UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Z.130

(02/99)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

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

Técnicas de descripción formal

Lenguaje de definición de objetos de la UIT

Recomendación UIT-T Z.130

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES UIT-T DE LA SERIE Z

LENGUAJES Y ASPECTOS GENERALES DE SOPORTE LÓGICO PARA SISTEMAS DE TELECOMUNICACIÓN

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

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

RECOMENDACIÓN UIT-T Z.130

LENGUAJE DE DEFINICIÓN DE OBJETOS DE LA UIT

Resumen

Esta Recomendación especifica el lenguaje de definición de objetos de la UIT (ODL). El UIT-ODL se utiliza para la especificación de sistemas desde la perspectiva del punto de vista computacional del procesamiento distribuido abierto (ODP) [3]. Define plantillas para las interfaces operacionales, interfaces de trenes, objetos de múltiples interfaces y grupos de objetos.

UIT-ODL es una extensión del lenguaje de definición de interfaz de ODP (ODP-IDL [8]) con adiciones para soportar la especificación de los conceptos del punto de vista computacional de ODP en un nivel sintáctico. UIT-ODL es un superconjunto de ODP-IDL. Esta relación entre UIT-ODL y ODP-IDL soporta la construcción de sistemas mediante las implementaciones de intermediario de petición de objetos especificadas por el grupo de gestión de objetos (OMG) [1]. Cabe esperar que los lectores de esta Recomendación estén familiarizados con el ODP-IDL.

Orígenes

La Recomendación UIT-T Z.130 ha sido preparada por la Comisión de Estudio 10 (1997-2000) del UIT-T y fue aprobada por el procedimiento de la Resolución N.° 1 de la CMNT el 12 de febrero de 1999.

PREFACIO

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

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

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

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

NOTA

En esta Recomendación, la expresión *empresa de explotación reconocida (EER)* designa a toda persona, compañía, empresa u organización gubernamental que explote un servicio de correspondencia pública. Los términos *Administración, EER* y *correspondencia pública* están definidos en la *Constitución de la UIT (Ginebra, 1992).*

PROPIEDAD INTELECTUAL

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

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

ÍNDICE

Página

Recomendación Z.130

LENGUAJE DE DEFINICIÓN DE OBJETOS DE LA UIT

(Ginebra, 1999)

1 Alcance

El UIT-ODL ha sido elaborado para:

- especificaciones computacionales de documentos. Por ejemplo, un componente de servicio puede ser descrito desde el punto de vista computacional utilizando una especificación UIT-ODL;
- proporcionar la sintaxis adecuada para desarrollar aplicaciones de apoyo de ingeniería de soporte lógico, tales como analizadores UIT-ODL, generadores de códigos, editores de especificaciones computacionales y herramientas relacionadas con CASE.

El UIT-ODL es una extensión del lenguaje de definición de interfaz de ODP (ODP-IDL, *ODP interface definition language*) [8]. El UIT-ODL soporta características que no están cubiertas (actualmente) por el ODP-IDL. Estas características se derivan del lenguaje computacional del modelo de referencia para procesamiento distribuido abierto [3] e incluye plantillas de objetos de múltiples interfaces, plantillas de grupo, plantillas de interfaz de trenes y descripciones de calidad de servicio[1] (*QoS, quality of service*). En el apéndice II figura una comparación completa de UIT-ODL y ODP-IDL.

Cabe señalar que el UIT-ODL está muy influido por el trabajo efectuado en TINA Consortium de acuerdo con el lenguaje TINA-ODL [4]. Sin embargo, algunos conceptos no pudieron ser adoptados y hubo que cambiarlos. Especialmente el concepto de plantilla de grupo ha obtenido una nueva semántica.

El UIT-ODL proporciona una sintaxis para describir aspectos estáticos del punto de vista computacional de los sistemas del procesamiento distribuido abierto (ODP, *open distributed processing*). Esto significa que el lenguaje abarca las estructuras y firmas de estos sistemas pero no el comportamiento (de una manera formal). Entre los aspectos que pueden ser expresados utilizando la sintaxis UIT-ODL figuran:

- la descripción de plantillas de objetos computacionales que soportan plantillas de múltiples interfaces. Las diferentes plantillas de interfaces soportadas representan servicios lógicamente distintos proporcionados por el mismo objeto;
- la capacidad de un objeto de tratar múltiples casos de la misma plantilla de interfaz. Esto permite que el objeto mantenga contextos específicos de cliente;
- la capacidad de controlar el acceso y la visibilidad de partes de una funcionalidad de objeto;
- la posibilidad de describir los servicios/funcionalidad que un objeto necesita de su entorno. Esta característica permite verificar la compatibilidad de plantillas de interfaz en un nivel de especificación estático;
- la capacidad de estructurar especificaciones utilizando plantillas de grupo de objetos. Las plantillas de objetos que comparten una propiedad común pueden ser agrupadas en una

[1] La notación de calidad de servicio está contenida en el apéndice I.

plantilla de grupo de objetos. Como ejemplos cabe citar los aspectos de realización así como los aspectos de gestión;

- la posibilidad de describir aspectos estáticos de plantillas de interfaz de trenes;
- la capacidad de asociar atributos de calidad de servicio con operaciones y flujos (este concepto se describe en el apéndice I).

El UIT-ODL sirve de base para describir componentes de soporte lógico reutilizables.

2 Referencias

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

[1] OMG Adopted Specification (1998), *The Common Object Request Broker: Architecture and Specification, Revision 2.2*.

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

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

[4] TINA-C Specification (1996), *Object Definition Language Manual, Version 2.3*.

[5] Recomendación UIT-T Z.100 (1993), *Lenguaje de especificación y descripción del CCITT*.

[6] Recomendación UIT-T Z.100 (1993)/add.1 (1996), *Lenguaje de Especificación y Descripción del CCITT – Addendum 1*.

[7] Recomendación UIT-T Z.105 (1995), *Lenguaje de especificación y descripción combinado con la notación de sintaxis abstracta uno*.

[8] Recomendación UIT-T X.920 (1997) | ISO/CEI 14750:1999, *Tecnología de la información – Procesamiento distribuido abierto – Lenguaje de definición de interfaz.*

[9] http://www.fokus.gmd.de/research/cc/platin/products/y-sce/ – An acceptor for ITU-ODL.

[10] ISO/CEI 14882:1998, *Programming languages – C++*.

3 Abreviaturas

En esta Recomendación se utilizan las siguientes siglas.

CASE Ingeniería de soporte lógico asistida por computador (*computer-aided software engineering*)

CORBA Arquitectura de intermediario de petición de objeto común (*common object request broker architecture*)

DPE Entorno de procesamiento distribuido (*distributed processing environment*)

IDL Lenguaje de definición de interfaz (*interface definition language*)

ODP Procesamiento distribuido abierto (*open distributed processing*)

ORB Intermediario de petición de objeto (*object request broker*)

4 Definiciones

En la presente Recomendación se utilizan los términos siguientes definidos en las Recomendaciones [2], [3] y [8].

X.902	X.903	X.920
compatibilidad en comportamiento	anuncio	excepción
objeto cliente	interfaz computacional	fichero
entorno de un objeto	objeto computacional	módulo
herencia	flujo	herencia múltiple
instanciación de una plantilla	grupo (de interacción)	
objeto	interrogación	
calidad de servicio	operación	
objeto servidor	firma de interfaz de operaciones	
servicio	firma de interfaz de trenes	
plantilla	terminación	

NOTA – Hay una discordancia en la definición del término objeto en ODP y en el grupo de gestión de objetos (OMG, *object management group*). Como el lenguaje UIT-ODL incluye el ODP-IDL como un subconjunto, el término objeto puede ser utilizado también para referirse a la definición del OMG. Cuando es así, se señala especialmente.

5 Fundamentos y reglas

Esta cláusula presenta los principios del UIT-ODL, su metaestructura y semántica. En esta cláusula se presenta también la sintaxis UIT-ODL con fines ilustrativos. El significado de esta sintaxis se explica a medida que se presenta, pero se recuerda al lector que en la cláusula siguiente figura una presentación detallada de la sintaxis UIT-ODL.

Inicialmente se presentan definiciones y convenios básicos, seguidos por las reglas para la denominación básica y la fijación de ámbito. Se añaden otras reglas de denominación y fijación de ámbito junto con cada adición arquitectural. Una característica del UIT-ODL es su capacidad de compartir y reutilizar especificaciones existentes. A continuación se presentan los principios según los cuales se especifica el comportamiento. Como se indica anteriormente, una forma de reutilización de especificaciones es mediante la composición, mientras que una segunda forma es mediante la especialización o herencia. En 5.5 se presenta la arquitectura de herencia específica adoptada en el UIT-ODL.

En esta cláusula, las reglas son destacadas mediante etiquetas de la forma Rn (por ejemplo, R1, R2, etc.). Las definiciones de términos se indican con etiquetas de la forma Dn (por ejemplo, D1, D2, etc.).

5.1 Definiciones y convenios

5.1.1 Definiciones

En esta Recomendación se definen los términos siguientes.

5.1.1.1 (D1) construcción: Una construcción del UIT-ODL es una plantilla de grupo de objetos, una plantilla de objeto, una plantilla de interfaz, operación o flujo. La especificación de firma de una construcción del UIT-ODL es declarada en el UIT-ODL.

5.1.1.2 (D2) definición soportadora (sustentadora): Las definiciones de tipos de datos, constantes y declaraciones de excepción se denominan definiciones sustentadoras.

En las siguientes subcláusulas se presentan definiciones adicionales, a medida que aparecen conceptos adicionales.

5.1.2 Convenios gráficos

Los siguientes convenios se aplican a la representación gráfica de los ejemplos dados en el resto de esta Recomendación.

- Las plantillas de objeto se representan como cajas (rectángulos) (véase la figura 1).

Figura 1/Z.130 – Plantillas de objeto

- Una plantilla de interfaz operacional soportada se representa como un rectángulo rellenado contiguo a la caja que representa la plantilla de objeto a la cual pertenece (véase la figura 2). Algunas plantillas de interfaz pudieran ser destacadas mediante el uso de un patrón especial. Una operación se representa como una flecha que apunta a la plantilla de interfaz operacional a la cual pertenece. Otros elementos de plantillas de interfaz operacional no se representan.

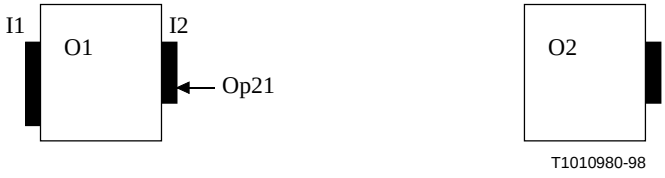

Figura 2/Z.130 – Plantillas de objeto con interfaces soportadas

- Una plantilla de interfaz de trenes soportada se representa como un rectángulo que contiene círculos abiertos y/o rellenados, contiguos a la caja que representa la plantilla de objeto a la cual pertenece (véase la figura 3). Algunas plantillas de interfaz pudieran ser destacadas mediante el uso de un patrón especial.
- Un sumidero de flujo se representa como una flecha que apunta a un círculo rellenado en la plantilla de interfaz de trenes a la cual pertenece. Una fuente de flujo se representa como una flecha que apunta fuera de un círculo abierto en la plantilla de interfaz de trenes a la cual pertenece. Otros elementos de plantillas de interfaz de trenes no se representan.

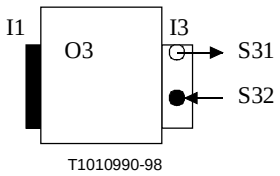

Figura 3/Z.130 – Plantilla de objeto con fuente y sumidero

- Las plantillas de grupo de objeto de representan mediante cajas con líneas de trazo interrumpido. El contenido en una plantilla de grupo de objetos se representa mediante el contenido en una caja de líneas de trazo interrumpido (véase la figura 4).

Figura 4/Z.130 – Plantilla de grupo

Tras haber presentado los términos básicos y los convenios gráficos utilizados en esta Recomendación, a continuación se examinará el marco de denominación y fijación de ámbito del UIT-ODL.

5.2 Denominación y fijación de ámbito

Se definen reglas de denominación y fijación de ámbito para permitir la identificación inequívoca de construcciones del UIT-ODL. Para un análisis comparativo del trabajo conexo, véanse también las reglas comparables en el ODP-IDL [8].

(R1) Un fichero UIT-ODL entero forma un ámbito de denominación.

(R2) Las siguientes clases de definiciones forman ámbitos jerarquizados:

- módulo;
- plantilla de grupo de objetos;
- plantilla de objeto;
- plantilla de interfaz;
- estructura;

- unión;
- operación; y
- excepción.

Por ejemplo, las siguientes definiciones del UIT-ODL están contenidas en un fichero. Especifica un módulo, M1, que contiene una plantilla de grupo de objeto, G1, que contiene una plantilla de objeto, O1, que contiene una plantilla de interfaz, I1, que contiene un tipo de datos, DataType1, y una operación, operation1. M1 tiene un ámbito global. G1 está dentro de M1. O1 está dentro de G1. I1 está dentro de O1. DataType1 y operation1 están dentro de I1.

```
module M1 {
        ...
        group G1 {
                ...
                CO O1 {
                        ...
                        interface I1 {
                                ...
                                typedef … DataType1;
                                ...
                                void operation1(in DataType1 variable11…);
                                ...
                        }; // end of I1
                }; // end of O1
        }; // end of G1
}; // end of M1
```

(R3) Se indican identificadores para la siguiente clase de definiciones:

- plantillas de grupo de objeto;
- plantillas de objeto;
- plantillas de interfaz;
- operaciones;
- tipos de datos;
- constantes;
- valores de enumeración;
- excepciones; y
- atributos.

(R4) Un identificador sólo puede ser definido una vez en un ámbito. Los identificadores pueden ser redefinidos en ámbitos jerarquizados.

(R5) Los identificadores son insensibles al tipo de letra.

(R6) Los identificadores definidos en un ámbito están disponibles para uso inmediato dentro de ese ámbito.

(R7) Un nombre calificado (uno de la forma <scoped-name>::<identifier>) es resuelto localizando la definición de <identifier> dentro del ámbito. El identificador debe estar definido directamente en el ámbito. No se busca el identificador en los ámbitos abarcadores.

Por ejemplo, sobre la base del ejemplo UIT-ODL anterior, el nombre calificado de G1 es M1::G1. De manera similar, el nombre calificado de DataType1 es M1::G1::O1::I1::DataType1.

(R8) Un nombre no calificado (uno de la forma <identifier>) puede ser utilizado dentro de un ámbito particular. Se resolverá buscando sucesivamente más lejos en otros ámbitos. Una vez que se utiliza un nombre no calificado en un ámbito, no puede ser redefinido.

Por ejemplo, el texto UIT-ODL a continuación muestra DataType1 definido en el ámbito del módulo M1. Se utiliza (como un nombre no calificado) dentro del ámbito de la plantilla de interfaz I1. Además dentro del ámbito de I1, DataType1 se define de nuevo. Esta segunda definición de DataType1 es ilegal. Si la segunda definición se colocase antes del enunciado (statement) operation1, la redefinición sería legal, y esta segunda definición se utilizaría para resolver el nombre no calificado en el enunciado operation1.

```
module M1 {
        ...
        typedef … DataType1;
        interface I1 {
                ...
                void operation1(in DataType1 variable11…);
                ...
                typedef … DataType1; // Illegal statement
                ...
        }; // end of I1
}; // end of M1
```

(R9) Cada definición UIT-ODL en un fichero tiene un nombre global dentro de ese fichero. La regla para crear un nombre global es igual que en el ODP-IDL [8]:

"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 de `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."

Además, para este fin, la plantilla de grupo de objetos y la plantilla de objeto son tratadas similarmente a interface (plantilla), struct, union, y exception.

5.2.1 (D3) nombre rotulado: Un nombre rotulado es uno de la forma <scoped_name>"."<scoped_name>.

Esta construcción se utiliza para tener un acceso calificado a las plantillas de interfaz soportadas de una plantilla de objeto de una plantilla de grupo de objetos en una especificación de interfaz requerida.

Tras haber presentado algunas de las construcciones básicas del lenguaje UIT-ODL, a continuación se examinará cómo las especificaciones UIT-ODL pueden ser presentadas como Recomendaciones, en particular con miras a reutilizar las especificaciones del UIT-ODL (o del ODP-IDL).

5.3 Separación y compartición de plantilla de interfaz, plantilla de objeto y plantilla de grupo de objetos

El diseñador de especificaciones computacionales dispone de libertad en cuanto a la declaración independiente de plantillas de interfaz, de plantillas de objeto y de plantillas de grupos de objetos. Cada plantilla de interfaz en el UIT-ODL puede ser reutilizada en cualquier número de plantillas de objeto. De manera similar, las plantillas de objeto pueden ser especificadas como definiciones individuales, y reutilizadas en cualquier número de plantillas de grupo de objetos.

