúa la continuación suprimiendo la barra oblicua invertida, y la nueva línea antes de la secuencia de entrada se divide en testigos. Un carácter de barra oblicua invertida puede no ser el último carácter de un fichero fuente.

Un testigo de preprocesamiento es un testigo ODP IDL (véase 4.1.1), un nombre de fichero como en una directiva `#include`, o cualquier carácter único, distinto del espacio en blanco, que no coincida con otro testigo de preprocesamiento.

Las facilidades de preprocesamiento se utilizan principalmente para incluir definiciones de otras especificaciones ODP IDL. El texto en los ficheros incluido con una directiva `#include` se trata como si apareciera en el fichero que lo incluye. La *Norma ANSI/ISO C++* contiene una descripción completa de las facilidades de preprocesamiento.

4.3 Gramática ODP IDL

```
(1)   <specification>          ::=  <definition>+
(2)   <definition>             ::=  <type_dcl>";"
                                |   <const_dcl>";"
                                |   <except_dcl>";"
                                |   <interface>";"
                                |   <module>";"
(3)   <module>                 ::=  "module"  <identifier> "{"  <definition>+ "}"
(4)   <interface>              ::=  <interface_dcl>
                                |   <forward_dcl>
(5)   <interface_dcl>          ::=  <interface_header> "{" <interface_body> "}"
(6)   <forward_dcl>            ::=  "interface" <identifier>
(7)   <interface_header>       ::=  "interface" <identifier> [<inheritance_spec> ]
(8)   <interface_body>         ::=  <export>*
(9)   <export>                 ::=  <type_dcl> ";"
                                |   <const_dcl> ";"
                                |   <except_dcl> ";"
                                |   <attr_dcl> ";"
                                |   <op_dcl> ";"
(10)  <inheritance_spec>       ::=  ":" <scoped_name> { "," <scoped_name> }*
(11)  <scoped_name>            ::=  <identifier>
                                |   "::" <identifier>
                                |   <scoped_name> "::" <identifier>
(12)  <const_dcl>              ::=  "const" <const_type> <identifier> "=" <const_exp>
```

```
(13)   <const_type>           ::=  <integer_type>
                                |   <char_type>
                                |   <wide_char_type>
                                |   <boolean_type>
                                |   <floating_pt_type>
                                |   <string_type>
                                |   <wide_string_type>
                                |   <fixed_pt_const_type>
                                |   <scoped_name>
(14)   <const_exp>            ::=  <or_exp>
(15)   <or_exp>               ::=  <xor_expr>
                                |   <or_expr> "|" <xor_expr>
(16)   <xor_expr>             ::=  <and_expr>
                                |   <xor_expr> "^" <and_expr>
(17)   <and_expr>             ::=  <shift_expr>
                                |   <and_expr> "&" <shift_expr>
(18)   <shift_expr>           ::=  <add_expr>
                                |   <shift_expr> ">>" <add_expr>
                                |   <shift_expr> "<<" <add_expr>
(19)   <add_expr>             ::=  <mult_expr>
                                |   <add_expr> "+" <mult_expr>
                                |   <add_expr> "-" <mult_expr>
(20)   <mult_expr>            ::=  <unary_expr>
                                |   <mult_expr> "*" <unary-expr>
                                |   <mult_expr> "/" <unary_expr>
                                |   <mult_expr> "%" <unary_expr>
(21)   <unary_expr>           ::=  <unary_operator> <primary_expr>
                                |   <primary_expr>
(22)   <unary_operator>       ::=  "-"
                                |   "+"
                                |   "~"
(23)   <primary_expr>         ::=  <scoped_name>
                                |   <literal>
                                |   "(" <const_exp> ")"
(24)   <literal>              ::=  <integer_literal>
                                |   <string_literal>
                                |   <wide_string_literal>
                                |   <character_literal>
                                |   <wide_character_literal>
                                |   <fixed_pt_literal>
                                |   <floating_pt_literal>
                                |   <boolean_literal>
(25)   <boolean_literal>      ::=  "TRUE"
                                |   "FALSE"
(26)   <positive_int_const>   ::=  <const_exp>
```

```
(27)    <type_dcl>              ::=  "typedef" <type_declarator>
                                |    <struct_type>
                                |    <union_type>
                                |    <enum_type>
(28)    <type_declarator>       ::=  <type_spec> <declarators>
(29)    <type_spec>             ::=  <simple_type_spec>
                                |    <constr_type_spec>
(30)    <simple_type_spec>      ::=  <base_type_spec>
                                |    <template_type_spec>
                                |    <scoped_name>
(31)    <base_type_spec>        ::=  <floating_pt_type>
                                |    <integer_type>
                                |    <char_type>
                                |    <wide_char_type>
                                |    <boolean_type>
                                |    <octet_type>
                                |    <any_type>
                                |    <object_type>
(31a)   <object_type>           ::=  "Object"
(32)    <template_type_spec>    ::=  <sequence_type>
                                |    <string_type>
                                |    <wide_string_type>
                                |    <fixed_pt_type>
(33)    <constr_type_spec>      ::=  <struct_type>
                                |    <union_type>
                                |    <enum_type>
(34)    <declarators>           ::=  <declarator> { "," <declarator> } *
(35)    <declarator>            ::=  <simple_declarator>
                                |    <complex_declarator>
(36)    <simple_declarator>     ::=  <identifier>
(37)    <complex_declarator>    ::=  <array_declarator>
(38)    <floating_pt_type>      ::=  "float"
                                |    "double"
                                |    "long" "double"
(39)    <integer_type>          ::=  <signed_int>
                                |    <unsigned_int>
(40)    <signed_int>            ::=  <signed_long_int>
                                |    <signed_short_int>
                                |    <signed_longlong_int>
(40a)   <signed_longlong_int>   ::=  "long" "long"
(41)    <signed_long_int>       ::=  "long"
(42)    <signed_short_int>      ::=  "short"
(43)    <unsigned_int>          ::=  <unsigned_long_int>
                                |    <unsigned_short_int>
                                |    <unsigned_longlong_int>
```

```
(43a)  <unsigned_longlong_int> ::= "unsigned" "long" "long"
(44)   <unsigned_long_int>    ::=  "unsigned" "long"
(45)   <unsigned_short_int>   ::=  "unsigned" "short"
(46)   <char_type>            ::=  "char"
(46a)  <wide_char_type>       ::=  "wchar"
(47)   <boolean_type>         ::=  "boolean"
(48)   <octet_type>           ::=  "octet"
(49)   <any_type>             ::=  "any"
(50)   <struct_type>          ::=  "struct" <identifier> "{" <member_list> "}"
(51)   <member_list>          ::=  <member>+
(52)   <member>               ::=  <type_spec> <declarators> ";"
(53)   <union_type>           ::=  "union"<identifier> "switch"
                                        "(" <switch_type_spec> ")" "{" <switch_body> "}"
(54)   <switch_type_spec>     ::=  <integer_type>
                               |    <char_type>
                               |    <boolean_type>
                               |    <enum_type>
                               |    <scoped_name>
(55)   <switch_body>          ::=  <case>+
(56)   <case>                 ::=  <case_label>+ <element_spec> ";"
(57)   <case_label>           ::=  "case" <const_exp> ":"
                               |    "default" ":"
(58)   <element_spec>         ::=  <type_spec> <declarator>
(59)   <enum_type>            ::=  "enum" <identifier> "{" <enumerator>
                                        {"," <enumerator> } * "}"
(60    <enumerator>           ::=  <identifier>
(61)   <sequence_type>        ::=  "sequence" "<" <simple_type_spec> ","
                                        <positive_int_const> ">"
                               |    "sequence" "<" <simple_type_spec> ">"
(62)   <string_type>          ::=  "string" "<" <positive_int_const> ">
                               |    "string"
(62a)  <wide_string_type>     ::=  "wstring" "<" <positive_int_const>
                                        ">" | "wstring"
(63)   <array_declarator>     ::=  <identifier> <fixed_array_size>+
(64)   <fixed_array_size>     ::=  "[" <positive_int_const> "]"
(65)   <attr_dcl>             ::=  [ "readonly" ] "attribute"<param_type_spec>
                                        <simple_declarator>
                                        { "," <simple_declarator> }*
(66)   <except_dcl>           ::=  "exception" <identifier> "{" <member>*"}"
(67)   <op_dcl>               ::=  [ <op_attribute> ] <op_type_spec>
                                        <identifier> <parameter_dcls>
                                        [ <raises_expr> ] [ <context_expr> ]
(68)   <op_attribute>         ::=  "oneway"
(69)   <op_type_spec>         ::=  <param_type_spec>
                               |    "void"
(70)   <parameter_dcls>       ::=  "(" <param_dcl> { "," <param_dcl> }* ")"
                               |    "(" ")"
(71)   <param_dcl>            ::=  <param_attribute> <param_type_spec>
                                        <simple_declarator>
```

```
(72)   <param_attribute>        ::=  "in"
                                |    "out"
                                |    "inout"
(73)   <raises_expr>            ::=  "raises" "(" <scoped_name> { ","<scoped_name> }* ")"
(74)   <context_expr>           ::=  "context "(" <string_literal>
                                          { ","<string_literal> } * ")"
(75)   <param_type_spec>        ::=  <base_type_spec>
                                |    <string_type>
                                |    <fixed_pt_type>
                                |    <wide_string_type>
                                |    <scoped_name>
(76)   <fixed_pt_type>          ::=  "fixed" "<" <positive_int_const>
                                          ","<integer_literal> ">"
(77)   <fixed_pt_const_type>    ::=  "fixed"
```