Como el UIT-ODL es un superconjunto del ODP-IDL, la correspondencia entre una plantilla de interfaz operacional del UIT-ODL y una especificación del ODP-IDL equivalente es trivial. La ventaja de esta correspondencia directa estriba en la capacidad de utilizar las herramientas existentes basadas en CORBA en la cadena de desarrollo de soporte lógico. Otra ventaja es la capacidad de reutilizar las definiciones de interfaz ODP-IDL existentes.

Como se indica anteriormente, una ventaja de separar declaraciones de construcciones es que esto proporciona un medio directo de compartir declaraciones de construcciones. A continuación se presentan más detalladamente los principios de compartición.

5.3.1 Tipos de datos

(R10) Los tipos de datos pueden ser declarados en cualquier ámbito UIT-ODL. Se permite la compartición de declaraciones de tipos de datos entre varias operaciones o flujos de diferentes plantillas de interfaz.

5.3.2 Operaciones

(R11) Las firmas de operaciones son declaradas dentro de plantillas de interfaz. Como las firmas de operaciones no son declaradas separadas de las plantillas de interfaz, y no hay un mecanismo de compartición adecuado específico, no es posible la compartición de una declaración de firma de operación entre varias plantillas de interfaz.

NOTA – Esta regla se deriva directamente del ODP-IDL. Una nueva versión de ODP-IDL puede mitigar esta regla. La repercusión de este cambio para el UIT-ODL es un asunto abierto.

(R12) Dos operaciones con el mismo identificador declarado en dos plantillas de interfaz distintas se consideran diferentes.

5.3.3 Flujos

(R13) Las firmas de flujos son declaradas dentro de plantillas de interfaz. Como las firmas de flujos no son declaradas separadas de las plantillas de interfaz, y no hay un mecanismo de compartición adecuado específico, no es posible la compartición de una declaración de firma de flujo entre varias plantillas de interfaz.

(R14) Dos flujos con el mismo identificador declarado en dos plantillas de interfaz distintas se consideran diferentes.

5.3.4 Plantillas de interfaz

Se supone que la compartición de declaración de plantilla de interfaz está destinada a la compartición de una especificación del UIT-ODL de una plantilla de interfaz entre varias plantillas de objeto. Se define la sintaxis del UIT-ODL para permitir la separación de declaraciones de plantilla de interfaz de las declaraciones de plantilla de objeto. Las especificaciones de plantilla de interfaz pueden ser incluidas en una declaración de plantilla de objeto como interfaces soportadas o como interfaces requeridas.

5.3.4.1 (D4) interfaces soportadas/interfaces ofrecidas declaradas: Las plantillas de interfaz indicadas como soportadas en una plantilla de objeto son las únicas plantillas de interfaz para las cuales pueden existir casos en los objetos[2]. Las interfaces ofrecidas de un objeto son las interfaces existentes en ese objeto en un momento determinado.

5.3.4.2 (D5) interfaces requeridas declaradas: Las interfaces requeridas declaradas en una plantilla de objeto enumeran las plantillas de interfaz que un caso de la plantilla de objeto necesita para invocar operaciones[3].

Las siguientes reglas tratan de la relación de declaraciones de operaciones, flujos, plantillas de interfaz y plantillas de objeto:

(R15) Las firmas de operaciones y flujos sólo son declaradas dentro de plantillas de interfaz. Las plantillas de interfaz pueden ser declaradas dentro y fuera de plantillas de objeto, y dentro y fuera de plantillas de grupo de objetos.

5.3.5 Plantillas de objeto

Se supone que la compartición de declaración de plantilla de objeto está destinada a la compartición de una especificación UIT-ODL de una plantilla de objeto entre varias plantillas de grupo. La sintaxis UIT-ODL está definida para permitir la separación de declaraciones de plantilla de objeto con respecto a las declaraciones de plantilla de grupo. Se puede hacer referencia a especificaciones de plantilla de objeto en una plantilla de grupo como miembros. Se puede hacer referencia a especificaciones de plantilla de interfaz en una plantilla de grupo de objetos como contratos soportados o requeridos, que son las plantillas de interfaz que pueden ser utilizadas por entidades externas al grupo de objetos (soportadas) o necesitadas por los miembros del grupo del entorno (requeridas).

5.3.5.1 (D6) miembros declarados: Los miembros declarados de una plantilla de grupo de objeto son las plantillas de objeto, o las plantillas de grupo de objetos, pertenecientes a esa plantilla de grupo. Los miembros declarados se enumeran como "miembros" en una plantilla de grupo.

5.3.5.2 (D7) contrato requerido/soportado declarado: Un contrato requerido/soportado declarado de una plantilla de grupo de objetos es una de las interfaces requeridas/soportadas de una plantilla de objeto de miembro o plantilla de grupo de esa plantilla de grupo de objetos. Los contratos requeridos/soportados declarados en una plantilla de grupo de objetos representan las únicas interfaces que pueden ser utilizadas (soportadas) por entidades externas a ese grupo de objetos o que los casos de los miembros pueden ser accedidos en el entorno (soportadas). Los contratos declarados se enumeran como requeridos o sustentados en una plantilla de grupo (similar a plantillas de objeto). Si no se especifican interfaces requeridas y soportadas en una plantilla de grupo, no se imponen restricciones relativas a la visibilidad de interfaces de los miembros del grupo[4].

La regla es la siguiente:

(R16) Las plantillas de objeto pueden ser declaradas fuera de plantillas de grupo de objetos.

---

[2] Las interfaces soportadas declaradas de una clase básica se consideran como interfaces soportadas de la subclase y pueden ser ejemplificadas también por el caso de objeto de la plantilla de subclase.

[3] Las interfaces requeridas declaradas de una clase básica se consideran como interfaces requeridas de la subclase.

[4] Esto se debe al hecho de que son posibles diferentes criterios para definir grupos y para algunos de ellos no es útil la especificación de contratos.

5.3.6 Reglas de fijación de ámbito

Las siguientes reglas de fijación de ámbito son pertinentes a especificaciones compartidas:

(R17) En el ámbito de una plantilla de objeto, un identificador de interfaz puede ser definido declarando la plantilla de interfaz asociada en línea, o puede ser utilizado declarándolo como soportado o requerido. En cada caso, el nombre global de la plantilla de interfaz será diferente; algo así como... <object-identifier>::<interface-identifier> en el primer caso y algo así como...<interface-identifier> en el segundo caso.

(R18) En el ámbito de una plantilla de grupo de objetos, un identificador de objeto puede ser definido declarando la plantilla de objeto asociada en línea, o puede ser utilizado declarándolo como un miembro. En cada caso, el nombre global de la plantilla de objeto será diferente.

(R19) En el ámbito de una plantilla de grupo de objetos, un identificador de interfaz puede ser definido declarando la plantilla de interfaz asociada en línea, o puede ser utilizado declarándolo como un contrato requerido/soportado. En cada caso, el nombre global de la plantilla de interfaz será diferente.

5.4 Comportamiento

El comportamiento de una entidad (interfaz, objeto, plantilla de grupo de objetos), en su sentido más general, consiste en todas las posibles interacciones que la entidad puede emprender dentro de su entorno. El UIT-ODL no está suficientemente maduro para proporcionar una especificación completa y detallada para el comportamiento en este sentido. En cambio, a continuación se describe una especificación de comportamiento informal para determinadas entidades:

Plantillas de interfaz

Esta especificación describe el servicio proporcionado por un caso de la plantilla que se define. Describe también la utilización prevista de la interfaz. Esta especificación documenta las restricciones de ordenación (o secuenciación) impuestas a las operaciones definidas en la plantilla de interfaz. Las invocaciones de operaciones en una de estas plantillas debe satisfacer estas restricciones. En la versión vigente del UIT-ODL esta especificación es un literal de cadena.

Plantillas de objeto

Esta especificación describe las responsabilidades de un objeto de prestar servicios mediante cada una de sus interfaces soportadas en el UIT-ODL.

Plantillas de grupo de objetos

Esta especificación describe el criterio que se mantiene para todas las entidades contenidas en la plantilla de grupo. Dependiendo de los criterios, la especificación de predicado de grupo puede contener información adicional. Por ejemplo, si el criterio es que los miembros del grupo prestan juntos un determinado servicio, la descripción del predicado puede definir además la funcionalidad proporcionada en cada contrato soportado.

NOTA – Se ha de señalar que la cadena proporcionada como una especificación del comportamiento puede ser una referencia a una especificación de comportamiento formal hecha en otro lenguaje. Un ejemplo podría ser un enlace de una definición de interfaz operacional a una especificación de tipo de proceso del SDL [7] que contiene una especificación de comportamiento formal para esa plantilla de interfaz. La especificación e interpretación de estos enlaces están sujetas a herramientas CASE que utilizan el UIT-ODL como una técnica de descripción para especificaciones del punto de vista computacional.

5.5 Herencia

5.5.1 Presentación y motivación

La plantilla de interfaz, las plantillas de objeto y las plantillas de grupo de objetos proveen modularidad de especificación. Como estas tres plantillas representan también tipos, es conveniente definir un mecanismo de reutilización donde:

- una construcción pueda depender directamente de las entidades definidas en otra construcción (de la misma clase de plantilla). Por ejemplo, un tipo de datos definido en una plantilla de interfaz se utiliza dentro de otra plantilla de interfaz;
- una construcción se derive de otra construcción (de la misma clase de plantilla). Por ejemplo, una especificación de plantilla de objeto se deriva de otra especificación de plantilla de objeto.

En la literatura basada en objetos clásicos, este mecanismo de reutilización se conoce como herencia. En el caso del UIT-ODL, la definición de reglas para la herencia permitirá que otras plantillas de interfaz, plantillas de objeto y plantillas de grupos de objetos sean declaradas como extensiones o restricciones de las definidas previamente.

En el resto de esta cláusula se describen los principios subyacentes de la reutilización por el UIT-ODL de especificaciones mediante herencia. En primer lugar se presentan los elementos esenciales de herencia de plantilla de interfaz, herencia de plantilla de objeto y herencia de plantilla de grupo de objetos. A continuación se explican las reglas de denominación y fijación de ámbito relacionadas con la herencia.

5.5.2 Definiciones

Las siguientes definiciones son pertinentes a la herencia.

5.5.2.1 (D8) construcción básica/derivada/especializada: Se dice que una construcción (plantilla de grupo, plantilla de objeto o plantilla de interfaz) se deriva de otra construcción o la especializa, denominada una construcción básica de la construcción derivada, si hereda de esta construcción básica.

5.5.2.2 (D9) más especializada: Las plantillas de objetos más especializadas (plantillas de grupo o plantillas de interfaz) dentro de un conjunto de plantillas de objeto (plantillas de grupo o plantillas de interfaz) son los elementos del conjunto del cual no se deriva ninguna otra construcción dentro del conjunto (es decir, que no son la base de cualquier otra construcción del conjunto).

5.5.2.3 (D10) base directa/indirecta: Una construcción se denomina una base directa de una construcción si es mencionada en la especificación de herencia de la declaración de construcción, y una base indirecta si no es una base directa sino la base de una base directa o indirecta (subherencia).

5.5.2.4 (D11) gráfico de herencia/gráfico de herencia parcial: El gráfico de herencia de plantillas de objeto (plantillas de grupo o plantillas de interfaz) es el gráfico acíclico dirigido que representa las relaciones de herencia entre plantillas de objeto (plantillas de grupo o plantillas de interfaz). Un gráfico de herencia parcial de un tipo dado es un gráfico de herencia restringido a un conjunto de construcciones.

NOTA – Las hojas del gráfico de herencia para un tipo dado de construcción (es decir para plantillas de grupo, plantillas de objeto o plantillas de interfaz) son las construcciones más especializadas de este tipo.

5.5.2.5 (D12) restricción/conjunto restringido: La restricción de un conjunto de construcciones se construye suprimiendo de este conjunto cualquier construcción que sea la base de cualquier otra construcción del conjunto. El resultado de la restricción de un conjunto se denomina conjunto restringido.

NOTA 1 – Un conjunto restringido puede ser considerado también como el conjunto de hojas del gráfico de herencia parcial.

NOTA 2 – El conjunto restringido de todas las construcciones de plantilla de objeto de tipo (plantilla de grupo o plantilla de interfaz) es el conjunto de plantillas de objeto más especializadas (plantillas de grupo o plantillas de interfaz).

Supóngase que se definen las siguientes plantillas de interfaz con la siguiente relación de herencia:

- plantilla de interfaz I1;
- plantilla de interfaz I2, que hereda de la plantilla de interfaz I1;
- plantilla de interfaz I3, que hereda de la plantilla de interfaz I1;
- plantilla de interfaz I4.

La restricción del conjunto de plantillas de interfaz (I1, I2, I3, I4) es el conjunto (I2, I3, I4).

5.5.3 Herencia en declaraciones de construcciones

5.5.3.1 Herencia de plantilla de interfaz

Se supone que la herencia de plantilla de interfaz proporciona la "reutilización por especialización" de (una especificación UIT-ODL de) una plantilla de interfaz. La plantilla de interfaz de la cual se hereda se denomina una plantilla de interfaz básica. La plantilla de interfaz que hereda se denomina la plantilla de interfaz derivada. Esta especialización puede adoptar dos formas:

- adición de nuevas operaciones, o flujos, a la lista de las plantillas de interfaz básicas;
- redefinición de las firmas de operaciones o flujos en las plantillas de interfaz básicas.

NOTA 1 – La versión actual del ODP-IDL no permite esta clase de herencia para plantillas de interfaz operacional (según la regla 24); y en consecuencia, tampoco lo permite el UIT-ODL.

NOTA 2 – Todas las plantillas de interfaz derivadas de una plantilla de interfaz básica se consideran "compatibles con" la plantilla de interfaz básica.

La sintaxis definida para el UIT-ODL apoya plenamente las reglas de herencia de (plantilla de) interfaz del ODP-IDL, y adopta reglas coherentes para ambas plantillas de interfaz operacional y de trenes. Las reglas de herencia de plantilla de interfaz del UIT-ODL son las siguientes:

Reglas generales

(R20) Una plantilla de interfaz puede ser derivada de una o varias otras plantillas de interfaz, cada una de las cuales se denomina una plantilla de interfaz básica de la plantilla de interfaz derivada. En el caso de derivación de múltiples interfaces básicas (herencia múltiple), el orden de derivación no es importante.

(R21) Una plantilla de interfaz no puede ser especificada como una plantilla de interfaz básica directa de una plantilla de interfaz derivada más de una vez. Puede ser una plantilla de interfaz básica indirecta más de una vez (es decir, es posible un gráfico de herencia de "forma de diamante").

(R22) Una plantilla de interfaz derivada puede declarar nuevas subconstrucciones (tipos de datos, operaciones o flujos). A menos que sean redefinidas, las subconstrucciones de la plantilla de interfaz básica pueden ser denominadas como si fuesen subconstrucciones de la plantilla de

interfaz derivada. La herencia de plantilla de interfaz hace que todos los identificadores en el cierre del gráfico de herencia sean importados en el ámbito de denominación vigente.

(R23) Es ilegal heredar de dos plantillas de interfaz que tienen los mismos identificadores de operación, o tienen los mismos identificadores de flujo.

(R24) Es ilegal redefinir una operación en la plantilla de interfaz derivada.

(R25) Es ilegal redefinir un flujo en la plantilla de interfaz derivada.

(R26) Una plantilla de interfaz derivada puede redefinir identificadores de tipos de datos heredados. Un identificador de tipo de datos de un ámbito abarcador puede ser redefinido en el ámbito vigente.

Comportamiento

(R27) El texto de comportamiento (behaviourText) de las plantillas de interfaz básicas no está disponible en una plantilla de interfaz derivada.

(R28) El atributo de utilización de las plantillas de interfaz básicas no está disponible en una plantilla de interfaz derivada.

Como un ejemplo ilustrativo, supóngase que la plantilla de objeto O4 declara como soportadas:

– la plantilla de interfaz I4, una especialización de la plantilla de interfaz I1 construida por la adición de operation41; y

– la plantilla de interfaz S2, una especialización de la plantilla de interfaz S1, con la adición de la fuente videoFlow21.

Es posible declarar que la plantilla de interfaz I4 hereda operation11 de I1, y añadir la operación operation41. De manera similar, S2 puede heredar de S1 el flujo fuente voiceDownStream y el flujo sumidero voiceUpStream, y añadir el flujo fuente videoFlow21.

I1 y S1 se definen como sigue:

```
interface I1{
        ...
        // data types
        typedef … DataType11;
        typedef … DataType12;

        void operation11 (in DataType11 …, out DataType12 …);
}; // end of I1

interface S1{
        ...
        // flow types
        typedef … VoiceFlowType;

        source VoiceFlowType voiceDownStream;
        sink VoiceFlowType voiceUpStream;
}; // end of S1
```

Las plantillas de interfaz que heredan pueden ser definidas como sigue:

```
interface I4: I1{
        ...
        typedef … DataType41;
        void operation41 (in DataType41 …);
}; // end of I4
```

```
interface S2: S1{
        ...
        typedef … FlowTypeS21;
        source FlowType21 videoFlow21;
}; // end of S2
```

La plantilla de objeto O4, que utiliza las plantillas de interfaz especializadas heredadas, puede ser definida como sigue:

```
CO O4{
        behaviour
        ...
        supports
                I4, S2;
        ...
}; // end of O4
```

5.5.3.2 Herencia de plantilla de objeto

Se supone que la herencia de plantilla de objeto está destinada a proporcionar "reutilización por especialización" de (una especificación UIT-ODL de) una plantilla de objeto. La plantilla de objeto de la cual se hereda se denomina una plantilla de objeto básica. La plantilla de objeto que hereda se denomina la plantilla de objeto derivada. Esta especialización puede adoptar dos formas:

- Adición de plantillas de interfaz: se pueden añadir nuevas plantillas de interfaz a la lista de interfaces soportadas/requeridas del objeto básico.
- Refinamiento de plantillas de interfaz: las interfaces soportadas en los objetos básicos pueden ser especializadas en el objeto derivado.

Las reglas de herencia para plantillas de objeto son las siguientes:

Reglas generales e interfaces soportadas

(R29) Se puede derivar una plantilla de objeto de una o varias otras plantillas de objeto, cada una denominada una plantilla de objeto básica de la plantilla de objeto derivada. En el caso de derivación de múltiples plantillas de objeto básicas (herencia múltiple), el orden de derivación no es importante.

NOTA 1 – El gráfico de herencia para plantillas de objeto está completamente separado del gráfico de herencia para plantillas de interfaz.

(R30) Una plantilla de objeto derivada puede declarar nuevas subconstrucciones (tipos de datos, plantillas de interfaz). A menos que sean redefinidas, las subconstrucciones de la plantilla de objeto básica se pueden considerar como si fuesen subconstrucciones de la plantilla de objeto derivada. La herencia de objeto hace que todos los identificadores en el cierre del gráfico de herencia sean importados en el ámbito de denominación vigente.

(R31) Una plantilla de objeto no se puede especificar como una plantilla de objeto básica directa de una plantilla de objeto derivada más de una vez. Puede ser una plantilla de objeto básica indirecta más de una vez (gráfico de herencia de "forma de diamante").

(R32) La plantilla de interfaz que puede ser ofrecida en un objeto derivado es la unión de las plantillas de interfaz soportadas en todos los objetos básicos, más cualesquiera plantillas de interfaz adicionales declaradas soportadas en la plantilla de objeto derivada.

NOTA 2 – Para añadir una nueva plantilla de interfaz a la lista de plantillas de interfaz heredadas de plantillas de objeto básicas, es suficiente declarar una plantilla de interfaz soportada adicional en la plantilla de objeto (adición de interfaz).

NOTA 3 – Para refinar una plantilla de interfaz soportada por plantillas de objeto básicas de un objeto, es suficiente declarar una plantilla de interfaz soportada en la plantilla de objeto, cuando esa plantilla de interfaz soportada es derivada de la anterior (refinamiento de plantilla de interfaz). El objeto puede después ofrecer casos de cualquiera de estas plantillas de interfaz.

(R33) Una plantilla de objeto derivada puede redefinir identificadores de tipos de datos heredados. Un identificador de tipo de datos de un ámbito abarcador puede ser redefinido en el ámbito vigente.

Comportamiento

(R34) El texto de comportamiento (behaviourText) de las plantillas de objeto básicas no está disponible en una plantilla en las plantillas de objeto derivadas.

(R35) Las interfaces requeridas de la plantilla de objeto básica es la unión de las interfaces requeridas de las plantillas de objeto básicas, más cualesquiera interfaces requeridas adicionales específicas de la plantilla de objeto derivada.

Iniciales

(R36) Las interfaces iniciales de las plantillas de objeto básicas no están disponibles en una plantilla de objeto derivada; las interfaces iniciales no son heredadas.

La plantilla de interfaz inicial de una plantilla de objeto derivada debe ser derivada de (o puede ser idéntica a, en el caso de herencia de una sola plantilla de objeto) las plantillas de interfaces iniciales de plantillas de objeto básicas directas. En los demás casos, un sistema de gestión (por ejemplo) tendrá dificultad para ver las plantillas de objeto derivadas como equivalentes a sus plantillas básicas.

Como un ejemplo ilustrativo, considérese la plantilla de objeto O6 que soporta las siguientes:

- plantilla de interfaz I1 que es soportada por las plantillas de objeto O1 y O2;
- plantilla de interfaz I5 que soporta la operación Op21 que está también definida en la plantilla de interfaz I2 soportada por O1, y además, Op51;
- plantilla de interfaz I3 soportada por O2;
- interfaz I6 que soporta la operación Op61.

Se pueden establecer las siguientes relaciones de herencia entre las plantillas de objeto O1, O2, y O6 (véase la figura 5):

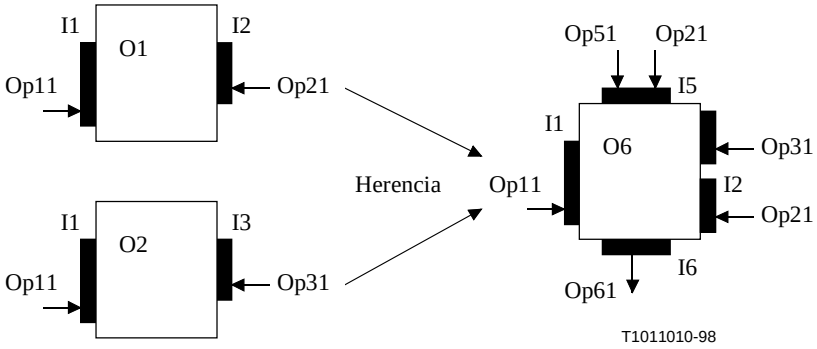

Figura 5/Z.130 – Herencia entre plantillas de objeto

En este caso, la definición de O6 se puede construir a partir de O1 y O2 con la ayuda de las reglas de herencia, y la adición de nuevos miembros.

La plantilla de interfaz I5 es declarada como derivada de la interfaz I2 y la plantilla de interfaz I6 es declarada aislada:

```
interface I5: I2{
        ...
        typedef … DataType51;
        void Op51(in DataType51 …);
}; // end of I5

interface I6{
        ...
        typedef … DataType61;
        void Op61(in DataType61 …);
}; // end of I6
```

La plantilla de objeto O6 es declarada como derivada de las plantillas de objeto O1 y O2, y las plantillas de interfaz I5 e I6 son declaradas en la lista de plantillas de interfaz soportadas de O6:

```
CO O6: O1, O2{
        ...
        supports
                I5, I6;
        ...
}; // end of O6
```

Como no hay relación de herencia de la plantilla de interfaz I6 con cualquier plantilla de interfaz declarada en las plantillas de objeto básicas O1 u O2, la plantilla de interfaz I6 se considera simplemente una interfaz soportada de O6.

Las plantillas de interfaz I3 y I2 aparecen en O2, y en las reglas de herencia se consideran también como interfaces soportadas de O6.

La plantilla de interfaz I1 aparece en O1 y O2, y en las reglas de herencia se considera también como una interfaz soportada de O6.

La plantilla de interfaz I5 hereda de I2 y ofrece las operaciones Op21 y Op51. Obsérvese que el objeto puede ejemplificar el tipo básico (I2) así como el subtipo (I5).

5.5.3.3 Herencia de plantilla de grupo de objetos

Se supone que la herencia de plantilla de grupo proporciona "reutilización por especialización" de (una especificación de UIT-ODL de) una plantilla de grupo. La plantilla de grupo de la cual se hereda se denomina una plantilla de grupo básica. La plantilla de grupo que hereda se denomina la plantilla de grupo derivada. Esta especialización puede adoptar cualquiera de las dos formas básicas siguientes:

- Adición de plantillas de objetos/plantillas de grupo: se pueden añadir nuevas plantillas de objeto a la lista de miembros de la plantilla de grupo básica.
- Refinamiento de plantillas de objeto/plantillas de grupos: los miembros de las plantillas de grupo básicas pueden ser especializados en las plantillas de grupo derivadas.

Las reglas de herencia para plantillas de grupo son las siguientes:

Reglas generales y miembros

(R37) Una plantilla de grupo puede ser derivada de una o varias otras plantillas de grupo, cada una denominada una plantilla básica de la plantilla de grupo derivada. En el caso de derivación

de múltiples plantillas de grupo básicas (múltiple herencia), el orden de derivación no es importante.

NOTA 1 – El gráfico de herencia para plantillas de objeto está completamente separado de los gráficos de herencia para plantillas de objeto y de interfaz.

(R38) Una plantilla de grupo derivada puede declarar nuevas subconstrucciones (tipos de datos, plantillas de interfaz, plantillas de objeto, plantillas de grupo). A menos que sean redefinidas, las subconstrucciones de la plantilla de grupo básica pueden ser consideradas como si fuesen subconstrucciones de la plantilla de grupo derivada. La herencia de plantilla de grupo hace que todos los identificadores en el cierre del gráfico de herencia sean importados en el ámbito de denominación vigente.

(R39) Una plantilla de grupo no puede ser especificada como una plantilla de grupo básica directa de una plantilla de grupo derivada más de una vez. Puede ser una plantilla de grupo básica indirecta más de una vez (gráfico de herencia de "forma de diamante").

(R40) Las plantillas de objeto/grupo que pueden comprender una plantilla de grupo derivada constituyen la unión de las plantillas de objeto/grupo de miembros declaradas en todas las plantillas de grupo básicas, más cualesquiera plantillas de objeto/grupo adicionales soportadas en la plantilla de grupo derivada.

NOTA 2 – Para añadir un nuevo objeto/grupo a la lista de plantillas de miembro heredadas de las plantillas de grupo básicas, es suficiente declarar una plantilla de objeto/grupo de miembro adicional en la plantilla de grupo (adición de objeto/grupo).

NOTA 3 – Para refinar una plantilla de objeto/grupo soportada por plantillas de grupo básicas de un grupo, es suficiente declarar una plantilla de objeto/grupo de miembro en la plantilla de grupo, cuando esa plantilla de objeto/grupo de miembro es derivada de la anterior (refinamiento de las plantillas de objeto/grupo). El grupo puede incluir casos de estas plantillas de objeto/grupo.

(R41) Una plantilla de grupo derivada puede redefinir identificadores de tipos de datos heredados. Un identificador de tipo de datos de un ámbito abarcador puede ser redefinido en el ámbito vigente.

Comportamiento

(R42) Los predicados de grupos básicos no están disponibles en un grupo derivado. El predicado del grupo derivado está definiendo los criterios que satisfacen todos los miembros del grupo.

Contratos

(R43) Los contratos requeridos/soportados que comprenden una plantilla de grupo derivada constituyen la unión de los contratos requeridos/soportados que comprenden las plantillas de grupo básico, más cualesquiera contratos requeridos/soportados adicionales declarados en la plantilla de grupo derivada.

Si los predicados del subgrupo son diferentes de los de la plantilla de base, se seguirán heredando los contratos si sus definiciones son compatibles[5]. El usuario debe evitar utilizar la herencia si los predicados de las plantillas de grupo básico y de la plantilla de subgrupo no son compatibles.

5.5.3.4 Denominación y fijación de ámbito con respecto a herencia

Se añaden las siguientes reglas de fijación de ámbito para soportar las capacidades de herencia:

(R44) La herencia introduce identificadores en la plantilla de interfaz derivada, la plantilla de objeto o la plantilla de grupo de objetos.

---

[5] La definición de compatibilidad de predicado no es el objeto de esta Recomendación.

(R45) La herencia de plantillas de interfaz, plantillas de objeto o plantillas de grupos de objeto introduce múltiples identificadores UIT-ODL globales para los identificadores heredados.

(R46) Un nombre calificado (uno de la forma <scoped-name>::<identifier>) es resuelto colocando la definición de <identifier> dentro del ámbito. El identificador debe ser definido directamente en el ámbito o (si el ámbito es una plantilla de grupo de objetos, plantilla de objeto o plantilla de interfaz) heredado en el ámbito. No se busca el identificador en los ámbitos abarcadores.

6 Especificación del UIT-ODL

Esta cláusula define la sintaxis de UIT-ODL. Se divide en cuatro partes, que tratan de las principales construcciones de UIT-ODL:

- declaración de tipo y constante;
- plantillas de interfaz;
- plantillas de objeto;
- plantillas de grupo de objetos.

6.1 Declaración de tipo y constante

6.1.1 Estructura

Los tipos de datos y las constantes pueden ser declarados en casi cualquier ámbito dentro de una especificación UIT-ODL. Estos tipos o constantes pueden ser utilizados para declaración de operación, excepción, flujo y otras construcciones de plantilla. Como para cualquier declaración de plantilla, se requiere que un tipo o constante sea declarado antes de su utilización (es decir, previamente en el fichero).

La sintaxis soportada por el UIT-ODL para la declaración de tipo y constante es idéntica a la del ODP-IDL. En el apéndice II el lector puede encontrar una descripción de esta sintaxis.

6.1.2 Ejemplo de declaraciones de tipo y constantes

El siguiente ejemplo muestra la declaración de tres tipos de datos: Bps, que es un sinónimo para flotante; Guarantee, que es una enumeración; y AudioQoS, que es una estructura.

```
typedef float Bps;

enum Guarantee {
        Deterministic,
        Statistical,
        BestEffort
};
struct AudioQoS {
        union Throughput switch (Guarantee){
                case Statistical:       Bps   mean;
                case Deterministic:     Bps   peak;
                case BestEffort:        range struct Interval {
                                                Bps min;
                                                Bps maxd;
                                        };
        };
```

```
        union Jitter switch (Guarantee) {
                case Statistical:       Bps   mean;
                case Deterministic:     Bps   peak;
        };
};
```

6.2 Plantilla de interfaz

6.2.1 Estructura

Una plantilla de interfaz computacional comprende:

- una especificación de comportamiento (textual),

  y, según proceda:

- una firma de interfaz operacional; o
- una firma de interfaz de trenes.

Esta estructura está reflejada en la regla del UIT-ODL para <interface_body>. En las siguientes subcláusulas de esta Recomendación se examinan los elementos de esta estructura con más detalle.

Se define la siguiente sintaxis para la declaración de plantilla de interfaz:

```
<interface_template>        ::=  <interface_header> "{" <interface_body> "}"
<interface_header>          ::=  "interface" <identifier>
                                 [ <interf_inheritance_spec> ]
<interf_inheritance_spec>   ::=":" <scoped_name> { "," <scoped_name> }*
<interface_body>            ::=  [<interf_behaviour_spec>]
                                 { <op_sig_defns> | <stream_sig_defns> }
<interf_behaviour_spec>     ::=  "behaviour" {
                                 {<interf_behaviour_text> [<interf_usage_spec>]}
                                 | <interf_usage_spec> }
<interf_behaviour_text>     ::=  "behaviourText" <string_literal> ";"
<interf_usage_spec>         ::=  "usage" <string_literal> ";"
```

6.2.2 Herencia de plantilla de interfaz

Las reglas que se aplican a la herencia de plantilla de interfaz son las especificadas para el ODP-IDL, con extensiones para tratar flujos.

Se define la siguiente sintaxis para la herencia de plantilla de interfaz:

```
<interface_header>          ::=  "interface" <identifier> [ <interf_inheritance_spec> ]
<interf_inheritance_spec>   ::=  ":" <scoped_name> { "," <scoped_name> }*
```

Se ha de señalar que la especificación de ODP-IDL, en su forma vigente prohibe la redefinición de un identificador de operación en una especificación de plantilla de interfaz derivada y prohibe la herencia de dos operaciones del mismo identificador. Inicialmente, el UIT-ODL estará restringido de acuerdo con esta limitación de ODP-IDL en las definiciones de plantillas de interfaz operacional y de trenes.

De conformidad con el ODP-IDL, la herencia de plantilla de interfaz es el equivalente de la simple inclusión de todos los atributos (incluidos los atributos de interfaz), operaciones y flujos de la plantilla de interfaz básica en la plantilla de interfaz derivada. Esta inclusión comprende todos los atributos, operaciones y flujos de la plantilla de interfaz básica, incluidos los obtenidos por herencia de otras especificaciones de plantilla de interfaz. En consecuencia, una plantilla derivada debe ser siempre capaz de proporcionar los servicios de la plantilla de interfaz básica.