4.4 Especificación ODP IDL

Una especificación ODP IDL consiste en una o más definiciones de tipo, definiciones de constante, definiciones de excepción, o definiciones de módulo. La sintaxis es:

```
<specification>::=      <defintion> +
<definition>            ::=  <type_dcl> ";"
                        |    <const_dcl> ";"
                        |    <except_dcl> ";"
                        |    <interface_dcl> ";"
                        |    <module_dcl> ";"
```

Para las especificaciones de `<const_dcl>, <type_dcl>,` y `<except_dcl>`, véanse 4.6, 4.7 y 4.9, respectivamente.

4.4.1 Declaración de módulo

Una definición de módulo cumple con la siguiente sintaxis:

```
<module>                ::=  "module" <identifier> "{ <definition>+"}"
```

El módulo construcción se utiliza para fijar el ámbito de la aplicación de los identificadores ODP IDL; para mayores detalles véase 4.12.

4.4.2 Declaración de interfaz

Una declaración de interfaz cumple con la siguiente sintaxis:

```
<interface>             ::=  <interface_dcl>
                        |    <forward_dcl>
<interface_dcl>         ::=  <interface_header> "{" <interface_body> "}"
<forward_dcl>           ::=  "interface" <identifier>
<interface_header>      ::=  "interface: <identifier> [ <inheritance_spec> ]
<interface_body>        ::=  <export>*
<export>                ::=  <type_dcl> ";"
                        |    <const_dcl> ";"
                        |    <except_dcl> ";"
                        |    <attr_dcl>";"
                        |    <op_dcl>";"
```

4.4.2.1 Encabezamiento de interfaz

El encabezamiento de interfaz consta de dos elementos:

- El nombre de la interfaz – El nombre debe ir precedido por la palabra clave `interface`, y consiste en un identificador que nombra la interfaz.
- Una especificación de herencia facultativa – La especificación de herencia se describe en 4.4.2.2.

El `<identifier>` que nombra una interfaz define un nombre de tipo legal. Este nombre de tipo se puede utilizar en cualquier sitio en el cual `<identifier>` sea legal en la gramática, con sujeción a las restricciones semánticas que se describen en las subcláusulas siguientes. El nombre `Object` es un nombre válido que permite transferir cualquier referencia de interfaz. Puesto que sólo se pueden retener referencias a una interfaz, el significado de un parámetro o miembro de estructura que es un tipo de interfaz es una *referencia* a una instancia de ese tipo de interfaz. Cada vinculación de lenguaje describe la forma según la cual el programador debe representar esas referencias de interfaz. Se puede utilizar, en particular, como parámetro en una descripción de operación lo que permite transmitir referencias de interfaz como parámetros.

4.4.2.2 Especificación de herencia

La sintaxis para la herencia es la siguiente:

```
<inheritance_spec>      ::=  ":" <scoped_name> { "," <scoped_name> }*

<scoped_name>           ::=  <identifier>

                        |    "::" <identifier>

                        |    <scoped_name>"::" <identifier>
```

Cada `<scoped_name>` en un `<inheritance_spec>` debe denotar una interfaz previamente definida. Para una descripción de herencia véase 4.5.

4.4.2.3 Cuerpo de interfaz

El cuerpo de interfaz contiene los siguientes tipos de declaraciones:

- Declaraciones de constante, que especifican las constantes que exporta la interfaz; la sintaxis de declaración de constante se describe en 4.6.
- Declaraciones de tipo, que especifican las definiciones de tipo que exporta la interfaz; la sintaxis de declaración de tipo se describe en 4.7.
- Declaraciones de excepción, que especifican las estructuras de excepción que exporta la interfaz; la sintaxis de declaración de excepción se describe en 4.9.
- Declaraciones de atributo, que especifican los atributos asociados exportados por la interfaz; la sintaxis de declaración de atributo se describe en 4.11.
- Declaraciones de operación, que especifican las operaciones que exporta la interfaz y el formato de cada una de ellas, incluido el nombre de la operación, el tipo de datos devueltos, los tipos de todos los parámetros de una operación, las excepciones legales que pueden devolverse como resultado de una invocación, y la información contextual que puede afectar el despacho de método; la sintaxis de declaración de operación se describe en 4.10.

Están permitidas las interfaces vacías (es decir, las que no contienen declaración alguna).

4.4.2.4 Declaración hacia adelante

Una declaración hacia adelante declara el nombre de una interfaz sin definirla. Esto permite la definición de interfaces que se refieren una a la otra. La sintaxis consiste simplemente en la palabra clave `interface` seguida por un `<identifier>` que nombra la interfaz. La definición real debe figurar más adelante en la especificación.

Son legales múltiples declaraciones hacia adelante en la misma interfaz.

4.5 Herencia

Una interfaz puede obtenerse a partir de otra interfaz, que entonces se llama interfaz *de base* de la interfaz derivada. Una interfaz derivada, como todas las interfaces, puede declarar nuevos elementos (constantes, tipos, atributos, excepciones y operaciones). Además, a menos que se vuelvan a definir en la interfaz derivada, se puede hacer referencia a los elementos de una interfaz de base como si fueran elementos de la interfaz derivada. Puede utilizarse un operador de resolución de nombre ("::") para hacer referencia explícitamente a un elemento de base; esto permite hacer referencia a un nombre que ha sido redefinido en la interfaz derivada.

Una interfaz derivada puede redefinir cualquier nombre de tipo, constante y excepción que haya sido heredado; las reglas de ámbito de aplicación para esos nombres se describen en 4.12.

Una interfaz se llama base directa si está mencionada en `<inheritance_spec>` y base indirecta si no es una base directa pero es una interfaz de base de una de las interfaces mencionadas en `<inheritance_spec>`.

Una interfaz puede obtenerse a partir de cualquier número de interfaces de base. A menudo la utilización de más de una interfaz de base directa se denomina herencia múltiple. El orden de obtención no es significante.

Una interfaz no puede ser especificada como interfaz de base directa de una interfaz derivada más de una vez; pero puede ser una interfaz de base indirecta más de una vez. Considérese el siguiente ejemplo:

```
interface A {...}
interface B:A {...}
interface C:A {...}
interface D:B,C {...}
```

Las relaciones entre estas interfaces se indican en la figura 1. Esta forma de "diamante" es legal.