Una plantilla de interfaz de trenes no puede heredar de una plantilla de interfaz operacional y viceversa.

6.2.3 Especificación del comportamiento de plantillas de interfaz

La firma de interfaz describe solamente la escritura sintáctica de una plantilla de interfaz. La compatibilidad de firmas es menos discriminadora que la compatibilidad de comportamientos. De hecho, es posible que dos interfaces tengan firmas compatibles pero difieran completamente en su comportamiento. Esta cláusula describe cómo se especifica un comportamiento textual en el UIT-ODL.

Se define la siguiente sintaxis para la especificación de comportamiento de plantillas de interfaz:

```
<interf_behaviour_spec> ::=   "behaviour" {
                                    {<interf_behaviour_text> [<interf_usage_spec>]}
                                    | <interf_usage_spec> }
<interf_behaviour_text>  ::=  "behaviourText" <string_literal> ";"
<interf_usage_spec>      ::=  "usage" <string_literal> ";"
```

6.2.4 Firma de interfaz operacional

Una firma de interfaz operacional comprende un conjunto de firmas de interrogación y de anuncio, una para cada tipo de operación en la plantilla de interfaz. Una firma de interfaz operacional especifica la siguiente información (similar al ODP-IDL):

- un atributo de operación facultativo que especifica qué semántica de invocación el sistema de comunicación debe proporcionar cuando se invoca la operación (interrogación o anuncio);
- el tipo de la operación devolver resultado (nulo en los demás casos);
- el identificador de operación;
- una lista de parámetros (ninguno o más parámetros de la operación);
- una expresión "raises" facultativa que indica las excepciones que se pueden producir como resultado de una invocación de esta operación.

Se define la siguiente sintaxis para la firma de plantilla de interfaz operacional. Es similar a la sintaxis ODP-IDL para declaración (de plantilla) de interfaz:

```
<op_sig_defns>          ::=  { <op_sig_defn> ";" }*
<op_sig_defn>           ::=  { <announcement> | <interrogation>
<announcement>          ::=  "one-way" "void" <identifier> <parameter_dcls>
<interrogation>         ::=  <attr_dcl>
                        |    <oper_dcl>
<attr_dcl>              ::=  ["readonly"] "attribute" <param_type_spec>
                             <declarators>
<oper_dcl>              ::=  <op_type_spec> <identifier>
                             <parameter_dcls>
                             [<raises_expr>] [<context_expr>]
<op_type_spec>          ::=  <param_type_spec>
                        |    "void"
<parameter_dcls>        ::=  "(" <param_dcl> { "," <param_dcl> }* ")"
                        |    "(" ")"
<param_dcl>             ::=  <param_attribute> <param_type_spec>
                             <declarator>
```

```
<param_attribute>     ::=   "in" | "out" | "inout"
<raises_expr>         ::=   "raises"
                            "(" <scoped_name> { "," <scoped_name> }* ")"
<context_expr>        ::=   "context"
                            "(" <string_literal> { "," <string_literal> }* ")"
<param_type_spec>     ::=   <base_type_spec>
                      |     <string_type>
                      |     <scoped_name>
```

6.2.5 Atributos de interfaces operacionales

Los atributos de interfaces operacionales equivalen lógicamente a definir un par de funciones de acceso: una para fijar el valor del atributo y una para obtener el valor del atributo.

6.2.6 Firma (de flujo) de trenes

Una plantilla de interfaz de trenes comprende un conjunto de tipos de flujo. Cada tipo de flujo contiene el identificador del flujo, el tipo de información del flujo, y una indicación de si es un productor o consumidor (pero no ambos) con respecto al objeto que proporciona el servicio definido por la plantilla.

La sintaxis definida aquí presupone una direccionalidad con respecto a las definiciones de plantilla de interfaz de trenes. Si dos objetos participan en una vinculación de trenes, uno es designado como proveedor de servicio, o servidor, y el otro como consumidor de servicio, o cliente. La plantilla de interfaz que describe interacciones entre ellos es expresada desde el punto de vista del cliente (que define el servidor). De muchas maneras, en particular cuando los flujos viajan en ambos sentidos, la elección de cliente y servidor puede parecer más bien arbitraria. Sin embargo, este modelo es coherente con muchos modelos de servicio familiares. Obsérvese que la plantilla de servidor incluye una declaración de que "soporta" la plantilla de interfaz de trenes, mientras que la plantilla de cliente incluye una declaración de que "requiere" la plantilla de interfaz de trenes.

Por ejemplo, para jugar un juego vídeo, un cliente (el Jugador) coloca una interfaz apropiada (VideoGame) al servidor (el Juego) (véase la figura 6). El servicio es definido naturalmente desde el punto de vista de fuentes (sources) de información (el vídeo y el audio) y sumideros (sink) [los controles etiquetados joystick1 y joystick2 (palancas de mando 1 y 2)]. Sin embargo, todas estas definiciones presuponen una direccionalidad, o punto de vista, a saber, el del cliente. La visión de servicio mantenida por el propio juego, que comprende una fuente de funciones de control y un sumidero de información vídeo y audio, puede ser obtenida fácilmente de la otra definición mediante una simple correspondencia. Como resultado, una de estas definiciones de servicio es redundante. Los adaptadores auxiliares (stubs) para el objeto Jugador (como un cliente) o para el objeto Juego (como un servidor) pueden ser producidos a partir de una especificación de plantilla de interfaz.

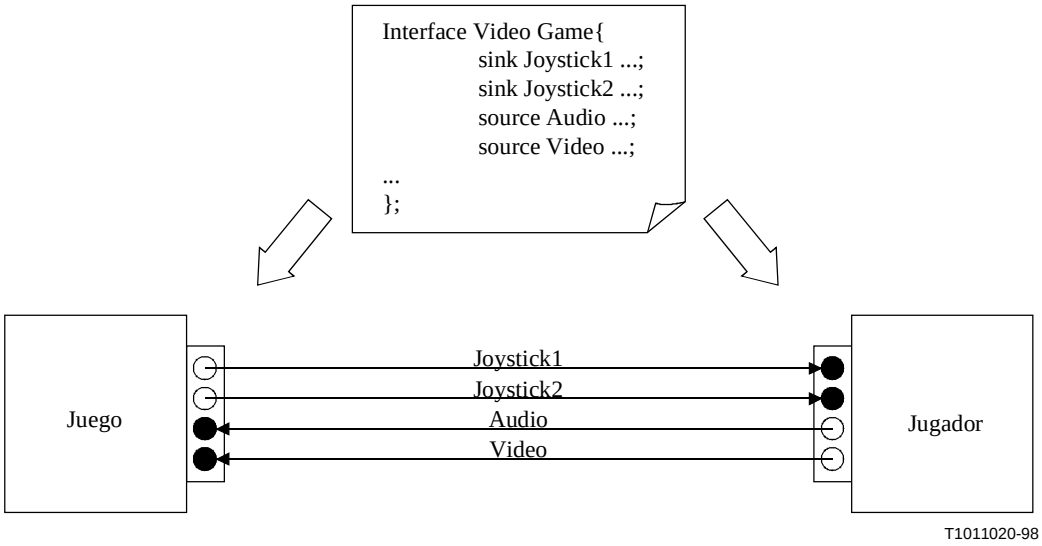

Figura 6/Z.130 – Ejemplo de plantilla de interfaz de trenes

En el UIT-ODL, las plantillas de interfaz de trenes son definidas como la visión del cliente en el servidor. Cada flujo es especificado como una fuente si la información fluye del servidor al cliente, y como un sumidero si fluye en el sentido opuesto. En la definición de plantilla de objeto del servidor, la plantilla de interfaz de trenes está indicada como una interfaz soportada, mientras que en el cliente, la plantilla de interfaz está soportada como una interfaz requerida.

```
<stream_sig_defns>          ::=   { <stream_flow_defn> ";" }*
<stream_flow_defn>          ::=   <flow_direction> <flow_type>
                                  <identifier>
<flow_direction>            ::=   "source" | "sink"
<flow_type>           ::=   <param_type_spec>
```

6.2.7 Ejemplo de declaración de plantilla de interfaz

A continuación se da un ejemplo de una plantilla de interfaz CSMConfiguration, que es derivada por herencia de una plantilla de interfaz ServiceManagement definida en el módulo Management. Contiene definiciones de operaciones, definiciones de atributos y una definición de comportamiento.

```
interface CSMConfiguration: Management:: ServiceManagement {
    behaviour
        behaviourText
        "This interface serves to configure the co CSM.

        The ReadState operation returns a complete
        representation of the CSM state. The WriteState operation
        allows the complete CSM state to be set.";

        usage
        "Operation init must be invoked prior to other
        operations defined on the service."
```

6.3 Plantilla de objeto

6.3.1 Estructura

Una especificación de plantilla de objeto comprende dos partes de alto nivel. La primera soporta la herencia, y está asociada con la declaración del identificador de la plantilla de objeto. La segunda parte es el cuerpo de la plantilla de objeto, que comprende las principales subpartes de la plantilla como sigue:

- una especificación de comportamiento;
- una especificación de interfaces requeridas;
- una especificación de interfaces soportadas; y
- una especificación de inicialización.

A continuación se examinan estas especificaciones más detalladamente, así como la sintaxis que soporta la herencia.

Se define la siguiente sintaxis para la declaración de plantilla de objeto:

```
<object_template>          ::=  <object_template_header>
                                "{" <object_template_body> "}"
<object_template_header>   ::=  "CO" <identifier>
                                [ <object_inheritance_spec> ]
<object_inheritance_spec>  ::=  ":" <scoped_name> { "," <scoped_name> }*
<object_template_body>     ::=  [<supporting_def_spec>]
                                [<interface_def_spec>]
                                [<object_behaviour_spec>]
                                [<reqrd_interf_templates>]
                                <suptd_interf_templates>
                                [<object_init_spec>]
```

6.3.2 Herencia de plantilla de objeto

La herencia de plantilla de objeto está destinada a soportar la reutilización de la especificación y proporcionar un mecanismo para definir la compatibilidad mediante relaciones de subtipificación.

Se define la siguiente sintaxis para la herencia de plantilla de objeto:

```
<object_template_header>   ::=  "CO" <identifier>[ <object_inheritance_spec> ]
<object_inheritance_spec>  ::=  ":" <scoped_name> { "," <scoped_name> }*
```

La herencia de plantilla de objeto es el equivalente de la inclusión simple de todos los atributos de constricciones, definiciones de tipos, especificaciones de plantillas de interfaces operacionales y de trenes requeridas y soportadas procedentes de las plantillas de objeto básicas en la plantilla de objeto derivada. Esta inclusión comprende todos los atributos, tipos y plantillas de interfaz de la plantilla de objeto básica, incluidos aquellos obtenidos por herencia de otras especificaciones de plantilla de objeto.

No se imponen restricciones a los nombres de plantilla de interfaz o tipos heredados de plantillas de objeto básicas del UIT-ODL.

La interfaz inicial especificada en una plantilla de objeto derivada debe ser de un tipo que sea igual que, o derivado de, todas las plantillas de interfaces iniciales de las correspondientes plantillas de objeto básico.

6.3.3 Especificación de comportamiento de plantillas de objeto

El comportamiento de un objeto se especifica como una cadena en la plantilla de objeto, que debe describir la función de un objeto al proporcionar servicios mediante cada una de sus interfaces.

Se define la siguiente sintaxis para la especificación de comportamiento de plantillas de objeto:

```
<object_behaviour_spec>        ::=   "behaviour"
                                     <string_literal> ";"
```

6.3.4 Plantillas de interfaces requeridas

La segunda comprende las interfaces requeridas (declaradas) que especifica las plantillas de interfaz utilizadas por casos de la plantilla de objeto para realizar sus funciones y prestar sus servicios.

Es posible direccionar una interfaz requerida por medio de <tagged_name> directamente de una plantilla de objeto particular. En ese caso, la plantilla de objeto especificada debe tener una plantilla de interfaz soportada, que sea compatible con la plantilla de interfaz requerida especificada. Las reglas de compatibilidad están abiertas, un ejemplo son las reglas proporcionadas en [2] y [3]. Esta notación asegura que la compatibilidad entre las interfaces requeridas y soportadas puede ser verificada en este nivel de especificación estática.

Se define la siguiente sintaxis para la definición de plantillas de interfaces requeridas:

```
<reqrd_interf_templates>       ::=   "requires" <req_interf_defn>
                                     {"," <req_interf_defn> }* ";"
<req_interf_defn>              ::=   <scoped_name> | <tagged_name>
```

6.3.5 Plantillas de interfaces soportadas

Las interfaces soportadas (declaradas) de una plantilla de objeto son las interfaces enumeradas como soportadas en las especificaciones de plantilla. Los casos de plantillas de interfaz declaradas como soportadas pueden ser ofrecidas por casos de plantillas que se definen.

Se define la siguiente sintaxis para la declaración de interfaz soportada:

```
<suptd_interf_templates>       ::= "supports" <suptd_interf> ";"
<suptd_interf>                 ::= <suptd_interf_defn> {"," <suptd_interf_defn> }*
<suptd_interf_defn>            ::= <scoped_name> |      <interface_template>
```

6.3.6 Especificación de inicialización de plantilla de objeto

La especificación de inicialización identifica una plantilla de interfaz, una referencia a la cual será retornada al creador de la plantilla de objeto que se define. Esta interfaz puede ser utilizada para inicializar el objeto recientemente creado. Se ha de señalar que la interfaz inicial es también una de las interfaces operacionales soportadas[6].

Se define la siguiente sintaxis para la especificación de inicialización:

```
<object_init_spec>             ::=   "initial"
                                     { <scoped_name> | <interface_template> } ";"
```

6.3.7 Ejemplo de declaración de plantilla de objeto

A continuación se da un ejemplo que muestra cómo se declara una plantilla de objeto. Comienza con la palabra clave "CO" que va seguida por el identificador de la plantilla de objeto, CSMfactory. Esta plantilla no hereda de ninguna otra, según lo indicado por la ausencia de cualesquiera especificaciones de herencia. El cuerpo de la plantilla se declara entre los corchetes.

---

6 La interfaz inicial no necesita incluirse en la cláusula de definición de interfaz soportada.

```
CO CSMfactory {
        requires
                QoSmanagerIF;

        supports
                Management,
                LcgFactory,
                CSMConfiguration;

        initial
                Management;

}; // end CSMfactory
```

La plantilla de interfaz MyManagement hereda de la interfaz inicial de la plantilla de objeto básica. Cabe señalar que la plantilla de objeto derivada soporta una plantilla de interfaz, MyCSMConfiguration, que es derivada de una plantilla de interfaz soportada por la plantilla básica, CSMConfiguration. La creación de uno o ambos de estos tipos en cualquier instante determinado es una decisión de la realización. Una realización de la plantilla de objeto MyCSMfactory puede crear ninguno o más casos de CSMConfiguration, por ejemplo, y es aún conforme a esta especificación. El número de casos de cualquier plantilla de interfaz determinada puede estar constreñido por la especificación de comportamiento de la plantilla de objeto.

```
interface MyManagement: Management{
        ...
}; // end MyManagement


interface MyCSMConfiguration: CSMConfiguration{
        ...
}; // end MyCSMConfiguration


CO MyCSMfactory: CSMfactory{

        requires
                AccountingEventIF;
        supports
                MyManagement,
                MyCSMConfiguration;

        initial
                MyManagement;

}; // end MyCSMfactory
```

A continuación se da un ejemplo de especificación de comportamiento.

```
CO Timer{

        behaviour
                "Instances of this co periodically call the
                tick function of a specified TimerInterrupt
                interface.";

        requires
                TimeInterrupt;
...
};
```

6.4 Plantilla de grupo de objetos

6.4.1 Estructura

Una especificación de plantilla de grupo comprende dos partes de alto nivel. La primera soporta la herencia y está asociada con la declaración del identificador de la plantilla de grupo. La segunda parte es el cuerpo de la plantilla de grupo, que comprende las principales subpartes de la plantilla, como sigue:

- una especificación de comportamiento;
- una especificación de plantillas de objetos contenidas y plantillas de grupos de objetos; y
- una especificación de plantillas de interfaz, cuyos casos son visibles fuera del grupo.

A continuación se examinan estas especificaciones con más detalle, así como la sintaxis que soporta la herencia.