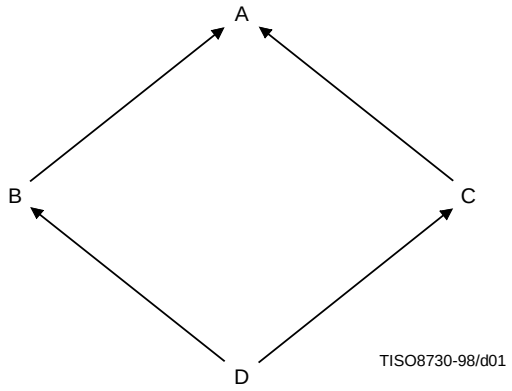

Figura 1 – Ejemplo de herencia múltiple legal

La referencia a los elementos de interfaz de base debe ser inequívoca. La referencia a un elemento de interfaz de base es ambigua si la expresión utilizada se refiere a un tipo de constante, o excepción en más de una interfaz de base. (Actualmente es ilegal heredar de dos interfaces con el mismo nombre de operación o atributo, o redefinir un nombre de operación o atributo en la interfaz derivada.) Las ambigüedades pueden resolverse mediante la calificación de un nombre con su nombre de interfaz (es decir, utilizando un `<scoped name>`).

Las referencias a constantes, tipos y excepciones están vinculadas a una interfaz cuando ésta está definida, es decir, sustituida por el `<scoped name>` equivalente global. Esto asegura que la sintaxis y la semántica de una interfaz no se cambian cuando la interfaz es una interfaz de base para una interfaz derivada. Considérese el siguiente ejemplo:

```
const long L=3;
interface A {
         typedef float coord[L]);
      void f (in coord s); // s has three floats };
```

```
interface B{
      const long L=4;
};
interface C: B, A {}                //what is f()'s signature?
```

La vinculación oportuna de constantes, tipos y excepciones en la definición de interfaz garantiza que la signatura de la operación `f` en la interfaz `C` sea:

```
typedef float coord[3];
void f (in coord s);
```

que es idéntica a la de la interfaz `A`. Esta regla también impide que la redefinición de una constante, un tipo o una excepción en la interfaz derivada afecte las operaciones y atributos heredados de una interfaz de base.

La herencia de interfaz hace que todos los identificadores en el cierre del árbol de herencia sean importados al ámbito de denominación vigente. Un nombre de tipo, de constante, de valor de enumeración o de excepción de un ámbito abarcador puede volverse a definir en el ámbito vigente.

Todas las operaciones que se puedan aplicar a determinado objeto deben tener nombres únicos. Este requisito prohíbe la redefinición de un nombre de operación en una interfaz derivada, así como la herencia de dos operaciones con el mismo nombre.

NOTA – Se prevé que en futuras revisiones del lenguaje esta regla será de alguna manera menos rigurosa, y quizá permita la sobrecarga o proporcione algún medio para hacer una distinción entre operaciones con el mismo nombre.

4.6 Declaración de constante

4.6.1 Sintaxis

En esta subcláusula se describe la sintaxis para las declaraciones de constante.

La sintaxis de una declaración de constante es:

```
<const_dcl>             ::=  "const" <const_type> <identifier> "=" <const_exp>
<const_type>            ::=  <integer_type>
                        |    <char_type>
                        |    <wide_char_type>
                        |    <boolean_type>
                        |    <floating_pt_type>
                        |    <string_type>
                        |    <wide_string_type>
                        |    <fixed_pt_const_type>
                        |    <scoped_name>
<const_exp>             ::=  <or_exp>
<or_exp>                ::=  <xor_expr>
                        |    <or_expr> "|" <xor_expr>
<xor_expr>              ::=  <and_expr>
                        |    <xor_expr> "^" <and_expr>
<and_expr>              ::=  <shift_expr>
                        |    <and_expr> "&" <shift_expr>
<shift_expr>            ::=  <add_expr>
                        |    <shift_expr> ">>" <add_expr>
                        |    <shift_expr> "<<" <add_expr>
<add_expr>              ::=  <mult_expr>
                        |    <add_expr> "+" <mult_expr>
                        |    <add_expr> "-" <mult_expr>
```

```
<mult_expr>             ::=  <unary_expr>
                        |    <mult_expr> "*" <unary-expr>
                        |    <mult_expr> "/" <unary_expr>
                        |    <mult_expr> "%" <unrary_expr>
<unary_expr>            ::=  <unary_operator> <primary_expr>
                        |    <primary_expr>
<unary_operator>        ::=  "-"
                        |    "+"
                        |    "~"
<primary_expr>          ::=  <scoped_name>
                        |    <literal>
                        |    "(" <const_exp> ")"
<literal>               ::=  <integer_literal>
                        |    <string_literal>
                        |    <character_literal>
                        |    <wide_character_literal>
                        |    <fixed_pt_literal>
                        |    <floating_pt_literal>
                        |    <boolean_literal>
<boolean_literal>       ::=  "TRUE"
                        |    "FALSE"
<positive_int_const>::= <const_exp>
```

4.6.2 Semántica

`<scoped_name>` en la producción `<const_type>` debe ser el nombre previamente definido de una constante `<integer_type>`, `<char_type>`, `<wide_char_type>`, `<boolean_type>`, `<floating_pt_type>`, `<fiexed_pt_const_type>`, `<string_type>`, o `<wide_string_type>`.

Un operador infijo puede combinar dos enteros, flotantes o fijos, pero no mezclas de éstos. Los operadores infijos no se pueden aplicar a otros tipos distintos de los enteros, de coma flotante y fijos.

Una expresión constante entera se evalúa como long long sin signo a menos que contenga un literal entero negado o el nombre de una constante entera con un valor negativo. En este último caso, la expresión constante se evalúa como long long con signo. El valor calculado se fuerza para que tome el tipo deseado en los inicializadores de constantes. Si el valor calculado supera la precisión del tipo deseado, hay un error. Si cualquier valor intermedio supera la gama del tipo evaluado (long long o long long sin signo), hay un error.

Todos los literales de coma flotante son long double, todas las constantes de coma flotante se fuerzan para que tomen el valor long double, y todas la expresiones de coma flotante se calculan como long doubles. El valor long double calculado se vuelve a forzar al tipo deseado en los inicializadores constantes. Si esta coacción fracasa o si cualquier valor intermedio (al evaluar la expresión) supera la gama de long double, hay un error.

Los operadores unarios (+) y binarios (* / + –) se pueden aplicar en las expresiones de coma flotante. Los operadores unarios (+ – ~) y binarios (* / % + – < < > > & | ^) se pueden aplicar en expresiones enteras.

El operador unario "~" indica que se debe generar el complemento binario de la expresión a la cual se aplica. A los efectos de esas expresiones, los valores son números complementarios de 2. Como tales, el complemento puede generarse como sigue:

`long long` –(value+1)

`unsigned long long` (2**64–1) – value

El operador binario "%" genera el resto de la división de la primera expresión por la segunda. Si el segundo operando de "%" es 0, el resultado es indefinido; de otro modo

```
(a/b)*b +a%b
```

es igual a a. Si ambos operandos son no negativos, el resto es no negativo; si no, el signo del resto depende de la implementación.

El operador binario "<<" indica que el valor del operando izquierdo debe desplazarse hacia la izquierda el número de bits especificado por el operando derecho, rellenando con 0 los bits vacantes. El operando derecho debe estar en la gama 0 ≤ operando derecho < 32.

El operador binario "&" indica que se debe generar el operador lógico AND binario de los operandos izquierdo y derecho.

El operador binario "|" indica que se debe generar el operador lógico OR binario de los operandos izquierdo y derecho.

El operador binario "∧" indica que se debe generar el operador lógico EXCLUSIVE-OR binario de los operandos izquierdo y derecho.

El valor de `<positive_int_const>` debe evaluarse a una constante entera positiva.