Se define la siguiente sintaxis para la declaración de plantilla de grupo de objetos:

```
<group_template>          ::=  <group_template_header>
                               "{" <group_template_body> "}"
<group_template_header>   ::=  "group" <identifier>
                               [ <group_inheritance_spec> ]
<group_template_body>     ::=  [<supporting_def_spec>]
                               [<interface_def_spec>]
                               [<object_def_spec>]
                               [<group_def_spec>]
                               [<group_predicate_spec>]
                               <supp_comp_templates>
                               [<supported_contract_interfaces>]
                               [<required_contract_interfaces>]
```

6.4.2 Herencia de plantilla de grupo de objetos

La herencia de plantilla de grupo de objetos está destinada a soportar la reutilización de la especificación y proporcionar un mecanismo para definir la compatibilidad mediante relaciones de subtipificación. La finalidad de esta compatibilidad para las especificaciones de plantilla de objeto es soportar el uso de especificaciones de marco de objetos.

Se define la siguiente sintaxis para la herencia de grupo:

```
<group_template_header>     ::=  "group" <identifier> [ <group_inheritance_spec> ]
<group_inheritance_spec>    ::=  ":" <scoped_name> { "," <scoped_name> }*
```

La herencia de plantilla de grupo es el equivalente de la simple inclusión de todas las definiciones de tipos, contratos y especificaciones de miembro procedentes de las plantillas de grupo básicas en la plantilla de grupo derivada. Esta inclusión comprende todos los atributos, tipos y plantillas de objeto del grupo básico, incluidos los obtenidos por herencia de otras especificaciones de plantilla de grupo.

No se imponen restricciones a los identificadores de plantillas de objeto o tipos heredados en plantillas de grupo.

6.4.3 Especificación de predicado de plantilla de grupo de objetos

Las especificaciones de predicado de una plantilla de grupo de objetos tiene la finalidad de identificar los criterios que satisfacen todos los miembros del grupo. La gama de posibles criterios está abierta. Entre los ejemplos cabe citar:

- Finalidad de estructuración.
- Aspectos de gestión (miembros de dominio, aplicación de las mismas políticas).

- Aspectos de la realización (los miembros del grupo juntos prestan un determinado servicio).

Dependiendo de los criterios, la especificación de predicado de plantilla de grupo puede contener cualquier información adicional que sea útil en el contexto.

Se define la siguiente sintaxis para la especificación de predicado de grupo:

<group_predicate_spec> ::= "predicate" <string_literal> ";"

6.4.4 Plantillas de objeto de miembros y plantillas de grupo

Las plantillas de objeto de miembros y las plantillas de grupo de objetos son las plantillas de objeto y las plantillas de grupo de objetos que pertenecen a la plantilla de grupo de objetos.

NOTA – Un objeto o grupo puede estar contenido en más de un grupo.

A continuación se indica la sintaxis que soporta las plantillas de objeto de miembros:

<supp_comp_templates> ::= "members" <suptd_comp> ";"
<suptd_comp> ::= <suptd_comp_defn> {"," <suptd_comp_defn> }*
<suptd_comp_defn> ::= <scoped_name>
| <object_template>
| <group_template>

6.4.5 Contratos

Los contratos son las interfaces de los miembros del grupo de objeto que son visibles a entidades fuera del grupo de objetos. Los miembros sólo pueden ser accedidos desde el entorno por medio de estas interfaces. En la especificación de plantilla de grupo de objetos están indicados como una simple lista de identificadores (facultativamente con fijación de ámbito).

A continuación se indica la sintaxis que soporta contratos:

<supported_contract_interfaces> ::= "supports" <scoped_name> { "," <scoped_name> }* ";"
<required_contract_interfaces> ::= "requires" {<scoped_name> | <tagged_name>}
{ "," {<scoped_name> | <tagged_name>} }* ";"

6.4.6 Ejemplo de declaración de plantilla de grupo

A continuación se da un ejemplo que muestra cómo se declara una plantilla de grupo. El ejemplo presentado es una plantilla de grupo de objetos subnetManager.

```
interface Configuration {...};
interface Configurator {...};
interface Trail {...};
interface TC {...};

CO CMC {

        requires
                Configuration;
        supports
                Configurator;
        ...
};
CO NetworkCoordinator {
        requires
                TC, SncService, SncServiceFactory;
        supports
                Trail, TC, Configuration;
        ...
};
```

```
CO NetworkCP {
        supports
                SncService, SncServiceFactory, Configuration;
        ...
};
CO ElementCP {...};


group SubnetManager {
        predicate
                "This group manages a subnetwork"

        members
                CMC, NetworkCoordinator, CMC, NetworkCP, ElementCP;

        supported
                Configurator, Trail, TC;
};
```

ANEXO A

Formato de Backus-Naur (BNF, *Backus-Naur format*)

A.1 Conformidad

En [9] está disponible un aceptador de referencia para el UIT-ODL. Cuando existan ambigüedades entre el texto en esta Recomendación y el aceptador de referencia, el texto tiene precedencia.

NOTA – Un aceptador es una herramienta que calcula si una determinada especificación es conforme o no con el UIT-ODL.

A.2 Convenios léxicos

El UIT-ODL utiliza los convenios léxicos y de preprocesador del ODP-IDL [8].

A.3 Palabras clave

La mayoría de las palabras clave son importadas del ODP-IDL pero se añaden varias para soportar las extensiones del UIT-ODL con respecto al ODP-IDL. Estas palabras clave aparecen subrayadas.

any	attribute	behaviour	behaviourText	boolean	case
char	members	const	context	default	double
enum	exception	FALSE	fixed	float	group
in	initial	inout	interface	long	module
Object	CO	octet	one-way	out	predicate
raises	requires	readonly	sequence	short	sink
source	string	struct	supports	switch	TRUE
typedef	unsigned	union	usage	void	wchar
wstring					

A.4 Notación BNF ampliada

Los siguientes metasímbolos se utilizan para describir la sintaxis del UIT-ODL. La descripción se efectúa en un formato Backus-Naur ampliado (BNF)[7].

Símbolo	Significado
`::=`	Se define que ha de ser
`|`	Alternativamente
`<text>`	No terminal
`"text"`	Terminal (es decir, literal)
`*`	La unidad sintáctica precedente puede ser repetida ninguna o más veces
`+`	La unidad sintáctica precedente puede ser repetida ninguna o más veces
`{}`	Las unidades sintácticas colocadas entre los corchetes están agrupadas como una sola unidad sintáctica
`[]`	La unidad sintáctica colocada entre los corchetes es facultativa, puede aparecer ninguna o una vez

A.5 Sintaxis

A continuación se presenta la sintaxis para el UIT-ODL. Las expresiones tomadas del ODP-IDL están marcadas con un asterisco "*".

Sintaxis de nivel máximo

```
<odl_spec>          ::=  <definition>*
<definition>        ::=  <module> ";"
                    |    <group_dcl> ";"
                    |    <object_dcl> ";"
                    |    <interface_dcl> ";"
                    |    <supporting_def> ";"
```

A.5.1 Sintaxis de módulo

```
<module>            ::=  "module" <identifier> "{" <definition>+ "}"
<scoped_name>       ::=  <identifier>
                    |    "::"<identifier>
                    |    <scoped_name> "::" <identifier>
<tagged_name>       ::=  <scoped_name> "." <scoped_name>
```

A.5.2 Sintaxis de grupo

```
<group_dcl>             ::=  <group_forward_dcl>
                        |    <group_template>
<group_forward_dcl>     ::=  “group” <identifier>
<group_template>        ::=  <group_template_header>
                             “{“ <group_template_body> “}”
<group_template_header> ::=  “group”<identifier>
                             [ <group_inheritance_spec> ]
<group_inheritance_spec> ::= “:” <scoped_name> { “,” <scoped_name> }*
```

---

[7] Obsérvese que la concatenación de símbolos tiene una precedencia más alta que |. Por ejemplo, X X X | Y Y Y es equivalente a {X X X} | {Y Y Y}.

```
<group_template_body>          ::=   [<supporting_def_spec>]
                                     [<interface_def_spec>]
                                     [<object_def_spec>]
                                     [<group_def_spec>]
                                     [<group_predicate_spec>]
                                     <supp_comp_templates>
                                     [<supported_contract_interfaces>]
                                     [<required_contract_interfaces>]
<supporting_def_spec>          ::=   {<supporting_def> “;”}*
<interface_def_spec>           ::=   {<interface_dcl> “;”}*
<object_def_spec>              ::=   {<object_dcl> “;” }
<group_def_spec>               ::=   {<group_dcl> “;” }
<group_predicate_spec>         ::=   “predicate” <string_literal> “;”
<supp_comp_templates>          ::=   “members” <supp_comp> “;”
<supp_comp>                    ::=   <supp_comp_defn> {“,” <supp_comp_defn> }*
<supp_comp_defn>               ::=   <scoped_name>
<supported_contract_interfaces ::=   “supports” <scoped_name>
                                     { “,” <scoped_name> }* “;”
<required_contract_interfaces> ::=   “requires” {<scoped_name> | <tagged_name>}
                                     { “,” {<scoped_name> | <tagged_name>} }* “;”
```

A.5.3 Sintaxis de objeto

```
<object_dcl>                   ::=   <object_forward_dcl>
                               |     <object_template>
<object_forward_dcl>           ::=   "CO" <identifier>
<object_template>              ::=   <object_template_header>
                                     "{" <object_template_body> "}"
<object_template_header>       ::=   "CO" <identifier>
                                     [ <object_inheritance_spec> ]
<object_inheritance_spec>      ::=   ":" <scoped_name> { "," <scoped_name> }*
<object_template_body>         ::=   [<supporting_def_spec>]
                                     [<interface_def_spec>]
                                     [<object_behaviour_spec>]
                                     <suptd_interf_templates>
                                     [<reqrd_interf_templates>]
                                     [<object_init_spec>]
<object_behaviour_spec>        ::=   "behaviour" <string_literal> ";"
<reqrd_interf_templates>       ::=   "requires" <req_interf_defn>
                                     {"," <req_interf_defn> }* ";"
<req_interf_defn>              ::=   <scoped_name>
                               |     <tagged_name>
<suptd_interf_templates>       ::=   "supports" <suptd_interf> ";"
<suptd_interf>                 ::=   <suptd_interf_defn> {"," <suptd_interf_defn> }*
<suptd_interf_defn>            ::=   <scoped_name>
<object_init_spec>             ::=   "initial" <scoped_name> ";"
```

A.5.4 Sintaxis de interfaz

```
<interface_dcl>                ::=   <interface_forward_dcl>
                               |     <interface_template>
<interface_forward_dcl>        ::=   "interface" <identifier>
<interface_template>           ::=   <interface_header> "{" <interface_body> "}"
<interface_header>             ::=   "interface" <identifier>
                                     [ <interf_inheritance_spec> ]
<interf_inheritance_spec>      ::=    ":" <scoped_name> { "," <scoped_name> }*
<interface_body>               ::=   [<supporting_def_spec>]
                                     [<interf_behaviour_spec>]
                                     { <op_sig_defns> | <stream_sig_defns> }
```

```
<interf_behaviour_spec>     ::=   "behaviour" {
                                      {<interf_behaviour_text> [<interf_usage_spec>]}
                                |     <interf_usage_spec> }
<interf_behaviour_text>     ::=   "behaviourText" <string_literal> ";"
<interf_usage_spec>         ::=   "usage" <string_literal> ";"
```

A.5.5 Sintaxis de interfaz (operacional)

```
<op_sig_defns>              ::=   { <op_sig_defn> ";" }*
<op_sig_defn>               ::=   { <announcement> | <interrogation> }
 <announcement>             ::=   "one-way" "void" <identifier> <parameter_dcls>
<interrogation>             ::=   <attr_dcl>
                            |     <oper_dcl>
<attr_dcl>                  ::=   ["readonly"] "attribute" <param_type_spec>
                                  <declarators>
<oper_dcl>                  ::=   <op_type_spec> <identifier>
                                  <parameter_dcls>
                                  [<raises_expr>] [<context_expr>]
<op_type_spec>              ::=   <param_type_spec>
                            |     "void"
<parameter_dcls>            ::=   "(" <param_dcl> { "," <param_dcl> }* ")"
                            |     "(" ")"
<param_dcl>                 ::=   <param_attribute> <param_type_spec>
                                  <declarator>
<param_attribute>           ::=   "in" | "out" | "inout"
<raises_expr>               ::=   "raises"
                                  "(" <scoped_name> { "," <scoped_name> }* ")"
<context_expr>              ::=   "context"
                                  "(" <string_literal> { "," <string_literal> }* ")"
<param_type_spec>           ::=   <base_type_spec>
                            |     <string_type>
                            |     <wide_string_type>
                            |     <fixed_pt_type>
                            |     <scoped_name>
```

A.5.6 Sintaxis de interfaz (de trenes)

```
<stream_sig_defns>          ::=   { <stream_flow_defn> ";" }*
<stream_flow_defn>          ::=   <flow_direction> <flow_type>
                                  <identifier>
<flow_direction>            ::=   "source" | "sink"
<flow_type>           ::=   <param_type_spec>
```

A.5.7 Sintaxis de definición soportadora

```
<supporting_def>            ::=   <const_dcl> ";"
                            |     <type_dcl> ";"
                            |     <except_dcl> ";"
<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_expr>
```

```
<or_expr>               ::=  <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> "%" <unary_expr>
<unary_expr>            ::=  <unary_operator> <primary_expr>
                        |    <primary_expr>
<unary_operator>        ::=  "-"
                        |    "+"
                        |    "~"
<primary_expr>          ::=  <scoped_name>
                        |    <literal>
                        |    "(" <const_exp> ")"
<literal>               ::=  <integer_literal>
                        |    <string_literal>
                        |    <wide_string_literal>
                        |    <character_literal>
                        |    <wide_character_literal>
                        |    <fixed_pt_literal>
                        |    <floating_pt_literal>
                        |    <boolean_literal>
<boolean_literal>       ::=  "TRUE"
                        |    "FALSE"
<typedcl>               ::=  "typedef" <type_declarator>
                        |    <struct_type>
                        |    <union_type>
                        |    <enum_type>
<type_declarator>       ::=  <type_spec> <declarators>
<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>
<object type>           ::=  "Object"
<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_longlong_int>        ::=  "long" "long"
<signed_long_int>            ::=  "long"
<signed_short_int>           ::=  "short"
<unsigned_int>               ::=  <unsigned_long_int>
                             |    <unsigned_short_int>
                             |    <unsigned_longlong_int>
<unsigned_longlong_int>      ::=  "unsigned" "long" "long"<unsigned_long_int>
                             ::=  "unsigned" "long"
<unsigned_short_int>         ::=  "unsigned" "short"
<char_type>                  ::=  "char"
<wide_char_type>             ::=  "wchar"
<boolean_type>               ::=  "boolean"
<octet_type>                 ::=  "octet"
<any_type>                   ::=  "any"
<template_type_spec>         ::=  <sequence_type>
                             |    <string_type>
                             |    <wide_string_type>
                             |    <fixed_pt_type>
<sequence_type>              ::=  "sequence" "<" <simple_type_spec> ","
                                  <positive_int_const> ">"
                             |    "sequence" "<" <simple_type_spec> ">"
<string_type>                ::=  "string" "<" <positive_int_const> ">"
                             |    "string"
<wide_string_type>           ::=  "wstring" "<" <positivi_int_const> ">"
                             |    "wstring"
<fixed_pt_type>              ::=  "fixed" "<" postive_int_const> "," <integer_literal> ">"
<fixed_pt_const_type>        ::=  "fixed"
<constr_type_spec>           ::=  <struct_type>
                             |    <union_type>
                             |    <enum_type>
<struct_type>                ::=  "struct" <identifier> "{" <member_list> "}"
<member_list>                ::=  <member>+
<member>                     ::=  <type_spec> <declarators> ";"
<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>
<declarators>                ::=  <declarator> { "," <declarator> }*
<declarator>                 ::=  <simple_declarator>
                             |    <complex_declarator>
<simple_declarator>          ::=  <identifier>
<complex_declarator>         ::=  <array_declarator>
<array_declarator>           ::=  <identifier> <fixed_array_size>+
<fixed_array_size>           ::=  "[" <positive_int_const> "]"
<positive_int_const>         ::=  <const_exp>
<enum_type>                  ::=  "enum" <identifier>
                                  "{" <enumerator> { "," <enumerator> }* "}"
<enumerator>                 ::=  <identifier>
<except_dcl>                 ::=  "exception" <identifier> "{" <member>* "}
```

ANEXO B

Correspondencia entre el SDL y la ASN.1

B.1 Motivación

Según lo descrito anteriormente, el lenguaje UIT-ODL está destinado a ser utilizado como una técnica de descripción para los aspectos estáticos de una especificación de punto de vista computacional. Sin embargo, para la elaboración de sistemas no triviales, podría ser también útil una especificación de comportamiento del sistema. Esta especificación de comportamiento puede ser hecha utilizando otros lenguajes normalizados por la UIT. Uno de estos lenguajes es el SDL.

Para combinar los dos lenguajes (UIT-ODL y SDL), en este anexo se propone una correspondencia de lenguajes del UIT-ODL al SDL. Dentro de este proceso todas las plantillas de objeto, plantillas de interfaz, firmas de operaciones y tipos de datos se especifican en UIT-ODL. Para especificar el comportamiento de la aplicación es posible derivar un elemento adaptador (stub) del SDL de la especificación UIT-ODL que proporciona las estructuras, firmas y tipos de datos que utiliza la correspondencia. Este adaptador (stub) puede ser enriquecido por el comportamiento aplicando el mecanismo de herencia. Esta especificación completa permite una prueba o simulación de la aplicación antes de la realización.

Por consiguiente, es necesario tener una correspondencia de lenguajes bien definida del UIT-ODL al SDL en combinación con la ASN.1. Esta correspondencia de lenguajes ofrece la posibilidad de derivar automáticamente un sistema SDL a partir de una especificación UIT-ODL. El comportamiento puede ser añadido de una forma directa con sólo sobrecargar los procedimientos virtuales.

B.2 Requisitos básicos

Como el UIT-ODL es un superconjunto del ODP-IDL, las correspondencias de lenguajes contenidas en [1] podrán servir como orientación para definir una correspondencia de lenguajes para el UIT-ODL. Se han de efectuar las siguientes correspondencias con respecto a la correspondencia de adaptador (stub) del lenguaje C que se presenta en [1].

Construcciones del UIT-ODL que existen también en el ODP-IDL:

- todos los tipos de datos básicos UIT-ODL;
- todos los tipos de datos construidos UIT-ODL;
- las constantes definidas en el UIT-ODL;
- las referencias a objetos CORBA[8] que se definen en el UIT-ODL;
- invocación de operaciones, incluidos el paso de parámetros y la recepción de resultados;
- excepciones, incluidas las que se producen cuando una operación plantea una excepción y cómo son accedidos los parámetros de excepción.
- acceso a atributos.

Otras construcciones específicas del UIT-ODL:

- plantilla de objetos;
- plantilla de grupo de objetos;

---

[8] Hay una contradicción en la definición del término objeto en el OMG y en el ODP. Esta Recomendación se basa en la definición del ODP. Si se menciona la definición del OMG, se señala como un objeto de CORBA.

- especificación de comportamiento;
- firmas de trenes.

B.3 Estructura

Un fichero UIT-ODL corresponde con dos paquetes SDL:

- <name>_interface; y
- <name>_definition;

donde <name> es el nombre de la especificación del UIT-ODL (por ejemplo, el nombre del fichero).

Package <name>_interface contiene toda la información que es pertinente para los lados cliente y servidor del sistema. En detalle, son las definiciones de tipo de datos, definiciones de constantes, definiciones de señales para operaciones, flujos, atributos y excepciones y las definiciones de listas de señales. Adicionalmente, hay definiciones de procedimiento que pueden ser utilizadas por el cliente para invocar una operación en un servidor y que manejan el intercambio de señales entre el cliente y el servidor (obsérvese que esto es sólo una notación abreviada).

Package <name>_definition contiene los esqueletos para el lado servidor del sistema. De hecho, hay tipos de proceso para plantillas de interfaz del UIT-ODL y tipos de bloque para plantillas de objeto y plantillas de grupo del UIT-ODL.

B.4 Nombres con campo de aplicación fijado

Se requiere que todas las plantillas, tipos, constantes y definiciones estén definidas en el SDL utilizando su nombre global. Esto se requiere porque el SDL no conoce un operador de ámbito.

Si se utilizan nombres globales en una especificación ASN.1, todos los subrayados tienen que ser sustituidos por guiones y todos los identificadores de tipo deben comenzar con una letra mayúscula. Todos los otros identificadores deben comenzar con una letra minúscula.

B.5 Correspondencia de módulos

Un módulo UIT-ODL no puede tener correspondencia en el SDL porque no existe un operador de ámbito en el SDL. Todas las construcciones UIT-ODL, tales como plantillas de interfaz o tipos de datos definidos en un módulo, deben ser visibles para todas las plantillas de objeto definidas fuera de este módulo. Por consiguiente, es necesario que todas las definiciones para plantillas de interfaz, tipos de datos, plantillas de objeto, etc., incluidas en un módulo sean hechas en los paquetes SDL utilizando su nombre global (véase B.4).

B.6 Correspondencia de plantillas de interfaz, operaciones, flujos y atributos

Las plantillas de interfaz UIT-ODL corresponden con tipos de procesos contenidos en package <name>_definition. Estos tipos de proceso contienen toda la especificación de comportamiento de plantilla de interfaz como un comentario (texto informal), las operaciones proporcionadas en esa interfaz como procedimientos virtuales, los flujos como declaraciones de variables distantes y los atributos como declaraciones de variables locales.

Operaciones

Actualmente las operaciones no pueden corresponder con procedimientos distantes. Esto se debe al hecho que el SDL no contiene un mecanismo para tratar excepciones. Sin embargo, el concepto de excepción es un requisito fundamental cuando se describen sistemas distribuidos. Por consiguiente, las operaciones corresponden con un conjunto de señales, como sigue:

- pCALL_<global name of operation>;
- pREPLY_<global name of operation> (no para operaciones unidireccionales).

La señal pCALL transporta todos los parámetros del cliente al servidor y en ambos sentidos de la operación, la señal pREPLY transporta todos los parámetros del servidor al cliente y en ambos sentidos de la operación y el valor de retorno. Las señales son definidas en package <name>_interface. El tipo de proceso generado para la plantilla de interfaz en package <name>_definition contiene un procedimiento local para la operación. El nombre del procedimiento es el nombre de la operación. Los parámetros de la operación son declarados como parámetros del cliente al servidor o en ambos sentidos en el procedimiento SDL. Si un parámetro se especifica como del servidor al cliente en el UIT-ODL, corresponde con un parámetro en ambos sentidos en el SDL.

Un cliente puede invocar una operación en un servidor enviando la señal pCALL apropiada. El servidor recibe la señal y (de acuerdo con su estado) llama al procedimiento local que contiene la realización de la operación. El resultado se devuelve al cliente utilizando la señal pREPLY.

NOTA 1 – Para simplificar el mecanismo de llamada en el lado cliente, package <name>_definition contiene un procedimiento para cada operación que envía la señal pCALL, espera la respuesta en el estado de espera y devuelve el resultado al cliente. Véase el ejemplo a continuación.

NOTA 2 – Las excepciones corresponden también con señales, y producir una excepción no es nada más que enviar la señal de excepción apropiada al cliente en vez de la señal pREPLY.

NOTA 3 – Para tener la seguridad de que no se mezclan las señales pCALL y pREPLY, cada invocación tiene que contener un identificador único, que se incluye después en la terminación. Esta correspondencia no prescribe una manera concreta de implementar esto.

```
exception ex{
};

interface i{
        behaviour
                behaviourText
                  "The interface serves for contacting type management";
                usage
                  "Operation op must be invoked prior to other operations defined on the service ";

        string op (
                in boolean p1,
                inout char p2,
                out octet p3
        )
         raises ( ex);

        one-way void op1(
                in double p1
        );
} /*end opr interface */;

use idltypes ;
package name_interface ;
     /* name shall be replaced by the name of the ODL specification */
     signal pCALL_i_op(ODL_boolean,ODL_char);
     signal pREPLY_i_op(ODL_char,ODL_octet,ODL_string);
     signal pCALL_i_op1(ODL_double);
     signal pREPLY_i_op1;
     signal pRAISE_ex;
     signallist i_INVOCATIONS=pCALL_i_op,pCALL_i_op1 ;
```

```
     signallist i_TERMINATIONS=pREPLY_i_op,pRAISE_ex ;
     procedure i_op ;
          fpar in p1 ODL_boolean,
               in/out p2 ODL_char,
               in/out p3 ODL_octet,
               in server ODL_Object ;
          returns ODL_string ;
          dcl ex_var ex,return_var ODL_string ;
          start;
               decision (server);
                    (Null): output pCall_i_op(p1,p2);
                    else: output pCall_i_op(p1,p2) to server ;
               enddecision;
               nextstate Wait ;
          state Wait ;
               save * ;
               input pReply_i_op(p2,p3,return_var);
                    output pReply_i_op(p2,p3,return_var) to self ;
                    return return_var;
               input pRAISE_ex;
                    output pRAISE_ex to self;
               return return_var;
          endstate Wait ;
     endprocedure i_op ;
endpackage name_interface ;

use idltypes;
use name_interface ;
package name_definition ;
     /* name shall be replaced by the name of the ODL specification */
     process type <<package name_definition>> i /*interface definition i*/
          comment
               "The interface serves for contacting type management
               Operation op must be invoked prior to other operations defined on
               the service ";
          gate i_INVOCATIONS in with (i_INVOCATIONS) ;
          gate i_TERMINATIONS out with (i_TERMINATIONS) ;
          dcl  op_p1 ODL_boolean,op_p2 ODL_char,op_p3 ODL_octet,op_return
          ODL_string,op1_p1 ODL_double ;
          virtual procedure <<package name_definition/process type i>> op ;
               fpar in p1 ODL_boolean,
                    in/out p2 ODL_char,
                    in/out p3 ODL_octet ;
               returns ODL_string ;
          endprocedure <<package name_definition/process type i>> op ;
          virtual procedure <<package name_definition/process type i>> op1 ;
               fpar in p1 ODL_double ;
          endprocedure <<package name_definition/process type i>> op1 ;
     endprocess type <<package name_definition>> i ;
endpackage name_definition ;
```

Flujos

Las firmas de trenes son representadas en SDL como declaraciones de variables distantes. El productor exporta la variable de trenes y el consumidor la importa. Esto significa en el lado servidor que un flujo especificado como sumidero corresponde con una especificación de variable importada y que un flujo especificado como fuente corresponde con una declaración de variable exportada. Estas declaraciones de variables están contenidas en el tipo de proceso generado para la plantilla de interfaz que utiliza su nombre global. En el lado cliente el tratamiento es inverso, pero no es generado automáticamente.

NOTA – Obsérvese que las variables importadas y exportadas tienen que ser declaradas como distantes en package <name>_definition.

Referencias

Las referencias de interfaces del UIT-ODL serán denotadas por los PId del SDL. Estos PId pueden ser argumentos, devolución de resultados o componentes de definiciones de tipos.

Para todas las plantillas de interfaz se introduce un nuevo tipo que representa la referencia a esa interfaz.

Ejemplo:

```
interface i { /* ITU-ODL */
...
};
IRef ::= PId; /* SDL + ASN.1 */
```

Una llamada de un cliente al servidor direccionada por una referencia especial se efectúa llamando al procedimiento contenido en package <name>_interface y añadiendo el PId del servidor como el último parámetro.

Atributos

Las declaraciones de atributos del UIT-ODL definidas en una plantilla de interfaz corresponden con la definición de variable local dentro de la definición de tipo de proceso que representa la plantilla de interfaz UIT-ODL. Adicionalmente, se definen dos procedimientos distantes exportados por el tipo de proceso que permiten el acceso distante (lectura y escritura) a estos atributos por el cliente que importa estos procedimientos distantes. Cuando un atributo es declarado de lectura solamente dentro de la especificación del UIT-ODL, sólo se proporciona un procedimiento distante que permite leer los valores del atributo correspondiente. La declaración distante se efectúa en package <name>_interface.

Ejemplo:

```
interface i3 {       /* ITU-ODL */
  attribute boolean attr1;
  readonly attribute short attr2;
};
use idltypes ;
package name_interface ;
     /* name shall be replaced by the name of the ODL specification */
     remote procedure i3__set_attr1;
          fpar in ODL_boolean;
     remote procedure i3__get_attr1;
          returns ODL_boolean;
     remote procedure i3__get_attr2;
          returns ODL_short;
endpackage;
use idltypes;
use name_interface;
package name_definition;
     /* name shall be replaced by the name of the ODL specification */
     process type i3;   /* SDL + ASN.1 */
          dcl
               attr1 ODL_boolean,
               attr2 ODL_short;
          exported procedure i3__set_attr1 referenced;
          exported procedure i3__get_attr1 referenced;
```

```
          exported procedure i3__get_attr2 referenced;
     endprocess type i3;
     exported procedure i3__set_attr1;
          fpar
               in value ODL_boolean;
          start;
               task
           attr1 := value;
          return;
     endprocedure i3__set_attr1;
     exported procedure i3__get_attr1;
          returns ODL_boolean;
          start;
               return attr1;
     endprocedure i3__get_attr1;
     exported procedure i3__get_attr2;
          returns ODL_short;
          start;
               return attr2;
     endprocedure i3__get_attr2;
endpackage;
```

B.7 Herencia de plantilla de interfaz

La herencia de plantilla de interfaz definida en el UIT-ODL tiene que estar representada como una estructura plana. Esto significa que todas las construcciones para operaciones, flujo y atributos definidas en la plantilla de interfaz básica tienen que estar definidas otra vez en la plantilla de interfaz derivada. Las especificaciones de comportamiento no se heredan.

No se puede utilizar el mecanismo de herencia del SDL porque sólo se admite la herencia única y no hay herencia múltiple como en el UIT-ODL.

Si la especificación sólo contiene una herencia, se puede aplicar la característica de herencia del SDL[9].

B.8 Correspondencia para plantillas de objeto

Las plantillas de objeto son representadas en el SDL como tipos de bloques definidos en package <name>_definition utilizando el nombre global de la plantilla de objetos. De manera similar a las plantillas de interfaz, la especificación de comportamiento de plantillas de objetos corresponde con un comentario. La correspondencia de las otras partes de la plantilla de objeto es como sigue:

Interfaces requeridas

La lista de plantillas de interfaces requerida corresponde con un comentario.

Interfaces soportadas

La correspondencia de las plantillas de interfaz anunciadas en la lista de interfaces soportadas es la siguiente:

Para cada plantilla de interfaz se introduce un nuevo tipo de proceso virtual en el tipo de bloque que representa la plantilla de objeto. Estos tipos de proceso heredan el tipo de proceso correspondiente definido para la plantilla de interfaz (véase B.6). Tiene el nombre global de la plantilla de interfaz.

---

[9] Corresponde a las herramientas utilizadas decidir si se utiliza o no la herencia del SDL.

Especificación de inicialización

Para la plantilla de interfaz anunciada en la construcción inicial se introduce un nuevo tipo de proceso virtual en la definición de tipo de bloque. Estos tipos de proceso heredan los tipos de proceso correspondientes definidos en el nivel de tipo de sistema (véase B.6). Tiene el nombre global de la plantilla de interfaz.

Herencia

La herencia de plantilla de objeto tiene que corresponder como una estructura plana. Esto se debe al hecho de que el SDL no tiene la característica de herencia múltiple. Si sólo una herencia está contenida en el fichero UIT-ODL, se puede utilizar la herencia del SDL.

Ejemplo:

```
interface i;          /*ITU-ODL*/
interface i1;
interface i2;
CO o {
      behaviour
            "use i to initialize object" ;
      requires
            i2;
      supports
            i1;
      initial
            i;
} /*end object */;

use idltypes ;
package name_interface ;
...
endpackage name_interface ;

use idltypes ;
use name_interface ;
package name_definition ;
     /* name shall be replaced by the name of the ODL specification */
     process type <<package name_definition>> i /*interface definition i*/;
          ...
     endprocess type <<package name_definition>> i ;
     process type <<package name_definition>> i1 /*interface definition i1*/;
          ...
     endprocess type <<package name_definition>> i1 ;
     process type <<package name_definition>> i2 /*interface definition i2*/;
          ...
     endprocess type <<package name_definition>> i2 ;
     block type <<package name_definition>> o /*object definition o*/
          comment "use i to initialize object"
          comment "requires i2";
          process type <<Block type o>> i1 /*supported interface i1*/
               inherits <<package name_definition>> i1;
          endprocess type <<Block type o>> i1;
          process type <<Block type o>> i /*initial interface i*/
               inherits <<package name_definition>> i;
          endprocess type <<Block type o>> i;
     endblock type <<package name_definition>> o ;
endpackage name_definition ;
```

B.9 Correspondencia para plantillas de grupo de objetos

Las plantillas de grupo de objetos corresponden con tipos de bloques en package <name>_definition utilizando el nombre global de la plantilla de grupo. Estos tipos de bloques contienen una subestructura de bloque con definiciones de tipo de bloque para cada uno de los miembros de la plantilla de grupo que heredan la forma de los tipos de bloque generados para los propios miembros. Tiene el nombre global de la plantilla de miembros.

El predicado de la plantilla de grupo corresponde con un comentario.

Los contratos no tienen correspondencia.