Las expresiones de constante de punto decimal fijo se evalúan como sigue. Un literal de punto fijo tiene el número aparente de dígitos totales y fraccionales, excepto los iniciales o los que están seguidos de ceros que no se consideran, incluidos los ceros no significativos antes del punto decimal. Por ejemplo, `0123.450d` se considera `fijo<5,2>` y `3000.00` es `fijo<1,-3>`. Los operadores prefijados no afectan la precisión ; un prefijo + es optativo, y no modifica el resultado. El cuadro siguiente muestra las vinculaciones superiores en el número de dígitos y en la escala el resultado de una expresión infix, `fixed<d1,s1> opfixed<d2,s2>`:

Op	Result: `fixed<d,s>`
+	`fixed<max(d1-s1,d2-s2) + max(s1,s2) + 1, max(s1,s2)>`
-	`fixed<max(d1-s1,d2-s2) + max(s1,s2) + 1, max(s1,s2)>`
*	`fixed<d1+d2, s1+s2>`
/	`fixed<(d1-s1+s2) + s inf , s inf >`

Un cociente puede tener un número arbitrario de posiciones decimales, definido por una escala `s` inf. El cálculo procede por pares, con las normas habituales para asociaciones de izquierda a derecha, precedencia de operador y paréntesis. Si un determinado cálculo entre un par de literales de punto fijo genera realmente mas de 31 dígitos significativos, entonces se mantiene un resultado de 31 dígitos como sigue :

```
fixed<d,s> => fixed<31, 31-d+s>
```

Los ceros iniciales y finales no se consideran significativos. Se rechazan los dígitos omitidos; no se realiza redondeo. El resultado del cálculo procede entonces como un operando literal del próximo par de literales de punto fijo que han de calcularse. Los operadores unario (`+ -`) y binario (`* / + -`) son aplicables en expresiones de coma flotante y de punto fijo. Los operadores unario (`+ – ~`) y binario (`* / % + – << >> & | ^`) son aplicables en expresiones enteras.

4.7 Declaración de tipo

El ODP IDL proporciona constructivos para la denominación de tipos de datos; es decir, proporciona declaraciones de lenguaje C que asocian un identificador con un tipo. El ODP IDL utiliza la palabra clave `typedef` para asociar un nombre con un tipo de datos. Un nombre también se asocia con un tipo de datos a través de las declaraciones `struct`, `union` y `enum`; la sintaxis es:

```
<type_dcl>              ::=  "typedef" <type_declarator>
                        |    <struct_type>
                        |    <union_type>
                        |    <enum_type
<type_declarator>       ::= <type_spec> <declarators>
```

Para las declaraciones de tipo, el ODP IDL define un conjunto de especificadores de tipo para representar valores tipificados. La sintaxis es la siguiente:

```
<type_spec>             ::=  <simple_type_spec>
                        |    <constr_type_spec>
<simple_type_spec>      ::=  <base_type_spec>
                        |    <template_type_spec>
                        |    <scoped_name>
<base_type_spec>        ::=  <floating_pt_type>
                        |    <integer_type>
                        |    <char_type>
                        |    <wide_char_type>
                        |    <boolean_type>
                        |    <octet_type>
                        |    <any_type>
                        |    <object_type>
<template_type_spec>    ::=  <sequence_type>
                        |    <string_type>
<constr_type_spec>      ::=  <struct_type>
                        |    <union_type>
                        |    <enum_type>
<declarators>           ::=  <declarator> { "," <declarator> } *
<declarator>            ::=  <simple_declarator>
                        |    <complex_declarator>
<simple_declarator>     ::=  <identifier>
<complex_declarator>    ::=  <array_declarator>
```

Como se observa de lo que antecede, los especificadores de tipo ODP IDL consisten en tipos de datos escalares y constructores de tipo. Los especificadores de tipo ODP IDL pueden utilizarse en declaraciones de operación para asignar tipos de datos a los parámetros de operación. En las siguientes subcláusulas se describen los especificadores de tipo básico y construido.

4.7.1 Tipos básicos

La sintaxis de los tipos básicos soportados es la siguiente:

```
<floating_pt_type>      ::=  "float"
                        |    "double"
                        |    "long" "double"
<integer_type>          ::=  <signed_int>
                        |    <unsigned_int>
<signed_int>            ::=  <signed_long_int>
                        |    <signed_short_int>
                        |    <signed_longlong_int>
<signed_long_int>       ::=  "long"
<signed_short_int>      ::=  "short"
<signed_longlong_int>   ::=  "long" "long"
<unsigned_int>          ::=  <unsigned_long_int>
                        |    <unsigned_short_int>
                        |    <unsigned_longlong_int>
<unsigned_long_int>     ::=  "unsigned" "long"
```

```
<unsigned_short_int>    ::=  "unsigned" "short"
<unsigned_longlong_int> ::=  "unsigned" "long" "long"
<char_type>             ::=  "char"
<wide_char_type>        ::=  "wchar"
<boolean_type>          ::=  "boolean"
<octet_type>            ::=  "octet"
<any_type>              ::=  "any"
<object_type>           ::=  "Object "
```

Cada tipo de datos ODP IDL se hace coincidir con un tipo de datos autóctono a través de la correspondencia de lenguaje adecuada. Durante la realización de una invocación de operación pueden surgir errores de conversión entre los tipos de datos ODP IDL y los tipos autóctonos a los que se corresponden.

4.7.1.1 Tipos enteros

El ODP IDL soporta tipos de datos enteros firmados y no firmados de precisión extendida (tipos de datos enteros `long`, `short`, `long long` y `unsigned`). El tipo `long` representa la gama -2^{31} .. $2^{31}-1$, `unsigned long` representa la gama 0 .. $2^{32}-1$, `short` representa la gama -2^{15} .. $2^{15}-1$, `unsigned short` representa la gama 0 .. $2^{16}-1$, `long long` representa la gama $-2^{63}-1$, mientras que el tipo `unsigned long long` representa la gama 0 .. $2^{64}-1$.

4.7.1.2 Tipo fijo

El tipo de datos `fixed` representa un número de punto decimal fijado de hasta 31 dígitos significativos. El factor escala es normalmente un entero no negativo inferior o igual al número total de dígitos (hay que destacar que siempre están permitidas constantes con escalas efectivamente negativas, como 10 000). Sin embargo, algunos lenguajes y entornos pueden ser capaces de acomodar tipos que tienen una escala negativa o una escala superior al número de dígitos.

4.7.1.3 Tipos de coma flotante

Los tipos de coma flotante ODP IDL son `float, double` y `long double`. El tipo `float` representa números de coma flotante de precisión simple IEEE; el tipo `double` representa números de coma flotante de precisión doble IEEE; el tipo `long double` representa números de coma flotante de extensión doble IEEE, que soportan un exponente por lo menos de 15 bits de longitud y una fracción de por lo menos 64 bits. Para mayor información sobre la precisión que proporcionan estos tipos, consúltese la especificación de la norma de coma flotante IEEE (*IEEE Standard for Binary Floating-Point Arithmetic,* ANSI/IEEE Std 754-1985).

En las implementaciones que no soportan totalmente el conjunto de valores de la norma de coma flotante IEEE 754 se deben especificar cabalmente sus diferencias con respecto a esta norma.

4.7.1.4 Tipo char

El ODP IDL define un tipo de datos `char`, que es una cantidad de 8 bits que:

1) codifica un carácter de byte único a partir de cualquier conjunto de código de bytes, o
2) cuando se utiliza información, codifica un carácter multibyte a partir de cualquier conjunto de código multibyte.

En otras palabras, una implementación es libre de utilizar cualquier conjunto de código internamente para codificar datos de carácter, aunque puede necesitarse para la transmisión su conversión a otra forma.

La norma de conjunto de caracteres ISO/CEI Latin-1 (ISO/CEI 8859-1) define por defecto el significado y la representación de todos los posibles caracteres gráficos (es decir, los caracteres de espacio, alfabéticos, dígitos y gráficos definidos en los cuadros 2, 3 y 4). El significado y la representación de los caracteres de nulo y formateo (véase el cuadro 5) es el valor numérico del carácter tal como se define en la norma ASCII (ISO/CEI 646). El significado de todos los otros caracteres depende de la implementación.

Durante la transmisión, los caracteres pueden convertirse a otras formas apropiadas, según las necesidades de una vinculación de lenguaje particular. Esas conversiones pueden modificar la representación de un carácter pero mantener el significado del mismo. Por ejemplo, un carácter puede convertirse a una representación adecuada en un conjunto de caracteres internacionales, y reconvertirse a partir de dicha representación.