La herencia de plantilla de grupo tiene que corresponder como una estructura plana. Esto se debe al hecho de que el SDL no tiene la característica de herencia múltiple. Si sólo una herencia está contenida en el fichero UIT-ODL, se puede utilizar la herencia del SDL.

Ejemplo:

```
interface i;          /*ITU-ODL*/
interface i1;
interface i2;

CO o;

group g{
     predicate "All group members have the following security policy:..." ;
     members ::o;
} /*end group */;

use idltypes ;
package name_interface ;
...
endpackage name_interface ;

use idltypes ;
use name_interface ;
package name_definition ;
     /* name shall be replaced by the name of the ODL specification */
     ...
     block type <<package name_definition>> o /*object definition o*/ ;
     ...
     endblock type <<package name_definition>> o ;

     block type <<package name_definition>> g /*group definition g*/;
          block type <<Block type g>> o /*group member definition o*/
               inherits <<package name_definition>> o;
          endblock type <<Block type g>> o;
     endblock type <<package name_definition>> g ;
endpackage name_definition ;
```

B.10 Correspondencia para constantes

Las constantes definidas en el UIT-ODL corresponden con sinónimos en el SDL.

Ejemplo:

```
const float x = 7.5 + 3.4;        /* ITU-ODL */

synonym x ODL_REAL = 7.5 + 3.4;   /* SDL + ASN.1 */
```

B.11 Correspondencia para tipos de datos básicos

En esta subcláusula se muestra cómo los tipos de datos básicos del UIT-ODL se expresan en ASN.1. Todos los tipos de datos básicos se definen como tipos ASN.1 según se muestra en el siguiente cuadro[10]. Cada tipo obtiene un nombre como el siguiente: ODL_<typename> donde el nombre de tipo es uno de los tipos de datos básicos del UIT-ODL. Las variables de estos tipos pueden ser declaradas en un enunciado del SDL y pueden ser utilizadas de la misma manera que las variables de los tipos de datos SDL normales. En [7] se describe cómo utilizar los tipos de datos ASN.1 en una especificación SDL.

Tipo de datos básicos UIT-ODL	Tipos de datos ASN.1 correspondientes
Short	**INTEGER(**-2^{15}**..**$2^{15}-1$**)**
Unsigned short	**INTEGER(0..**$2^{16}-1$**)**
Long	**INTEGER(**-2^{31}**..**$2^{31}-1$**)**
Unsigned long	**INTEGER(0..**$2^{32}-1$**)**
Float	**REAL** (representa los números de coma flotante de precisión simple del IEEE[11])
Double	**REAL** (representa los números de coma flotante de precisión doble del IEEE)
Char	**VisibleString SIZE(1)**
Boolean	**BOOLEAN**
Octet	**BIT STRING SIZE(8)**
Any	**ANY**

B.12 Correspondencia para tipos de datos construidos

B.12.1 Correspondencia para tipos de estructura

Los tipos estructurados definidos en el UIT-ODL como struct se representan en ASN.1 como una SEQUENCE. Los miembros del mismo tipo en la estructura pueden ser señalados como una lista como en el UIT-ODL. Esto no es posible en una SEQUENCE del ASN.1. Por consiguiente, cada miembro debe ser especificado separadamente.

---

[10] Como una alternativa, los tipos de datos pueden corresponder con tipos de datos del SDL. Esta correspondencia no está contenida en este anexo.

[11] IEEE Standard for Binary Floating Point Arithmetic, ANSI/IEEE Std. 754-1985.

Ejemplo:

```
struct S {    /* ITU-ODL */
  long mem1;
  short mem2;
};
S ::= SEQUENCE {    /* SDL + ASN.1 */
   mem1 ODL-long,
   mem2 ODL-short
};
```

Hay operadores para generar un caso del tipo definido anteriormente y acceder a sus miembros y modificarlos. Estos operadores se explican en [7].

```
Dcl
     s S,
     i ODL_long,
     j ODL_short;
...
     task
          s := (. 2, 1 .),
          i := s!mem1,
          j := s!mem2,
          s!mem2 := 5;
```

B.12.2 Correspondencia para unión

union del UIT-ODL corresponde con SEQUENCE de ASN.1 que contiene un rótulo que indica qué tipo significa y un CHOICE para todos los tipos definidos en union del UIT-ODL.

Ejemplo:

```
union u switch (short) { /* ITU-ODL */
  case 1: long l;
  case 2: short s;
  default: float f;
};


U ::= SEQUENCE {    /* SDL + ASN.1 */
   tag ODL_short,
   union CHOICE {
      l ODL-long,
      s ODL-short,
      f ODL-float
   }
};
```

El discriminador es siempre un rótulo denominado y CHOICE es una unión denominada.

```
dcl
   u U,
   ll ODL_long,
   ss ODL_short,
   ff ODL_float;
...
/* making a call to get a value for u */
...
decision (u!tag);
   (1): task
             ll := u!union!l;
   (2): task
             ss := u!union!s;
```

```
   else: task
            ff := u!union!f;
enddecision;
```

B.12.3 Correspondencia para enumeración

Las enumeraciones del UIT-ODL (enum) se representan en ASN.1 como ENUMERATED. No se debe cambiar el orden de elementos en la enumeración. Una orden explícita en la enumeración ASN.1 ha de ser destacada.

Para los tipos de enumeración hay operadores relacionales y operadores para obtener el primer y último enumerador y el predecesor y sucesor de cada enumerador. Estos operadores se indican en [7].

B.12.4 Correspondencia para tipos de secuencia

Una SEQUENCE UIT-ODL corresponde con un tipo SEQUENCE OF de la ASN.1, con límite de tamaño o no dependiente de la descripción de SEQUENCE del UIT-ODL.

Ejemplo:

```
typedef sequence <short, 30> seq;  /* ITU-ODL */
SEQ ::= SEQUENCE SIZE(30) OF ODL-short;    /*SDL + ASN.1 */
```

En [7] se definen los operadores para acceder a los miembros de SEQUENCE OF y modificarlos.

B.12.5 Correspondencia para cadenas

Una STRING del UIT-ODL se representa en la ASN.1 como una VisibleString, con límite de tamaño o no dependiendo de la descripción STRING del UIT-ODL.

El tipo ASN.1 obtiene el nombre ODL_string.

B.12.6 Correspondencia para conjuntos

Un array del UIT-ODL corresponde con un tipo ASN.1 SEQUENCE OF para cada dimensión definida con límite de tamaño.

Ejemplo:

```
<type> a[8][7][67];       /* ITU-ODL */
```

Si es un miembro de una estructura:

```
a SEQUENCE SIZE(8) OF SEQUENCE SIZE(7) OF SEQUENCE
                               SIZE(67) OF <type>;
```

Si es una especificación de tipo:

```
A ::= SEQUENCE SIZE(8) OF SEQUENCE SIZE(7) OF SEQUENCE
                                      SIZE(67) OF <type>;
```

dependiendo de que el array sea un miembro de una struct o una especificación de tipo.

B.13 Correspondencia para excepciones

Para cada EXCEPTION definida en una especificación UIT-ODL se define un tipo SEQUENCE de la ASN.1 en package <name>_interface con el nombre global de la excepción. Esta SEQUENCE contiene los parámetros de la excepción. Además, hay una definición de señal pRAISE_<name>, donde name es el nombre global de la excepción. Esa señal transporta el tipo de datos definido para los parámetros de excepción. El servidor que desea producir una excepción envía sencillamente la correspondiente señal pRAISE al cliente. El cliente puede obtener los valores de los parámetros de excepción a partir de la señal.

Ejemplo:

```
exception ex{                    /*ITU-ODL*/
     long p1;
};

use idltypes ;                   /*SDL,ASN.1*/
package name_interface ;
     /* name shall be replaced by the name of the ODL specification */
     Ex ::= SEQUENCE {
          ODL_long
     };
     signal pRAISE_ex( Ex );
endpackage;
```

B.14 Definiciones adicionales

Se proponen algunas definiciones adicionales, que no son necesarias pero están destinadas a facilitar el tratamiento de la especificación SDL generada.

Listas de señales

Para cada plantilla de interfaz operacional se define una lista de señales (signallist) en package <name>_interface que contiene las señales (pCALL) para las operaciones que pueden ser invocadas en esa interfaz. Una segunda lista de señales contiene todas las terminaciones posibles, lo que significa todas las señales pREPLY y pRAISE para las posibles terminaciones de las operaciones. Las dos listas de señales se denominan <name>_INVOCATIONS y <name>_TERMINATIONS, donde <name> es el nombre global de la plantilla de interfaz.

Compuertas

Los tipos de proceso generados para plantillas de interfaz operacional contienen dos definiciones de compuerta (gate), una compuerta de salida con la lista de señales TERMINATIONS y una compuerta de entrada con la lista de señales INVOCATIONS. Las compuertas se denominan <name>_INVOCATIONS y <name>_TERMINATIONS, donde <name> es el nombre de la plantilla de interfaz.

ANEXO C

Correspondencia con C++

C.1 Motivación

El UIT-ODL es un superconjunto estricto del ODP-IDL [8]. Todos los aspectos relativos a la comunicación entre los componentes de un sistema se describen utilizando descripciones de interfaces del ODP-IDL. Para poder utilizar los productos CORBA existentes como una plataforma para implementar una especificación del UIT-ODL, la correspondencia para la parte del ODP-IDL se adopta del OMG [1]. El propósito es que los compiladores del ODP-IDL específicos de ORB existentes puedan ser utilizados para hacer corresponder la parte del ODP-IDL de una especificación del UIT-ODL con C++.

Además, el componente o la información de componentes se debe reflejar también en el lenguaje de realización. Esta información se puede considerar como información de diseño. No es un requisito que corresponda con el lenguaje de realización. La aplicación funcionará incluso si sólo corresponde la parte ODP-IDL (sólo las interfaces son de importancia para la comunicación). Sin embargo, el UIT-ODL se debe considerar como un lenguaje de diseño computacional y la información contenida

en él debe facilitar la programación de aplicaciones distribuidas. Por tanto, en esta Recomendación se propone una correspondencia de lenguajes del UIT-ODL con C++ [10].

C.2 Requisitos básicos

Como el UIT-ODL es un superconjunto de ODP-IDL, las correspondencias de lenguajes contenidas en [1] sirven como orientación para definir una correspondencia de lenguaje para UIT-ODL. Los siguientes conceptos tienen que corresponder con respecto a "C++ Language Stub Mapping" que se presenta aquí.

Construcciones del UIT-ODL que existen también en el ODP-IDL:

- todos los tipos de datos básicos del UIT-ODL;
- todos los tipos de datos construidos del UIT-ODL;
- las constantes definidas en el UIT-ODL;
- referencias a interfaces definidas en el UIT-ODL;
- invocación de operaciones, incluido el paso de parámetros y la recepción de resultados;
- excepciones, incluidas las que surgen cuando una operación produce una excepción y cómo se accede a los parámetros de excepción;
- acceso a atributos.

Otras construcciones específicas del UIT-ODL:

- plantillas de objeto (CO);
- plantillas de grupo;
- especificación de comportamiento;
- firmas de trenes (actualmente en este documento no se define una correspondencia de trenes).

C.3 Estructura

Un fichero UIT-ODL corresponde con el encabezamiento C++ y (posiblemente) ficheros de realización. No hay una estructura prescrita de los ficheros generados.

C.4 Nombres con ámbitos de aplicación fijado

Las reglas para nombres globales en el UIT-ODL son las mismas que en el ODP-IDL. Sólo hay tres entidades más que forman un ámbito de nombres: plantillas de interfaz de trenes, plantillas CO y plantillas de grupo. Todas las entidades pueden ser accedidas en C++ utilizando el operador de ámbito "::". No se necesitan otras reglas.

C.5 Correspondencia de módulos

Se supone que los módulos correspondan con espacios de nombre (namespace) en C++. Como un espacio de nombre puede ser reabierto, no hay problema en utilizar un compilador ODP-IDL específico de ORB y generar ficheros adicionales para las plantillas de grupo y de objeto de UIT-ODL.

Si debido al compilador o a restricciones de ORB no es posible una correspondencia de los espacios de nombre, el problema de la reapertura se debe resolver de otra manera. Esto se podrá hacer colocando las definiciones de plantilla de grupo y de objeto en una clase adicional ::ODL. En ese caso, los nombres globales que hacen referencia a CO o a grupos deben ser prefijados con ese nombre de clase.

C.6 Correspondencia de plantillas de interfaz, operaciones, flujos y atributos

Las plantillas de interfaz operacional, operaciones y atributos corresponden con clases de C++ de la misma manera que se describe en la correspondencia del ODP-IDL con C++ [1].

C.6.1 Cláusulas de comportamiento y utilización

Las descripciones de comportamiento y las descripciones de utilización figuran en texto claro solamente. Corresponderán con comentarios. Si se utiliza un compilador ODP-IDL existente, no hay correspondencia.

C.6.2 Flujos

Actualmente no disponible.

C.6.3 Herencia de plantilla de interfaz

La herencia de plantilla de interfaz corresponde de la misma manera descrita para la correspondencia del ODP-IDL con C++.

C.7 Correspondencia para plantillas de objeto

Las plantillas de objeto corresponden con clases de C++ (clases de CO). La principal funcionalidad proporcionada por las clases generadas es que contienen métodos para creación y supresión de las interfaces soportadas de la definición de CO del UIT-ODL.

Se puede proporcionar funcionalidad adicional, tal como punto de comprobación o migración, pero no se prescribe y depende de la plataforma en la cual debe funcionar la aplicación. Por ese motivo pudiera ser útil tener una clase básica común similar a ::CORBA::Object, y dejar que todas las clases generadas hereden de ella.

C.7.1 Especificación de interfaces requeridas

Las plantillas de interfaces requeridas no tienen correspondencia. Hay la correspondencia de cliente CORBA para la plantilla de interfaz solamente.

Sin embargo, se necesita la información de interfaz requerida si se debe construir una realización de servidor mínima (con respecto al cómputo de clases generadas). En ese caso, sólo hay que generar la parte cliente de la correspondencia CORBA para aquellas interfaces que son requeridas y la parte servidor para las interfaces que son soportadas.

C.7.2 Especificación de interfaces soportadas

La correspondencia de las plantillas de interfaces anunciadas en la lista de interfaces soportadas en una CO arbitraria es la siguiente:

- Hay una clase de C++ generada que implementa la clase generada de acuerdo con la correspondencia del ODP-IDL. Esta clase contiene implementaciones para todas las operaciones de la interfaz. El comportamiento es delegar una llamada entrante a una clase de realización. El programador no tiene que tocar ninguna de estas clases de delegación.
- La clase de implementación es generada para todas las interfaces enumeradas en cualquier parte como soportadas. La clase de realización contiene todas las operaciones como métodos virtuales puros. El programador es responsable de proporcionar realizaciones de estas clases.
- Hay una posibilidad de hacer una referencia a un caso de una clase de implementación conocida del caso de clase de delegación. Esto puede ser realizado en el constructor de la clase de delegación o mediante method _ _set_implementation.

CO puede crear casos de sus plantillas de interfaz soportadas de la siguiente manera:

1) Crear un caso de la clase de delegación.

2) Obtener una referencia a un caso de la clase de realización (puede existir o ser creada).

3) Hacer la referencia al caso de clase de realización conocido del caso de clase de delegación.

C.7.3 Especificación de inicialización

La interfaz inicial corresponde en el mismo sentido que las interfaces soportadas.

C.7.4 Herencia

La herencia de plantilla de objeto se realiza como herencia de C++ entre las clases de C++ generadas.

C.7.5 Ejemplo

Supóngase que la siguiente definición del UIT-ODL tiene que corresponder con C++.

```
interface i3;        /*ITU-ODL*/
interface i1;
interface i2;
CO o {
        behaviour
                "use i to initialize object" ;
        requires
                i1;
        supports
                i2;
        initial
                i3;
} /*end object */;
```

La figura 7 muestra las clases generadas y sus relaciones. Las flechas representan la relación de herencia mientras que las flechas de trazo interrumpido simbolizan delegación. Las clases con sufijo _impl son generadas por la mayoría de los compiladores ODP-IDL para la realización de interfaces. Las clases con sufijo _dpe son las clases de delegación y las clases con sufijo _IMPL son las clases de realización presentadas anteriormente. La clase dpe es una clase básica para todas las clases de delegación y la clase dpe_o es una clase básica para todas las clases CO. No son prescritas por esta correspondencia.