4.7.1.5 Tipo wide char

El ODP IDL define un tipo de datos `wchar` que puede representar caracteres amplios a partir de cualquier conjunto de caracteres. Como ocurre con los datos de caracteres, una implementación es libre de utilizar cualquier conjunto de códigos internamente para representar caracteres amplios, aunque una vez más para la transmisión puede ser necesaria su conversión a otra forma. El tamaño de `wchar` depende de la implementación.

4.7.1.6 Tipo boolean

El tipo `boolean` se usa para denotar un elemento de datos que sólo puede tomar uno de los valores TRUE y FALSE.

4.7.1.7 Tipo octet

El tipo `octet` es una cantidad de 8 bits que se sabe que no sufrirá ninguna conversión al ser transmitida por el sistema de comunicación.

4.7.1.8 Tipo any

El tipo `any` permite la especificación de valores que pueden expresar cualquier tipo ODP IDL.

4.7.2 Tipos construidos

Los tipos construidos son:

```
<constr_type_spec>      ::= <struct_type>
                        |       <union_type>
                        |       <enum_type>
```

Como en el ODP IDL es sintácticamente posible generar especificaciones de tipo recursivo, esa recursión está semánticamente constreñida. La única forma permisible de especificación de tipo recursivo es a través del uso del tipo de plantilla `sequence`.

Por ejemplo, lo siguiente es legal:

```
struct foo {
        long value;
        sequence <foo> chain;
}
```

Para mayores detalles sobre el tipo de plantilla `sequence`, véase 4.7.3.1.

4.7.2.1 Estructuras

La sintaxis estructura es:

```
<struct_type>           ::=  "struct" <identifier> "{" <member_list> "}"
<member_list>           ::=  <member>+
<member>                ::=  <type_spec> <declarators> ";"
```

El `<identifier>` en `<struct_type>` define un nuevo tipo legal. Los tipos de estructura también pueden denominarse utilizando una declaración `typedef`.

Las reglas de fijación del ámbito del nombre exigen que los declaradores de miembro en una estructura particular sean únicos. El valor de una `struct` es el valor de todos sus miembros.

4.7.2.2 Uniones discriminadas

La sintaxis de `union` discriminada es:

```
<union_type>            ::=  "union"<identifier> "switch"
                                  "(" <switch_type_spec>"")" "{" <switch_body> "}"
<switch_type_spec>      ::=  <integer_type>
                        |    <char_type>
                        |    <boolean_type>
                        |    <enum_type>
                        |    <scoped_name>
```

```
<switch_body>          ::=  <case>+

<case>                 ::=  <case_label>+ <element_spec> ";"

<case_label>           ::=  "case" <const_exp> ":"

                       |    "default" ":"

<element_spec>         ::=  <type_spec> <declarator>
```

Las uniones ODP IDL son un cruce entre las declaraciones C `union` y `switch`. Las uniones ODP IDL deben estar discriminadas; esto es, el encabezamiento de la unión debe especificar un campo de rótulo tipificado que determina cuál miembro de la unión se ha de utilizar para la instancia vigente de una llamada. El `<identifier>` que sigue a la palabra clave `union` define un nuevo tipo legal. Los tipos de unión también pueden denominarse utilizando una declaración `typedef`. El `<const_exp>` en `<case_label>` debe ser coherente con `<switch_type_spec>`. Un caso `default` puede aparecer como máximo una vez. El `<scoped_name>` en la producción `<switch_type_spec>` debe ser un tipo `integer, char, boolean` o `enum` previamente definido.

Las etiquetas de caso deben coincidir o ser automáticamente moldeables con el tipo definido del discriminador. El cuadro 10 contiene el conjunto completo de reglas de correspondencia.

Las reglas de fijación de ámbito de nombre exigen que los declaradores de elementos en una unión particular sean únicos. Si el `<switch_type_spec>` es un `<enum_type>`, el identificador para la enumeración está en el ámbito de la unión; como resultado de ello, debe ser distinto de los declaradores de elementos.

No es necesario que todos los posibles valores del discriminador de unión estén enumerados en `<switch_body>`. El valor de una unión es el valor del discriminador junto con uno de los valores siguientes:

- si el valor del discriminador está enumerado explícitamente en una declaración de `case`, el valor del elemento asociado con esa declaración de `case`;
- si se especificó una etiqueta de `case` por defecto, el valor del elemento asociado con la etiqueta de `case` por defecto;
- ningún valor adicional.

El acceso al discriminador y los elementos conexos depende de la correspondencia del lenguaje.

Cuadro 10 –Correspondencia de etiquetas de caso

Tipo de discriminador	En correspondencia con
long	Cualquier valor entero en la gama de valores de long
long long	Cualquier valor entero en la gama de valores de long long
short	Cualquier valor entero en la gama de valores de short
unsigned long	Cualquier valor entero en la gama de valores de unsigned long
unsigned long long	Cualquier valor entero en la gama de valores de unsigned long long
unsigned short	Cualquier valor entero en la gama de valores de unsigned short
char	char
wchar	wchar
boolean	TRUE o FALSE
enum	Cualquier enumerador para el discriminador enumtype

4.7.2.3 Enumeraciones

Los tipos enumerados consisten en listas ordenadas de identificadores. La sintaxis es:

```
<enum_type>          ::=  "enum"<identifier>"{" <enumerator> { ","<enumerator> }*"}"

<enumerator>         ::=  <identifier>
```

En una enumeración se pueden especificar como máximo 2^{32} identificadores; como tales, los nombres enumerados deben coincidir con un tipo de datos autóctonos capaz de representar una enumeración de tamaño máximo. El orden según el cual se nombran los identificadores en la especificación de una enumeración define el orden relativo de los identificadores. Cualquier correspondencia del lenguaje que permita comparar dos enumeradores o defina funciones de sucesor/predecesor en enumeradores debe estar en consonancia con esta relación de orden. El `<identifier>` que sigue a la palabra clave `enum` define un nuevo tipo legal. Los tipos enumerados también se pueden nombrar utilizando una declaración `typedef`.

4.7.3 Tipos de plantilla

Los tipos de plantilla son:

```
<template_type_spec>    ::=  <sequence_type>

                         |   <string_type>

                         |   <wide_string_type>

                         |   <fixed_pt_type>
```

4.7.3.1 Secuencias

El ODP IDL define el tipo `sequence`. Una secuencia es una disposición unidimensional con dos características: un tamaño máximo (que se fija en el tiempo de compilación) y una longitud máxima (que se determina en el tiempo de ejecución).

La sintaxis es:

```
<sequence_type>         ::=  "sequence" "<" <simple_type_spec>
                                    ","<positive_int_const> ">"

                         |   "sequence" "<" <simple_type_spec> ">"
```

El segundo parámetro en una declaración de secuencia indica el tamaño máximo de la secuencia. Si se especifica una constante entera positiva para el tamaño máximo, la secuencia se termina en una secuencia vinculada. Antes de pasar una secuencia vinculada como un argumento de función (o como un campo en una estructura o unión), se debe fijar la longitud de la secuencia de una manera dependiente de la correspondencia del lenguaje. Tras recibir el resultado de una secuencia a partir de una invocación de operación, la longitud de la secuencia devuelta habrá sido fijada; este valor puede obtenerse de una manera dependiente de la correspondencia del lenguaje.

Si no se especifica tamaño máximo, el tamaño de la secuencia es no especificado (no vinculado). Antes de pasar de esa secuencia como un argumento de función (o como un campo en una estructura o unión), se debe fijar la longitud de la secuencia, el tamaño máximo de la secuencia y la dirección de una memoria tampón para retener la secuencia, de una manera dependiente de la correspondencia del lenguaje. Tras recibir el resultado de esa secuencia a partir de una invocación de operación, la longitud de la secuencia devuelta habrá sido fijada; este valor puede obtenerse de una manera dependiente de la correspondencia del lenguaje.

Un tipo de secuencia puede utilizarse como el parámetro tipo para otro tipo de secuencia. Por ejemplo, lo siguiente:

```
typedef sequence <sequence <long> > Fred;
```

declara que Fred es del tipo "unbounded sequence of unbounded sequence of long". Obsérvese que en el caso de las declaraciones de secuencia jerarquizadas, se debe utilizar espacio en blanco para separar los dos testigos ">" que terminan la declaración, para que estos no sean analizados como un único testigo ">>".

4.7.3.2 Cadenas

El ODP IDL define el tipo de cadena `string` como una cadena formada de todas las cantidades posibles de 8 bits salvo el nulo. Una cadena es similar a una secuencia de char. Tal como ocurre con las secuencias de cualquier tipo, antes de pasar una cadena como un argumento de función (o como un campo en una estructura o unión) se debe fijar la longitud de la cadena de una manera dependiente de la correspondencia del lenguaje. La sintaxis es:

```
<string_type>           ::=  "string" "<" <positive_int_const> ">"

                         |   "string"
```

El argumento para la declaración de cadena es el tamaño máximo de la cadena. Si se especifica un tamaño máximo entero positivo, la cadena se denomina cadena vinculada; si no se especifica tamaño máximo, la cadena se denomina cadena no vinculada.

Las cadenas se distinguen como un tipo separado porque numerosos lenguajes tienen funciones especiales incorporadas o funciones de biblioteca normalizadas para la manipulación de cadenas. Un tipo de cadena individual puede permitir optimizar considerablemente el tratamiento de las cadenas, en comparación con lo que se puede hacer con secuencias de tipo general.

4.7.3.3 Cadenas Wide char

El tipo de datos `wstring` representa una secuencia de `wchar` terminada en nulo (nota: un nulo de carácter amplio). El tipo `wstring` es análogo a `string`, salvo porque su tipo de elemento es `wchar` en vez de `char`.

```
<wide_string_type>      ::=  "wstring" "<" <positive_int_const> ">"
                        |    "wstring"
```

4.7.4 Declarador complejo

4.7.4.1 Matrices

El ODP IDL define matrices multidimensionales de tamaño fijo. Una matriz contiene tamaños explícitos para cada dimensión.

La sintaxis para la matriz es:

```
<array_declarator>      ::= <identifier> <fixed_array_size>+
<fixed_array_size>      ::= "[" <positive_int_const> "]"
```

El tamaño de matriz (en cada dimensión) es fijo en el tiempo de compilación. Cuando una matriz se pasa como un parámetro en una invocación de operación, se transmiten todos los elementos de la matriz.

La implementación de índices de matriz es específica a la correspondencia del lenguaje; la transmisión de un índice de matriz como un parámetro puede generar resultados incorrectos.

4.8 Typecodes y Principals

Typecodes y Principals son tipos particulares que pueden usarse para tener tipos como valores, en particular como parámetros en operaciones. Por ello, deben definirse mediante una vinculación de lenguaje para una determinada implementación. En el anexo B se describe una codificación particular para Typecodes en la especificación CORBA.

4.9 Declaración de excepción

Las declaraciones de excepción permiten declarar estructuras de datos a modo estructura que pueden devolverse para indicar que durante la realización de una petición ha ocurrido una condición excepcional. La sintaxis es la siguiente:

```
<except_dcl>            ::="exception" <identifier> "{" <member>* "}"
```

Cada excepción está caracterizada por su identificador ODP IDL, un identificador de tipo de excepción, y el tipo del valor de retorno asociado (especificado por el `<member>` en su declaración). Si una excepción se devuelve como resultado de una petición, el valor del identificador de excepción es accesible para el programador con miras a determinar qué excepción particular se planteó.

NOTA – Aunque esta norma es neutra desde el punto de vista de la arquitectura, se reservan ciertas excepciones que figuran en el anexo A.

Si se declara que una excepción tiene miembros, el programador podrá acceder a los valores de esos miembros cuando se plantea una excepción. Si no se especifican miembros, no se tendrá acceso a información adicional cuando se plantea una excepción.

4.10 Declaración de operación

Las declaraciones de operación en el ODP IDL son similares a las declaraciones de función C. La sintaxis es la siguiente:

```
<op_dcl>                ::= [<op_attribute> ] <op_type_spec>
                                <identifier><parameter_dcls>

                            [<raises_expr>] [ <context_expr>]


<op_type_spec>          ::= <param_type_spec>

                            | "void"
```

Una declaración de operación consiste en:

- Un atributo de operación facultativo que especifica cuál es la semántica de invocación que debería proporcionar el sistema de comunicaciones cuando se invoca la operación. Los atributos de operación se describen en 4.10.1.
- El tipo de resultado de retorno de la operación – Este tipo puede ser cualquier tipo que pueda definirse en ODP IDL. Las operaciones que no devuelven un resultado deben especificar el tipo vacante.
- Un identificador que nombra la operación en el ámbito de aplicación de la interfaz en la cual está definido.
- Una lista de parámetros que especifica cero o más declaraciones de parámetro para la operación. Las declaraciones de parámetro se describen en 4.10.2.
- Una expresión "provocadora" (raises expression) facultativa que indica las excepciones que pueden plantearse como resultado de una invocación de esta operación. Las expresiones "provocadoras" se describen en 4.10.3.
- Una expresión de contexto facultativa que indica cuáles son los elementos del contexto de petición que pueden ser consultados mediante el método que implementa la operación. Las expresiones de contexto se describen en 4.10.4.

4.10.1 Atributo de operación

El atributo de operación especifica cuál es la semántica de invocación que debe proporcionar el servicio de comunicación para las invocaciones de una operación particular. Un atributo de operación es facultativo. La sintaxis para esta especificación es la siguiente:

```
<op_attribute>          ::= "oneway"
```

Cuando un cliente invoca una operación con el atributo `oneway`, la semántica de la invocación es "mejor esfuerzo" (best-effort), lo que no garantiza la entrega de la llamada; "best-effort" implica que la operación se invocará como máximo una vez. Una operación con el atributo `oneway` puede no contener ningún parámetro de salida y debe especificar un tipo de retorno `void`. Una operación definida con el atributo `oneway` puede no incluir una expresión "provocadora"; no obstante, la invocación de esa operación puede plantear una excepción de norma.

Si no se especifica `<op_attribute>`, la semántica de la invocación es una vez como máximo (at-most-once) si se plantea una excepción; la semántica es exactamente una vez (exactly-once) si la invocación de la operación retorna con éxito.

4.10.2 Declaraciones de parámetro

Las declaraciones de parámetro en las declaraciones de operación ODP IDL tienen la siguiente sintaxis:

```
<parameter_dcls>        ::=  "(" <param_dcl> { "," <param_dcl> }* ")"

                        |    "(" ")"

<param_dcl>             ::=  <param_attribute>
                                  <param_type_spec><simple_declarator>

<param_attribute>       ::=  "in"

                        |    "out"

                        |    "inout"
```

```
<param_type_spec>       ::=  <base_type_spec>
                        |    <string_type>
                        |    <wide_string_type>
                        |    <wide_string_type>
                        |    <scoped_name>
```

Una declaración de parámetro debe tener un atributo direccional que informa al cliente y al servidor del servicio de comunicación el sentido en el cual se ha de transmitir el parámetro. Los atributos direccionales son:

- `in` – El parámetro se transmite del cliente al servidor.
- `out` – El parámetro se transmite del servidor al cliente.
- `inout` – El parámetro se transmite en ambos sentidos.

Se prevé que una implementación no tratará de modificar un parámetro `in`. La capacidad incluso para intentar esa modificación es específica a la correspondencia de lenguaje; el efecto de esa acción es no definido.

Si se plantea una excepción como resultado de una invocación, los valores del resultado de retorno y de cualquier parámetro `outand inout` son no definidos.

Cuando se transmite una `string`, `wstring` o `sequence` no vinculada como un parámetro `inout`, el valor devuelto no puede ser más largo que el valor de entrada.

4.10.3 Expresiones raises

Una expresión `raises` especifica cuáles son las excepciones que pueden plantearse como resultado de una invocación de la operación. La sintaxis para su especificación es la siguiente:

```
<raises_expr>           ::=  "raises" "(" <scoped_name> { "," <scoped_name> }* ")"
```

Los `<scoped_name>` en la expresión `raises` deben ser excepciones previamente definidas.

Además de cualquier excepción específica a la operación especificada en la expresión `raises`, hay un conjunto de excepciones normalizadas que se pueden señalar.

La ausencia de una expresión `raises` en una operación significa que no hay excepciones específicas a la operación. Las invocaciones de esa operación siguen siendo susceptibles de recibir una de las excepciones normalizadas.

4.10.4 Expresiones de contexto

Una expresión `context` especifica cuáles son los elementos del contexto del cliente que podrían afectar el comportamiento de una petición por el objeto. La sintaxis para su especificación es la siguiente:

```
<context_expr>          ::=  "context" "(" <string_literal>
                                    {"," <string_literal> }* ")"
```

El sistema tiempo de ejecución (runtime) garantiza que el valor (en su caso) asociado con cada `<string_literal>` en el contexto del cliente esté a disponibilidad de la implementación de objeto cuando se entrega la petición. La infraestructura de distribución y/u objeto tiene libertad para utilizar información en este *contexto de petición* durante la resolución y el desempeño de la petición.

La ausencia de expresión de contexto indica que no hay contexto de petición asociado a peticiones para esta operación.

Cada `string_literal` es una secuencia arbitrariamente larga de caracteres alfabético, dígito, punto ("."), subrayado ("_") y asterisco ("*"). El primer carácter de la cadena debe ser un carácter alfabético. Un asterisco sólo puede utilizarse como último carácter de la cadena. Algunas implementaciones pueden utilizar el carácter punto para dividir el espacio de nombre.

4.11 Declaración de atributo

Una interfaz puede tener atributos, así como operaciones; como tales, los atributos se definen como parte de una interfaz. Una definición de atributo es lógicamente equivalente a declarar un par de funciones de acceso; una para recuperar el valor del atributo y otra para fijar el valor del atributo.

La sintaxis para la declaración `attribute` es la siguiente:

```
<attr_dcl>              ::= ["readonly"]"attribute"<param_type_spec>
                                <simple_declarator>

                            {","<simple_declarator>}*
```

La palabra clave facultativa `readonly` indica que hay una sola función de acceso – la función de recuperación del valor. Considérese el siguiente ejemplo:

```
interface foo {
      enum material_t {rubber, glass};
      struct position_t {
      float x, y;
      };

      attribute float radius;
      attribute material_t material;
      readonly attribute position_t position;
      ...
};
```

Las declaraciones de atributo son equivalentes al siguiente fragmento de seudoespecificación:

```
float           _get_radius  ();
void            _set_radius  (in float r);
material_t      _get_material();
void   _        set_material (in material_t m);
position_t      _get_position ();
```

El nombre de atributo está sujeto a las reglas de fijación del ámbito de aplicación del nombre del ODP IDL; se garantiza que los nombres de acceso o función no colisionen con cualquier nombre de operación legal especificable en ODP IDL.

Las operaciones de atributo sólo devuelven errores por medio de excepciones normalizadas.

Los atributos se heredan. Un nombre de atributo no se puede volver a definir para transformarlo en un tipo diferente. Para mayor información sobre las limitaciones de redefinición y el tratamiento de ambigüedades, véase 4.12.

4.12 Módulo CORBA

El ámbito de denominación CORBA (denominado módulo CORBA) está reservado en el ODP IDL. Las definiciones en el módulo CORBA deben utilizarse únicamente cuando se escriben definiciones de interfaz específicas de dominio, y no deben utilizarse en descripciones de interfaz de cálculo ODP generales.

4.13 Nombres y fijación del ámbito de aplicación

Un fichero ODP IDL completo forma un ámbito de denominación. Además, los siguientes tipos de definiciones forman ámbitos jerarquizados:

- módulo;
- interfaz;
- estructura;
- unión;
- operación;
- excepción.

Los identificadores de los siguientes tipos de definiciones tienen fijado el ámbito de aplicación:

- tipos;
- constantes;
- valores de enumeración;
- excepciones;
- interfaces;
- atributos;
- operaciones.

Un identificador sólo puede ser definido una vez en un ámbito. No obstante, los identificadores se pueden redefinir en ámbitos jerarquizados.

Debido a las posibles restricciones que impondrán las futuras vinculaciones de lenguaje, los identificadores ODP IDL no son sensibles al tamaño de las letras (mayúscula/minúscula), y por lo tanto cuando dos identificadores difieren únicamente en el tamaño de sus caracteres son considerados redefiniciones de otro. Sin embargo, en todas las referencias a una definición se debe utilizar el mismo tamaño de carácter que en la ocurrencia definidora (esto permite la correspondencia natural con lenguajes sensibles al tamaño de los caracteres).

Los nombres de tipo definidos en un ámbito están disponibles para su utilización inmediata dentro de ese ámbito. Véase en particular 4.7.2 sobre los ciclos en las definiciones de tipo.

Un nombre se puede utilizar de una forma no calificada dentro de un ámbito particular; éste se resolverá buscando sucesivamente cada vez más lejos en los ámbitos abarcadores. Una vez que se utiliza en un ámbito un nombre no calificado, éste no se puede redefinir; es decir que si en el ámbito vigente se ha utilizado un nombre definido en un ámbito abarcador, no se puede redefinir una versión del nombre en el ámbito vigente. Esas redefiniciones generan un error de compilación.

Un nombre calificado (uno de la forma <scoped-name>::<identifier>) se resuelve resolviendo primero el calificador <scoped-name> para un ámbito, y luego situando la definición del <identifier> dentro de S. El identificador debe estar directamente definido en S o (si S es una interfaz) heredado en S. El <identifier> no es objeto de búsqueda en los ámbitos abarcadores.

Cuando un nombre calificado comienza con "::", el proceso de resolución comienza con el módulo abarcador más pequeño, y ubica los identificadores subsiguientes en el nombre calificado mediante la regla descrita en el párrafo anterior.

Cada definición ODP IDL en un fichero tiene un nombre global dentro de ese fichero. El nombre global para una definición se construye de la siguiente manera.

Antes de empezar a explorar un fichero que contiene una especificación ODP IDL, el nombre de la raíz vigente está inicialmente vacío ("") y el nombre del ámbito vigente también está inicialmente vacío (""). Siempre que se encuentra una palabra clave `module`, la cadena "::" y el identificador asociado se anexan al nombre de la raíz vigente; tras detectar la terminación del `module`, la etiqueta de cola "::" y el identificador se borran del nombre de la raíz vigente. Siempre que se encuentra una palabra clave `interface, struct, union` o `exception`, la cadena "::" y el identificador asociado se anexan al nombre del ámbito vigente; tras detectar la terminación de `interface, struct, union` o `exception`, la etiqueta de cola "::" y el identificador se borran del nombre del ámbito vigente. Además, cuando se procesan los parámetros de una declaración de operación se incorpora un nuevo ámbito no designado; esto permite a los nombres de parámetro duplicar otros identificadores; cuando se termina el procesamiento de un parámetro, el ámbito no designado se abandona.

El nombre global de una definición ODP IDL es la concatenación de la raíz vigente, el ámbito vigente, "::", y el <identifier>, que es el nombre local para esa definición.

La herencia produce versiones sombra de los identificadores heredados, es decir que introduce nombres en la interfaz derivada, pero se considera que esos nombres son semánticamente "los mismos" que en la definición original. Las versiones sombra del mismo original (resultante de la forma de diamante que se ilustra en la figura 1) introducen un solo nombre en la interfaz derivada y no están en conflicto entre sí.

La herencia introduce múltiples nombres ODP IDL globales para los identificadores heredados. Considérese el siguiente ejemplo:

```
interface A{
      exception E {
      long L;
      };
      void f() raises (E);
};

interface B:A {
      void g() raises(E);
};
```

En este ejemplo, la excepción se conoce por los nombres globales `::A::E` y `::B::E`.

Pueden surgir ambigüedades en las especificaciones a causa de los ámbitos de designación jerarquizados. Por ejemplo:

```
interface A{
      typedef string <128> string_t;
};

interface B{
      typedef string <256> string_t;
};

interface C:A,B {
      attribute string_t Title; /* AMBIGUOUS */
};
```

La declaración de atributo en C es ambigua, puesto que el compilador no sabe cuál `string_t` se desea. Las declaraciones ambiguas generan errores de compilación.

4.14 Diferencias con respecto a C++

Si bien la gramática ODP IDL trata de ajustarse a la sintaxis C++, es un poco más restrictiva. Las restricciones actuales son las siguientes:

- Es obligatorio un tipo de retorno de función.
- Se debe suministrar un nombre con cada parámetro formal para una declaración de operación.
- No se permite utilizar una lista de parámetros consistente en el testigo único `void` como sinónimo para una lista de parámetros vacíos.
- Se requieren rótulos para estructuras, uniones discriminadas y enumeraciones.
- Los tipos enteros no pueden definirse sencillamente como enteros o sin signo; deben ser declarados explícitamente `short` o `long`.
- `char` no puede calificarse con las palabras clave `signed` o `unsigned`.

Anexo A

Excepciones normalizadas reservadas

(Este anexo es parte integrante de esta Recomendación | Norma Internacional)

Este anexo presenta un conjunto normalizado de excepciones para una infraestructura ODP. Estos identificadores de excepción pueden retornarse como resultado de cualquier invocación de operación, independientemente de la especificación de interfaz. Las excepciones normalizadas pueden no estar enumeradas en expresiones `raises`.

Para limitar la complejidad en el tratamiento de las excepciones normalizadas, el conjunto de excepciones normalizadas se debería mantener a un volumen manejable. Esta limitación fuerza la definición de clases de equivalencia de excepciones en vez de enumerar muchas excepciones similares. Por ejemplo, una invocación de operación puede fallar en muchos puntos diferentes debido a la incapacidad de atribuir memoria dinámica. En vez de enumerar varias excepciones diferentes que corresponden a las diferentes formas según las cuales el fallo de atribución de memoria provoca la excepción (durante el ordenamiento, desordenamiento, en el cliente, en la implementación de objeto, al asignar paquetes de red, etc.), se define una sola excepción correspondiente a la falla de atribución de memoria dinámica. Cada excepción normalizada contiene un código menor para designar la subcategoría de la excepción; la asignación de valores a los códigos menores es una tarea que incumbe a cada implementación ODP.

Cada excepción normalizada incluye asimismo un código completion_status que adopta uno de los valores {COMPLETED_YES, COMPLETED_NO, COMPLETED_MAYBE}. Estos valores tienen los siguientes significados:

COMPLETED_YES	La implementación de objeto ha terminado el procesamiento antes de que se plantee la excepción.
COMPLETED_NO	La implementación de objeto nunca se inició antes de que se plantee la excepción.
COMPLETED_MAYBE	El estado de terminación de la implementación es indeterminado.

Las excepciones normalizadas se definen a continuación. Los objetos de cliente deben estar preparados para manejar excepciones de sistema que no se encuentran en esta lista, debido a que futuras versiones de esta especificación pueden definir excepciones normalizadas adicionales y además implementaciones de infraestructura ODP pueden dar lugar a excepciones de sistema no normalizadas.

```
#define ex_body {unsigned long minor; completion_status completed}


enum completion_status {COMPLETED_YES, COMPLETED_NO, COMPLETED_MAYBE};
enum exception_type {NO_EXCEPTION, USER_EXCEPTION, SYSTEM_EXCEPTION};


exception UNKNOWN                   ex_body;    //the unknown exception
exception BAD_PARAM                 ex_body;    //an invalid parameter was passed
exception NO_MEMORY                 ex_body;    //dynamic memory allocation failure
exception IMP_LIMIT                 ex_body;    //violated implementation limit
exception COMM_FAILURE              ex_body;    //communication failure
exception INV_OBJREF                ex_body;    //invalid object reference
exception NO_PERMISSION             ex_body;    //no permission for attempted op.
exception INTERNAL                  ex_body;    //ORB internal error
exception MARSHAL                   ex_body;    //error marshalling param/result
exception INITIALIZE                ex_body;    //ORB initialization failure
exception NO_IMPLEMENT              ex_body;    //operation implementation unavailable
exception BAD_TYPECODE              ex_body;    //bad typecode
exception BAD_OPERATION             ex_body;    //invalid operation
exception NO_RESOURCES              ex_body;    //insufficient resources for req.
exception NO_RESPONSE               ex_body;    //response to req. not yet available
```

```
exception PERSIST_STORE             ex_body;    //persistent storage failure
exception BAD_INV_ORDER             ex_body;    //routine invocations out of order
exception TRANSIENT                 ex_body;    //transient failure – reissue request
exception FREE_MEM                  ex_body;    //cannot free memory
exception INV_IDENT                 ex_body;    //invalid identifier syntax
exception INV_FLAG                  ex_body;    //invalid flag was specified
exception INTF_REPOS                ex_body;    //error accessing interface repository
exception BAD_CONTEXT               ex_body;    //error processing context object
exception OBJ_ADAPTER               ex_body;    //failure detected by object adapter
exception DATA_CONVERSION           ex_body;    //data conversion error
exception OBJECT_NOT_EXIST          ex_body;    //non-existent object, delete reference
exception TRANSACTION_REQUIRED      ex_body;
exception TRANSACTION_ROLLEDBACK    ex_body;
exception INVALID_TRANSACTION       ex_body;
```

A.1 No existencia de objeto

La excepción OBJECT_NOT_EXIST se produce cuando se realiza una invocación en un objeto suprimido. Es un informe de fallo "hard" autoritario. El que lo reciba puede (incluso debe) suprimir todas las copias de esta referencia de objeto y realizar otros procedimientos pertinentes del tipo "recuperación final".

A.2 Excepciones de transacción

La excepción TRANSACTION_REQUIRED indica que la petición transmitía un contexto de transacción nulo, pero que se necesita una transacción activa. La excepción TRANSACTION_ROLLEDBACK indica que la transacción asociada con la petición ya se ha restituido o que se ha marcado ya como restituida. Por lo tanto, la operación solicitada o no pudo realizarse o no se realizó porque el cálculo posterior en función de la transacción sería inútil.

INVALID_TRANSACTION indica que la petición transmitía un contexto de transacción no válido. Por ejemplo, esta excepción podría emitirse si apareciera un error cuando se intenta registrar un recurso.

Anexo B

Codificación Typecode en la especificación CORBA

(Este anexo no es parte integrante de esta Recomendación | Norma Internacional)

La interfaz ODP IDL para Typecodes en CORBA es la siguiente:

```
module CORBA {
   enum TCKind {
      tk_null, tk_void,
      tk_short, tk_long, tk_ushort, tk_ulong,
      tk_float, tk_double, tk_boolean, tk_char,
      tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
      tk_struct, tk_union, tk_enum, tk_string,
      tk_sequence, tk_array, tk_alias, tk_except,
      tk_longlong, tk_ulonglong, tk_longdouble,
      tk_wchar, tk_wstring, tk_fixed_pt
      };
   interface TypeCode {
      exception              Bounds{};
      exception              BadKind{};

      // for all TypeCode kinds
      boolean                equal (in TypeCode tc);
      TCkind                 kind ();

      // for tk_objref, tk_struct, tk_union, tkenum, tk_alias, and tk_except
      RepositoryId           id () raises (BadKind) ;

      // for tk_objref, tk_struct, tk_union, tkenum, tk_alias, and tk_except
      Identifier             name () raises (BadKind) ;

      // for tk_struct, tk_union, tk_enum, and tk_except
      unsigned long          member_count () raises (BadKind);
      Identifier             member_name (in unsigned long index)
                                  raises (BadKind, Bounds);

      // for tk_struct, tk_union, and tk_except
      TypeCode               member_type (in unsigned long index)
                                  raises (BadKinds, Bounds);

      //for tk_union
      any                    member_label (in unsigned long index)
                                  raises (BadKind, Bounds);
      TypeCode               discriminator_type () raises (BadKind);
      long                   default_index () raises (BadKind);
      //for tk_string, tk_sequence, and tk_array
      unsigned long length () raises (BadKind);
```

```
            //for tk_sequence, tk_array, and tk_alias
            TypeCode                content_type () raises (BadKind);


            // deprecated interface
            long                    parameter_count ();
            any                     parameter (in long index) raises (Bounds) ;
        };
```

Con las operaciones que anteceden se puede descomponer cualquier código de tipo en las partes que lo constituyen.

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 de programación