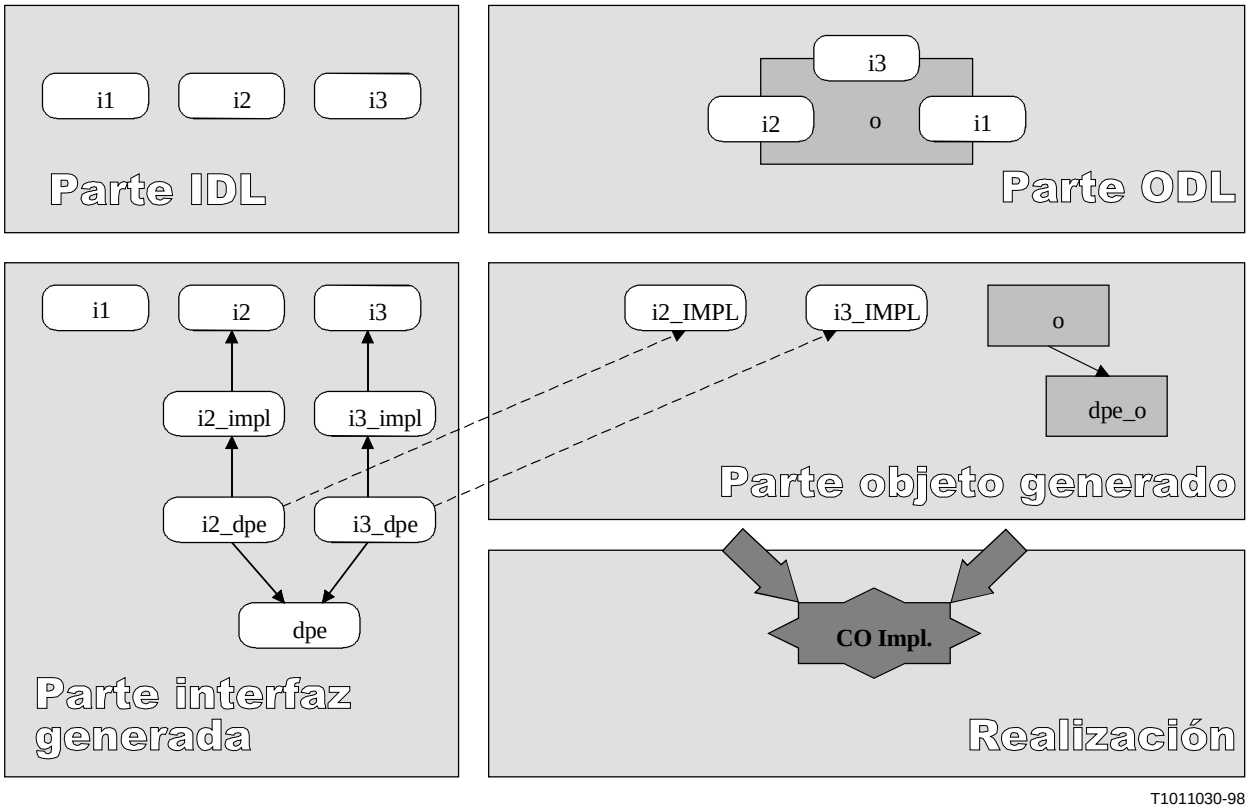

Figura 7/Z.130 – Correspondencia de plantillas de objetos y plantillas de interfaces soportadas

El programador es libre de elegir cómo estructurar la aplicación. Sólo debe proporcionar implementaciones para las clases de realización y posiblemente para las clases CO.

La figura 8 muestra algunos ejemplos de cómo pudiera estructurarse la realización.

- Hay la posibilidad de tener una realización para cada clase de realización de interfaz. Esto ocurre normalmente cuando hay mucha información de estado relacionada con la interfaz. El intercambio de sincronización y de información es tratado por la realización de clase de CO.
- Es posible tener una realización para varias clases de realización de interfaz.
- Si no hay información de estado relacionada con las interfaces, la realización podrá hacerse directamente en la realización de clase de CO.

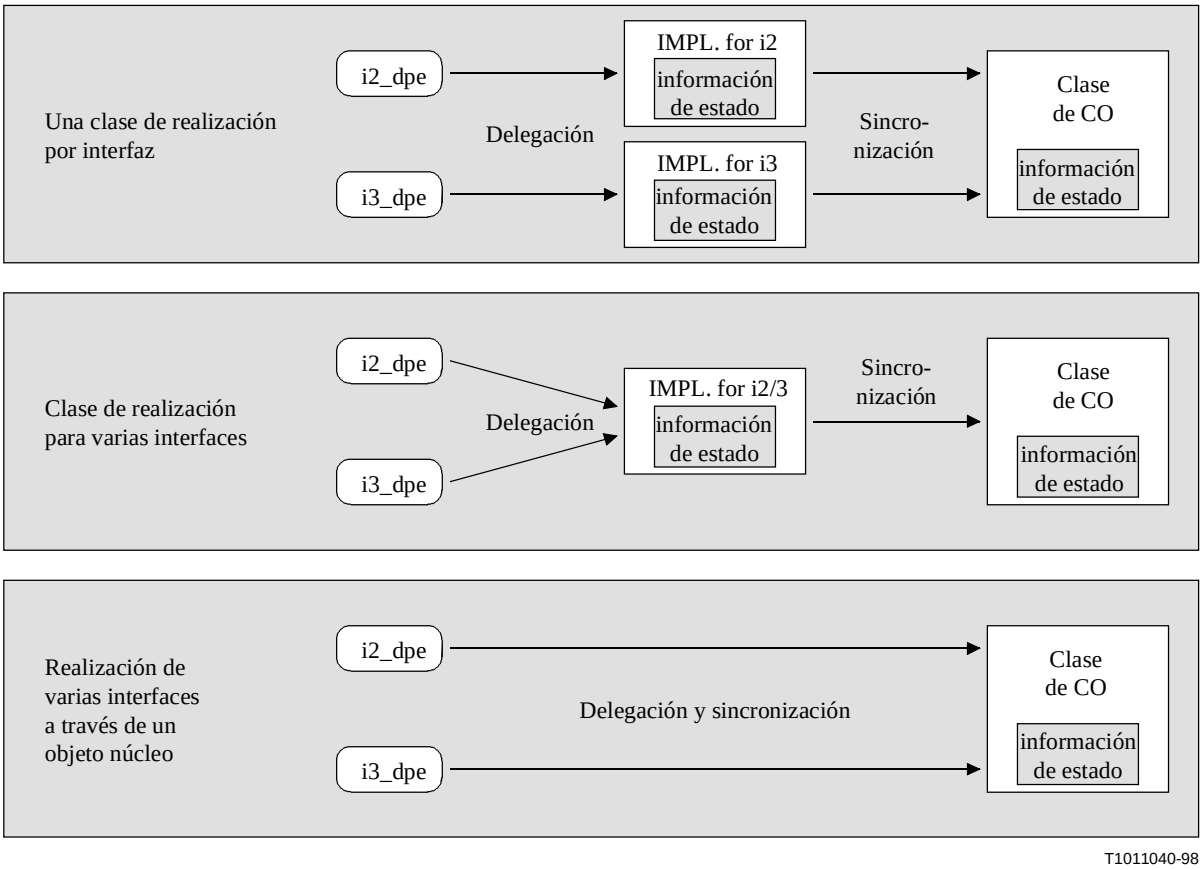

Figura 8/Z.130 – Diferentes estrategias de realización

C.8 Correspondencia para plantillas de grupo

Las plantillas de grupos son conjuntos de CO y grupos computacionales que pueden ser aglomerados juntos por cualquier razón. El motivo para agruparlos está contenido en la especificación de predicado de grupo. Como la especificación de predicado puede variar, no hay una correspondencia prescrita para grupos.

Sin embargo, si la finalidad del grupo es agrupar a sus miembros para la realización, el grupo puede corresponder con una definición de clase de la misma manera que CO. Los contratos del grupo (las interfaces soportadas y requeridas) tienen la misma correspondencia que los objetos.

C.9 Correspondencia para constantes

Las constantes corresponden de la misma manera descrita en la correspondencia del ODP-IDL con C++.

C.10 Correspondencia para tipos de datos básicos

Los tipos de datos básicos corresponden de la misma manera descrita en la correspondencia del ODP-IDL con C++.

C.11 Correspondencia para tipos de datos construidos

Los tipos de datos construidos corresponden de la misma manera descrita en la correspondencia del ODP-IDL con C++.

C.12 Correspondencia para excepciones

Las excepciones corresponden de la misma manera descrita en la correspondencia del ODP-IDL con C++.

APÉNDICE I

Calidad de servicio

I.1 Motivación

La especificación de los aspectos funcionales de una operación o flujo puede tener que ser aumentada con una especificación de la "norma del servicio" requerida. Hay varias maneras en que se pudiera indicar la información sobre dicha calidad de servicio. Por ejemplo, puede haber:

- Enunciados de capacidades obligatorias. Por ejemplo, un flujo debe soportar conexiones de una determinada anchura de banda (no más y no menos), si no, no se ofrece.
- Enunciados de expectación. Por ejemplo, una operación debe responder dentro de un determinado tiempo para la mayoría de los casos.
- Enunciados de soporte. Por ejemplo, cuando se vincula a una interfaz de trenes, la calidad de servicio ha de ser negociada utilizando, digamos, la anchura de banda y la fluctuación de fase.

Con respecto a lo que es "específicamente el asunto de la información de calidad de servicio" en este documento sólo se ofrece una vaga definición. Las calidades de servicio asociadas con una operación o flujo son las constricciones de temporización, el grado de paralelismo, las garantías de disponibilidad, etc., que han de ser proporcionadas por un objeto. La información de calidad de servicio trata de la provisión del servicio, más bien que del servicio propiamente dicho. Las especificaciones de calidad de servicio permiten enunciados sobre el "nivel de servicio" ofrecido por las construcciones. En general, esta información puede ser proporcionada en el momento de la especificación o dinámicamente por el sistema de gestión responsable de iniciar la creación de objetos. Puede incluso ser alterada durante la vida de la oferta de servicio.

El problema de añadir especificaciones de calidad de servicio al UIT-ODL se puede considerar más bien como un problema de semántica que de sintaxis. Esto se destaca cuando se considera que para una sintaxis dada es probable que las tres interpretaciones anteriores pudieran ser posibles cuando se considera el significado de una simple sintaxis. Por ejemplo, un enunciado de la calidad de servicio asociada con un flujo de anchura de banda BandwidthType = 3 Mbit/s puede significar que la interfaz con ese flujo no puede ser ofrecida, a menos que pueda ser la fuente/sumidero de 3 Mbit/s exactamente, o que típicamente se hagan conexiones al flujo a 3 Mbit/s; pero otros valores pueden ser negociados, o la anchura de banda máxima de una conexión es a 3 Mbit/s, y sólo pueden ser negociados valores más bajos, etc. La semántica de calidad de servicio particular que ha de ser soportada en el UIT-ODL no ha sido aún terminada.

En esta versión del UIT-ODL, la semántica se deja al programador.

NOTA – El trabajo en curso en el UIT-T sobre "Procesamiento distribuido abierto – Modelo de referencia – Calidad de servicio" debe integrarse después de aprobación.

I.2 Sintaxis

El UIT-ODL permite asociar un tipo de calidad de servicio e identificador de variable a cualquier definición de:

- operación; o
- flujo.

No se adscriben valores a las variables de calidad de servicio en el momento de la especificación. En cambio, han de ser asignadas en el momento de la realización. Las semánticas dependen del servicio. Se reconoce que esta capacidad está muy restringida. La semántica de una especificación de calidad de servicio puede ser descrita en la especificación de comportamiento de la interfaz.

A continuación se indica la sintaxis de una especificación de calidad de servicio operacional. Obsérvese que los parámetros de calidad de servicio se añaden después de la palabra clave "with". Obsérvese que el identificador de la variable definida debe ser único dentro de la interfaz:

```
<op_sig_defn>        ::=  { <announcement> | <interrogation> }
                          ["with" <QoS_attribute>]
<QoS_attribute>      ::=  <QoS_attr_type> <QoS_attr_name>
<QoS_attr_type>      ::=  <simple_type_spec>
<QoS_attr_name>      ::=  <simple_declarator>
```

La sintaxis para vincular una especificación de calidad de servicio a un flujo es la siguiente:

```
<stream_flow_defn>   ::=  <flow_direction> <flow_type>
                          <identifier> ["with" <QoS_attribute>]
```

I.3 Ejemplo

A continuación se da un ejemplo de una especificación de calidad de servicio de flujo. Los parámetros de calidad de servicio aparecen después de la palabra clave "with". La calidad de servicio se ofrece utilizando un caso de VideoQoS. Un caso de VideoQoS se representa como una flotación, pero dependiendo del valor de Guarantee (sea estadístico o determinístico), la flotación se ha de interpretar como una velocidad de trama "media" o "máxima".

```
struct VideoQoS {
        union Throughput switch (Guarantee) {          /* in frames/s */
              case Statistical:        float  mean;
              case Deterministic:      float  peak;
        };
}; // end of VideoQoS

interface I3 {
       ...
       sink VideoFlowType display with VideoQoS requiredQoS;
       ...
}; // end of I3
```

El UIT-ODL no especifica parámetros normalizados de calidad de servicio para las operaciones o flujos.

I.4 Correspondencia con el SDL

La correspondencia para una especificación de calidad de servicio con el SDL es similar a la correspondencia de atributos de interfaz de lectura solamente. Para cada parámetro de calidad de servicio hay una variable local en el tipo de proceso generado para el lado servidor junto con un

procedimiento de obtención para obtener su valor. Los convenios de denominación son iguales que los descritos en B.6.

APÉNDICE II

Comparación de UIT-ODL con ODP-IDL y TINA-ODL

II.1 Objetivo del UIT-ODL y objetivo del ODP-IDL

Los objetivos del UIT-ODL son los siguientes:

- Lenguaje para especificación de aplicación (en el momento del desarrollo).
- Lenguaje para reutilización de aplicación (en el momento de desarrollo).
- Lenguaje que soporta ejecución e interacción de aplicaciones (en el momento de ejecución).

El ODP-IDL comparte la mayoría de estos objetivos (soporte para especificación de aplicaciones, reutilización de aplicaciones e interacción de aplicaciones).

II.2 Modelo de objeto

El modelo de objeto soportado por el UIT-ODL amplía el modelo de objeto del OMG [1] de las siguientes maneras:

En el nivel de objeto, el UIT-ODL ofrece soporte para la definición de:

- comportamiento de objeto;
- objetos con múltiples interfaces;
- grupos de objetos.

En el nivel de interfaz, el UIT-ODL ofrece soporte para la definición de:

- interfaces de trenes;
- comportamiento de interfaz.

En el nivel de operaciones y flujos, el UIT-ODL ofrece soporte para la definición de:

- firmas de trenes (flujos).

II.3 Sintaxis del UIT-ODL y sintaxis del ODP-IDL

El UIT-ODL en su versión actual es un superconjunto de ODP-IDL. Esto entraña que es posible utilizar especificaciones del ODP-IDL como parte de especificaciones del UIT-ODL (como declaración de interfaz operacional). En consecuencia, la sintaxis definida en el UIT-ODL para la declaración de interfaz operacional abarca y soporta todas las reglas definidas para el ODP-IDL [8].

II.3.1 Sintaxis general

Para una especificación del UIT-ODL, generalmente:

- Las estructuras añadidas al ODP-IDL son las declaraciones de objeto (<object_dcl>), y las declaraciones de grupo de objetos (<group_dcl>).
- Las estructuras compartidas con el ODP-IDL son la definición soportadora (<supporting_def>), y la declaración de módulo (<module>).
- La estructura modificada del ODP-IDL es la declaración de interfaz (<interface_dcl>).

II.3.2 Sintaxis de interfaz

En la sintaxis para declaración de interfaz:

- Las estructuras añadidas al ODP-IDL son la declaración (facultativa) de comportamiento de interfaz (<interf_behaviour_spec>), la especificación de utilización de interfaz (<interf_usage_spec>) y la declaración de firma de trenes (flujos) (<stream_sig_defns>).
- Las estructuras compartidas con el ODP-IDL son la declaración hacia adelante de interfaces (<interface_forward_dcl>), la palabra clave de encabezamiento y nombre de interfaz ("interface" e <identifier>), y la especificación de herencia de interfaz (<interf_inheritance_spec>).
- La estructura modificada del ODP-IDL es la declaración de firma de operaciones (<operation_sig_defns>).

II.3.3 Sintaxis de operaciones

En la sintaxis para declaración de operaciones:

- Las estructuras compartidas con el ODP-IDL son la declaración de anuncio (<announcement>), y la declaración de interrogación (<interrogation>).

Obsérvese que todas las adiciones al ODP-IDL son facultativas.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Transmisiones de señales radiofónicas, de televisión y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos y comunicación entre sistemas abiertos
Serie Y	Infraestructura mundial de la información
Serie Z	**Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación**