UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T X.780

(01/2001)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE X: REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

Gestión de interconexión de sistemas abiertos – Funciones de gestión y funciones de arquitectura de gestión distribuida abierta

Directrices de la RGT para la definición de objetos gestionados mediante arquitectura de intermediario de petición de objeto común

Recomendación UIT-T X.780

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES UIT-T DE LA SERIE X

REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

REDES PÚBLICAS DE DATOS	
Servicios y facilidades	X.1–X.19
Interfaces	X.20–X.49
Transmisión, señalización y conmutación	X.50–X.89
Aspectos de redes	X.90–X.149
Mantenimiento	X.150–X.179
Disposiciones administrativas	X.180–X.199
INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Modelo y notación	X.200–X.209
Definiciones de los servicios	X.210–X.219
Especificaciones de los protocolos en modo conexión	X.220–X.229
Especificaciones de los protocolos en modo sin conexión	X.230–X.239
Formularios para declaraciones de conformidad de implementación de protocolo	X.240–X.259
Identificación de protocolos	X.260–X.269
Protocolos de seguridad	X.270–X.279
Objetos gestionados de capa	X.280–X.289
Pruebas de conformidad	X.290–X.299
INTERFUNCIONAMIENTO ENTRE REDES	
Generalidades	X.300–X.349
Sistemas de transmisión de datos por satélite	X.350–X.369
Redes basadas en el protocolo Internet	X.370–X.399
SISTEMAS DE TRATAMIENTO DE MENSAJES	X.400–X.499
DIRECTORIO	X.500–X.599
GESTIÓN DE REDES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS Y ASPECTOS DE SISTEMAS	
Gestión de redes	X.600–X.629
Eficacia	X.630–X.639
Calidad de servicio	X.640–X.649
Denominación, direccionamiento y registro	X.650–X.679
Notación de sintaxis abstracta uno	X.680–X.699
GESTIÓN DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Marco y arquitectura de la gestión de sistemas	X.700–X.709
Servicio y protocolo de comunicación de gestión	X.710–X.719
Estructura de la información de gestión	X.720–X.729
Funciones de gestión y funciones de arquitectura de gestión distribuida abierta	**X.730–X.799**
SEGURIDAD	X.800–X.849
APLICACIONES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Compromiso, concurrencia y recuperación	X.850–X.859
Procesamiento de transacciones	X.860–X.879
Operaciones a distancia	X.880–X.899
PROCESAMIENTO DISTRIBUIDO ABIERTO	X.900–X.999

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

Recomendación UIT-T X.780

Directrices de la RGT para la definición de objetos gestionados mediante arquitectura de intermediario de petición de objeto común

Resumen

Esta Recomendación especifica directrices para la definición de interfaces basadas en CORBA aplicables a los objetos de soporte lógico que representan a recursos gestionables por una red de gestión de las telecomunicaciones (RGT). Trata de las directrices para el modelado de la información, reglas para la traslación de modelos desde las GDMO, así como convenios de estilo del IDL. También proporciona un módulo IDL para la definición de tipos de datos, superclases y notificaciones a utilizar en las especificaciones del modelo de información basado en CORBA.

Orígenes

La Recomendación UIT-T X.780, preparada por la Comisión de Estudio 4 (2001-2004) del UIT-T, fue aprobada por el procedimiento de la Resolución 1 de la AMNT el 19 de enero de 2001.

Palabras clave

Arquitectura de intermediario de petición de objeto común (CORBA), directrices para la definición de objetos gestionados (GDMO), interfaces de la RGT, lenguaje de definición de interfaz (IDL), notación de sintaxis abstracta uno (ASN.1), objetos gestionados, procesos distribuidos.

PREFACIO

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

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

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

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

NOTA

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

PROPIEDAD INTELECTUAL

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

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

ÍNDICE

		Página
1	Alcance	1
1.1	Finalidad	1
1.2	Aplicaciones	2
1.3	Guía	3
1.4	Convenios	4
1.5	Compilación del IDL	4
2	Referencias	5
2.1	Referencias normativas	5
3	Definiciones y abreviaturas	6
3.1	Definiciones de UIT-T X.701	6
3.2	Definiciones de UIT-T X.703	6
3.3	Abreviaturas	6
4	Objetivos del modelado y requisitos CORBA	7
4.1	Objetivos	7
4.1.1	Interoperabilidad de aplicaciones	8
4.1.2	Uso común de servicios de objetos comunes de CORBA	8
4.1.3	Transparencia del modelo de información	8
4.2	Entidades	8
4.2.1	Granularidad de acceso	8
4.3	Principios de contenencia y denominación	9
4.3.1	Denominación	10
4.3.2	Identificación de entidad	10
4.4	Clases de objetos gestionados	11
4.5	Lotes	11
4.6	Atributos	11
4.6.1	Obtener (GET) y asignar (SET)	11
4.6.2	Atributo genérico Get	11
4.6.3	Atributos con el valor asignado (Set)	11
4.7	Creación y supresión de objetos gestionados	12
4.7.1	Creación	12
4.7.2	Supresión	13
4.8	Herencia	13
5	El módulo IDL del modelo de objeto	13
5.1	La interfaz de objeto gestionado Base (Tope)	14
5.1.1	La operación *nameGet()*	15

Página

5.1.2 La operación *objectClassGet()* 15
5.1.3 La operación *packagesGet()* 16
5.1.4 La operación *creationSourceGet()* 16
5.1.5 La operación *deletePolicyGet()* 16
5.1.6 La operación *attributesGet()* 16
5.1.7 La operación *destroy()* 17
5.2 La factoría de objetos gestionados 17
5.3 La interfaz notificaciones 18
5.4 Las definiciones de tipos de datos 20
5.5 Excepciones 20
5.5.1 La excepción ApplicationError 20
5.5.2 La excepción *CreateError* 21
5.5.3 La excepción *DeleteError* 22
5.6 Definiciones de macros 22
5.7 Las definiciones de constantes 23
6 Directrices para el modelado de información 23
6.1 Módulos 23
6.2 Interfaces 23
6.3 Atributos 24
6.3.1 Atributos legibles 25
6.3.2 Atributos asignables 25
6.3.3 Atributos valores de conjunto 25
6.3.4 Excepciones 25
6.3.5 Atributos estándar 26
6.4 Acciones 27
6.5 Notificaciones 27
6.6 Lotes condicionales 28
6.7 Comportamiento 28
6.8 Información de vinculación de nombres 29
6.9 Factorías 31
6.9.1 Operaciones de creación 31
6.9.2 Buscador de factoría 33
6.10 Tipos de valores de clase de objeto gestionado 33
6.11 Constantes 35
6.12 Registro 36
6.13 Versión de las especificaciones del IDL de CORBA 36
7 Traslación desde GDMO 37

		Página
7.1	Clases de objetos gestionados	37
7.2	Lotes	38
7.3	Atributos	39
7.4	Grupos de atributos	40
7.5	Acciones	40
7.6	Notificaciones	40
7.7	Comportamientos	41
7.8	Vinculaciones de nombres	41
7.9	Parámetros	42
7.9.1	Información de acción (ACTION-INFO) y respuesta a acción (ACTION-REPLY)	42
7.9.2	Información de evento (EVENT-INFO) y respuesta a evento (EVENT-REPLY)	42
7.9.3	Palabra clave de contexto (Context-Keyword)	44
7.9.4	Error específico (SPECIFIC-ERROR)	44
7.10	Tipos de datos ASN.1	45
7.10.1	Tipos básicos	45
7.10.2	Secuencia	45
7.10.3	Secuencia de	46
7.10.4	Conjunto de	46
7.10.5	Elección	46
7.10.6	Identificador de objeto (OID)	46
7.10.7	Ejemplar de objeto	46
7.10.8	Cadena de bits	46
8	Modismos de estilo para la especificación del IDL de CORBA	48
8.1	Utilizar sangrado consistente	49
8.2	Uso consistente del tamaño de letra en los identificadores	49
8.3	Seguir la técnica de la gestión conjunta entre dominios (JIDM) en las IMPORTACIONES	50
8.4	Seguir la técnica JIDM para opcional (OPTIONAL) y elección (CHOICE)	50
8.5	Utilizar un sufijo de tipo consecuente	50
8.6	Utilizar un sufijo consecuente para los tipos secuencia	51
8.7	Utilizar un sufijo consecuente para los tipos conjunto	51
8.8	Utilizar un sufijo consecuente para los tipos opcionales	51
8.9	Disponer los parámetros de operación de una forma consecuente	51
8.10	Suponga espacios de identificación no globales	51
8.11	Definiciones a nivel de módulo	51
8.12	Utilizar las excepciones y los códigos de retorno	51

Página

8.13 Operaciones explícitas versus operaciones implícitas 51
8.14 No cree un número grande de excepciones 51
9 Cumplimiento y conformidad 51
9.1 Cumplimiento de los documentos de normas 52
9.2 Conformidad del sistema 52
9.3 Directrices para la declaración de conformidad 52
Anexo A – El módulo modelo de objeto del IDL de CORBA 53
Anexo B – Definiciones de constantes de gestión de red 74
Apéndice I – Bibliografía 77

Recomendación UIT-T X.780

Directrices de la RGT para la definición de objetos gestionados mediante arquitectura de intermediario de petición de objeto común

1 Alcance

La arquitectura de la red de gestión de las telecomunicaciones (RGT) definida en UIT-T M.3010, introduce conceptos sobre procesos distribuidos e incluye el uso de diversos protocolos de gestión. Las especificaciones iniciales de la interfaz de la RGT para las interfaces intra RGT y entre RGT se desarrollaron utilizando la notación de las directrices para la definición de objetos gestionados (GDMO, *guidelines for the definition of managed objects*) basada en la gestión de sistemas OSI haciendo uso del protocolo común de información de gestión (CMIP, *common management information protocol*). La interfaz entre RGT (X) incluía tanto el CMIP como el GIOP/IIOP de CORBA como posibles elecciones en la capa aplicación.

Se considera a CORBA, una tecnología de procesos distribuidos, como utilizable en arquitecturas de comunicación de la RGT, principalmente por su aceptación por parte de la industria de la tecnología de la información. Se espera que esta aceptación incremente la disponibilidad de interfaces basadas en CORBA debido a la existencia de herramientas de desarrollo más adecuadas y a la mayor difusión del conocimiento para el desarrollo de interfaces basadas en CORBA. Esta tecnología, desarrollada por el grupo de gestión de objetos (OMG, *object management group*) está también siendo tenida en cuenta por múltiples industrias. Las especificaciones que utilizan estas tecnologías proporcionan soporte a interfaces de programación de aplicaciones (API, *application programming interfaces*) estándar, así como vinculaciones de lenguaje para los lenguajes de programación, facilitando también la portabilidad de los soportes lógicos. Las soluciones de interoperabilidad ofrecidas por el intermediario de petición de objetos se combinan con la interoperabilidad de direcciones del protocolo Inter-ORB entre el cliente y el servidor. Mientras que el CMIP y los modelos de información proporcionan soluciones para la interoperabilidad entre los sistemas gestor y agente, CORBA define interacciones entre objetos pudiendo estar los objetos distribuidos.

1.1 Finalidad

Diversos grupos están desarrollando especificaciones para le gestión de red que utilizan las técnicas de modelado de CORBA, con la notación del IDL, junto con servicios CORBA. La intención de la presente Recomendación es la definición de directrices adecuadas para ser utilizadas en la especificación de interfaces interoperativas basadas en CORBA para la gestión de red. Los requisitos exigidos de estas interfaces son distintos de los de aquellos utilizados "dentro" de una administración, en las interfaces "Q". El alcance de la presente Recomendación trata de todas las interfaces de la RGT en las que se puede utilizar CORBA. Se espera que no sean necesarias todas las capacidades y modelos que aquí se definen en todas las interfaces de la RGT. Esto implica que este marco de trabajo se pueda utilizar para las interfaces entre sistemas de gestión a todos los niveles de abstracción (internos a una administración y entre administraciones) así como entre sistemas de gestión y elementos de red.

La Recomendación UIT-T Q.816 [1] define un conjunto de servicios que son necesarios en las interfaces de la RGT basadas en CORBA. La presente Recomendación define directrices para la especificación de modelos de información escritos en el IDL de CORBA para los que son aplicables los servicios. También da reglas para la traslación de modelos existentes en GDMO a IDL. Finalmente, define varios códigos IDL base para ser utilizados en todos los modelos de información de la RGT basados en CORBA. La unión de la presente Recomendación y de UIT-T Q.816

constituye un *marco* de trabajo para la definición e implementación de interfaces de la RGT basadas en CORBA.

La utilización de un marco de trabajo común para las interfaces de gestión de las telecomunicaciones proporciona diversas ventajas. Algunos ejemplos son: facilitar la reutilización de modelos ya desarrollados para cumplir los requisitos genéricos de las telecomunicaciones; ajustar los servicios CORBA utilizables por la industria de las telecomunicaciones; facilitar la definición de nuevos servicios para la RGT; la reutilización de la semántica del amplio conjunto de modelos existente; y la armonización de la técnica de modelado entre los grupos utilizando una única fuente similar a la de UIT-T X.720, X.721 y X.722 para el CMIP. La reutilización de una técnica común para el modelado de recursos y la reutilización de un modelo de información genérico para una variedad de tecnologías de red y aplicaciones de gestión de red acelerará la introducción de nuevos servicios de red, al tiempo que mantendrá bajos los costes de desarrollo de los sistemas de gestión.

La industria de las telecomunicaciones ha invertido una gran cantidad de tiempo y energía en el desarrollo de modelos de información para el CMIP de gestión de red. Una meta fundamental del marco de trabajo de la RGT de CORBA es la reutilización de estos modelos de información facilitando su traslación al lenguaje de definición de interfaz (IDL, *interface definition language*) de CORBA con pocos cambios en la semántica. Como resultado, se espera que los primeros modelos de información IDL deriven desde los modelos CMIP.

1.2 Aplicaciones

La Recomendación UIT-T M.3020 define tres fases en el desarrollo de una especificación de la RGT. Las tres fases son requisitos, análisis y diseño. La figura 1 presenta este proceso y el objetivo de la presente Recomendación para el desarrollo de especificaciones de interfaces basadas en CORBA relativas a él.

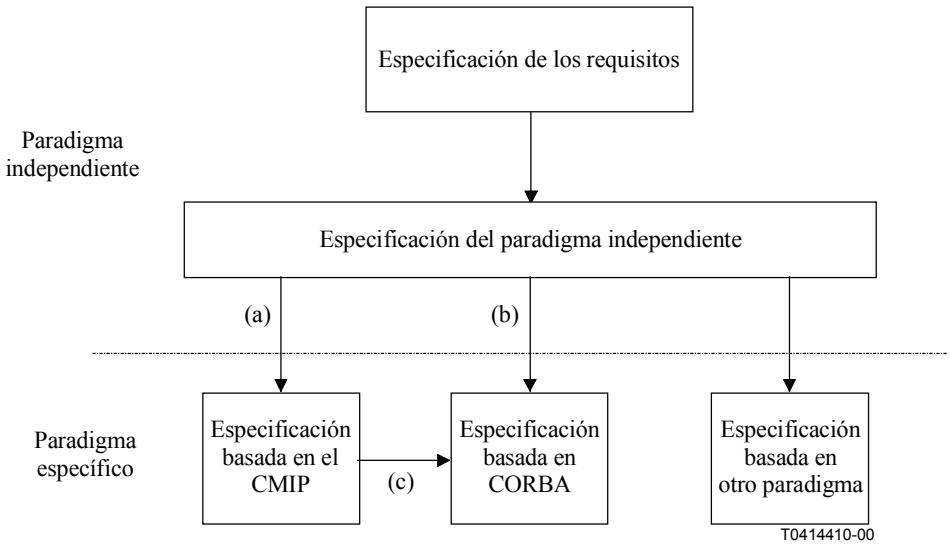

Figura 1/X.780 – Especificación basada en CORBA

Los requisitos y el análisis se especifican utilizando una técnica que no es específica de un paradigma de tecnología. El resultado de las fases de análisis, especificación independiente del paradigma, se utiliza como elemento de partida de la fase de diseño específico del paradigma.

En la fase de diseño, se utilizan las características específicas del paradigma de gestión de red para la definición de los modelos de información. Estas características específicas del paradigma incorporan

el comportamiento (normalmente el lenguaje natural) y signaturas de interfaz formales (por ejemplo, GDMO, IDL).

Las flechas señaladas como (a) y (b) indican que se hace corresponder el resultado del análisis con el modelo basado en GDMO/ASN.1 para utilizar con el CMIP, o con modelos IDL a utilizar con el CORBA/IIOP, respectivamente. No se dispone por el momento de reglas de prescripción para la generación de estos modelos, puede que sea posible el desarrollo de dichas reglas en el futuro en UIT-T M.3020.

La presente Recomendación trata de la reutilización de modelos existentes desarrollados en el paradigma CMIP si se utiliza CORBA/IIOP en lugar de CMIP. Esta Recomendación trata también la flecha marcada (c).

En el desarrollo de la transformación a partir de las definiciones GDMO/ASN.1 a CORBA/IDL son posibles dos métodos:

- En el primero de ellos se traslada cada elemento de la sintaxis a CORBA/IDL utilizando un algoritmo bien especificado o una definición prescriptiva. Este método es el tomado en la gestión conjunta entre dominios (JIDM, *joint inter-domain management*) en el que se puede utilizar una pasarela para soportar la interoperabilidad.
- El segundo método (utilizado en esta Recomendación) no traslada prescriptivamente cada elemento de la sintaxis, sino que los elementos se trasladan de un GDMO existente de modo que preserve la semántica y, al mismo tiempo, utilice las características de CORBA. Esta técnica no se utiliza para el interfuncionamiento con pasarelas, sino para preservar los requisitos y la semántica de los modelos desarrollados para ajustarse al contexto de las telecomunicaciones. Esto se aplica cuando los sistemas gestor y gestionado están diseñados para que se comuniquen por medio de CORBA/IIOP.

Además de las recomendaciones para la traslación desde los modelos de información GDMO aquí definidos, la Recomendación UIT-T Q.816 define recomendaciones para los servicios CORBA a utilizar en la gestión de las redes de telecomunicaciones. En el marco de trabajo son aplicables diversos aspectos de la Q.816, independientemente de cómo se desarrollen las especificaciones basadas en CORBA (es decir, utilizando los caminos marcados como (b) o como (c) en la figura 1).

Además de aprovechar los modelos de información CMIP, otro de los propósitos de las directrices es el aprovechamiento de CORBA. El marco de trabajo potencia las funciones definidas en las especificaciones de CORBA, incluyendo un conjunto de servicios de objetos comunes. Asimismo, estas directrices reutilizan, siempre que son aplicables, técnicas y modelos de diseño CORBA. Finalmente, al tiempo que es importante la reutilización de modelos existentes, también lo es igualmente que el marco de trabajo soporte el desarrollo de nuevos modelos. Estas directrices no precisan que se desarrolle un modelo GDMO previamente al desarrollo de un modelo IDL. En realidad, el desarrollo de un nuevo modelo de información IDL a utilizar en este marco de trabajo es directo y se proporcionan directrices para su realización.

La Recomendación UIT-T M.3120 [11] proporciona una versión del IDL de CORBA del modelo genérico de información de red originalmente definido en UIT-T M.3100. La versión IDL sigue las directrices de modelado de objetos aquí definidas y está diseñada para que haga uso de los servicios de la RGT basados en CORBA definidos en UIT-T Q.816.

1.3 Guía

Esta Recomendación tiene la siguiente estructura:

Cláusula 1 Introducción, guía del documento y actualizaciones.

Cláusula 2 Referencias.

Cláusula 3 Definición de las abreviaturas utilizadas en el resto de la Recomendación.

Cláusula 4 Requisitos para las directrices de modelado de objetos. Éstas son las metas de diseño que deben cumplir las directrices.

Cláusula 5 Descripción del módulo del IDL de CORBA que define las interfaces a utilizar y convertir en subclases en las especificaciones de interfaces de gestión. El IDL real está en los anexos A y B.

Cláusula 6 Directrices para la definición de modelos de información de la RGT basados en CORBA. Estas directrices están diseñadas específicamente para objetos del IDL que utilizan los servicios de la RGT basados en CORBA de UIT-T Q.816.

Cláusula 7 Directrices para la traslación de modelos de información de las GDMO a modelos del IDL adecuados para ser utilizados con los servicios de la RGT basados en CORBA de UIT-T Q.816.

Cláusula 8 Modismos de estilo para las especificaciones de la interfaz de gestión de red basadas en el IDL de CORBA.

Cláusula 9 Directrices de cumplimiento y conformidad.

Anexo A Módulo del IDL para la especificación de directrices de modelado. Este anexo es normativo.

Anexo B IDL adicional que define constantes utilizadas por las directrices de modelado. Este anexo es normativo.

1.4 Convenios

Se siguen algunos convenios en la Recomendación para que el lector se dé cuenta de la finalidad del texto. Mientras que la mayor parte de la Recomendación es normativo, los párrafos que sucintamente indican requisitos obligatorios a cumplir por un sistema de gestión (gestor y/o gestionado) están precedidos por una "R" (*required*, obligatorio) en negrita encerrada entre paréntesis, seguida de un nombre corto que indica el tema del requisito y de un número. Por ejemplo:

(R) EJEMPLO-1 Un requisito obligatorio de ejemplo.

De forma análoga, los requisitos que pueden ser implementados opcionalmente por un sistema de gestión están precedidos por una "O" (*optional*, opcional) en lugar de por una "R". Por ejemplo:

(O) OPCIÓN-1 Un requisito opcional de ejemplo.

Las declaraciones de requisitos se utilizan para la creación de perfiles de cumplimiento y conformidad.

En esta Recomendación se incluyen muchos ejemplos del IDL de CORBA, y en los anexos normativos se incluyen IDL que especifican los tipos de datos y las clases base. Los IDL se presentan con tipo de letra courier de 9 puntos.

```
// Ejemplo IDL
interface foo {
        void operation1 ();
};
```

En la cláusula 1.5 se proporcionan las instrucciones para la extracción de un módulo IDL de una versión electrónica de esta Recomendación y la compilación del mismo.

1.5 Compilación del IDL

Una ventaja de utilizar el IDL para especificar interfaces de gestión de la red es que el IDL puede ser "compilado" en códigos de programación a través de los medios que acompañan a un mediador de petición de objeto (ORB). Esto realmente automatiza el desarrollo de algunos de los códigos necesarios para permitir la interoperación de aplicaciones de gestión de red. Esta Recomendación presenta dos anexos que contienen códigos que los implementadores desearán extraer y compilar.

Los anexos A y B son normativos y han de ser utilizados por los diseñadores que implementan sistemas de conformidad con esta Recomendación. La IDL en esta Recomendación ha sido verificada con dos compiladores para asegurar su exactitud. Se debe utilizar un compilador que soporte la especificación CORBA 2.3.

Los anexos han sido formatados de modo tal que la operación de cortar y pegar en ficheros de texto simple sea sencilla y que entonces puedan ser compilados. A continuación se dan algunas indicaciones para poner esto en práctica.

1) La operación de cortar y pegar parece funcionar mejor con la versión Microsoft® Word® de esta Recomendación. El recorte y pegado del formato de fichero Adobe® Acrobat® parece incluir encabezamientos y pies de páginas que no se pueden compilar.

2) Todo el anexo A, que comienza con la línea "/* Este código IDL ..." hasta el final, debe ser almacenado en un fichero denominado "itut_x780.idl" en un directorio donde será hallado por el compilador IDL.

3) Todo el anexo B, que comienza con la línea "/* Este código IDL ..." hasta el final debe ser almacenado en un fichero denominado "itu_x780Const.idl" en el mismo directorio que el fichero que contiene el anexo A.

4) Los encabezamientos incorporados en estos anexos no necesitan ser extraídos. Han de estar encapsulados en comentarios de IDL y serán ignorados por el compilador.

5) Los comentarios que comienzan con la secuencia especial "/**" son reconocidos por los compiladores que convierten IDL en HTML. Estos comentarios tienen a menudo instrucciones especiales de formatado para esos compiladores. Quienes trabajen con el IDL querrán generar HTML pues los ficheros HTML resultantes tienen vínculos que permiten "navegar" a través de los ficheros.

Los anexos han sido formatados con espacios tabulados a intervalos de 8 espacios y cambios de renglón fijos que permiten a la mayoría de los editores trabajar con el texto.

2 Referencias

2.1 Referencias normativas

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

[1] UIT-T Q.816 (2001), *Servicios RGT basados en la arquitectura de intermediario de petición de objeto común.*

[2] OMG Document formal/99-10-07, *The Common Object Request Broker: Architecture and Specification*, Revision 2.3.1.

[3] OMG Document formal/2000-08-01, *CORBA/TMN Interworking, Version 1,* Edition 4.31.

[4] UIT-T X.701 (1997) | ISO/CEI 10040:1998, *Tecnología de la información – Interconexión de sistemas abiertos – Visión general de la gestión de sistemas.*

[5] UIT-T X.703 (1997) | ISO/CEI 13244:1998, *Tecnología de la información – Arquitectura de gestión distribuida abierta.*

[6] UIT-T X.721 (1992) | ISO/CEI 10165-2:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Definición de la información de gestión*.

[7] UIT-T X.722 (1992) | ISO/CEI 10165-4:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados.*

3 Definiciones y abreviaturas

3.1 Definiciones de UIT-T X.701

Los siguientes términos utilizados en esta Recomendación se definen en la visión general de la gestión de sistemas (UIT-T X.701):

– clase de objeto gestionado;

– gestor;

– agente.

3.2 Definiciones de UIT-T X.703

El siguiente término utilizado en esta Recomendación se define en la arquitectura de gestión distribuida abierta (UIT-T X.703).

– notificación.

3.3 Abreviaturas

En esta Recomendación se utiliza las siguientes siglas:

ASN.1	Notación de sintaxis abstracta uno (*abstract syntax notation No. 1*)
ATM	Modo de transferencia asíncrono (*asynchronous transfer mode*)
CMIP	Protocolo común de información de gestión (*common management information protocol*)
CORBA	Arquitectura de intermediario de petición de objeto común (*common object request broker architecture*)
COS	Servicios de objetos comunes (*common object services*)
DN	Nombre distinguido (*distinguished name*)
EMS	Sistema de gestión de elementos (*element management system*)
GDMO	Directrices para la definición de objetos gestionados (*guidelines for the definition of managed objects*)
GIOP	Protocolo de interoperabilidad general (*general interoperability protocol*)
HTML	Lenguaje de marcaje de hipertexto (*hypertext markup language*)
ID	identificador
IDL	Lenguaje de definición de interfaz (*interface definition language*)
IIOP	Protocolo de interoperabilidad de Internet (*Internet interoperability protocol*)
IOR	Referencia de objeto interoperable (*interoperable object reference*)
JIDM	Gestión conjunta entre dominios (*joint inter-domain management*)

MO	Objeto gestionado (*managed object*)
NE	Elemento de red (*network element*)
NMS	Sistema de gestión de red (*network management system*)
OAM&P	Operación, administración, mantenimiento y suministro (*operations, administration, maintenance and provisioning*)
OID	Identificador de objeto (*object identifier*)
OMG	Grupo de gestión de objetos (*object management group*)
ORB	Mediador de petición de objetos (*object request broker*)
OSI	Interconexión de sistemas abiertos (*open systems interconnection*)
PDU	Unidad de datos de protocolo (*protocol data unit*)
QoS	Calidad de servicio (*quality of service*)
RDN	Nombre distinguido relativo (*relative distinguished name*)
RGT	Red de gestión de las telecomunicaciones
TTP	Punto de terminación de camino (*trail termination point*)
UID	Identificador universal (*universal identifier*)
UIT-T	Unión Internacional de Telecomunicaciones – Sector de Normalización de las Telecomunicaciones
UML	Lenguaje de modelado unificado (*unified modelling language*)
UTC	Código de tiempo universal (*universal time code*)

4 Objetivos del modelado y requisitos CORBA

Esta cláusula describe los objetivos principales para el modelado de recursos de la RGT utilizando CORBA, así como los requisitos que deben cumplir las directrices de modelado para soportar estos objetivos. La cláusula 4.1 presenta los objetivos de las directrices de modelado. Después, las subsiguientes subcláusulas presentan ya la terminología y los requisitos. Los requisitos de la cláusula 4 son los que debe soportar el marco de trabajo. Están basados en las necesidades de la gestión de las telecomunicaciones. Las cláusulas 5, 6, 7 y 8 describen entonces las directrices de modelado que satisfacen a esas necesidades y definen cómo conseguir los requisitos de la cláusula 4 utilizando CORBA de una forma determinada. Las reglas de las cláusulas 5, 6, 7 y 8 sobre la forma de utilización de CORBA se consideran también como requisitos.

4.1 Objetivos

Esta Recomendación especifica directrices para la definición de objetos gestionados CORBA para ser utilizados en interfaces soportadas por sistemas de gestión de las redes de telecomunicaciones y por los elementos de red. Algunos de los objetivos clave de las directrices de modelado son:

- Interoperabilidad de aplicaciones.
- Uso común de servicios de objetos comunes de CORBA.
- Transparencia del modelo de información.

Esta cláusula trata de estos tres objetivos.

4.1.1 Interoperabilidad de aplicaciones

Un objetivo clave de la arquitectura de la RGT, y en particular de la arquitectura de la información, es el de la promoción de un marco de trabajo estándar que proporcione la interoperabilidad y el intercambio de información entre sistemas de un conjunto diverso de suministradores de sistemas de gestión de red. En su capa más inferior debe existir un mecanismo de comunicación común que soporte una sintaxis común, el establecimiento de la conectividad y el intercambio de operaciones petición/respuesta entre sistemas. Este aspecto de interoperabilidad está soportado inherentemente por la especificación de CORBA.

En la RGT, es necesario proporcionar interoperabilidad a las aplicaciones. Esto es, se utilizarán los sistemas de gestión de diferentes suministradores dentro de una determinada RGT de administración para soportar las diferentes funciones necesarias para la gestión de sus redes. Con el fin de simplificar la integración de estos sistemas de diferentes suministradores, deberán acordar la semántica de la información a intercambiar. Esto se consigue por medio de la especificación de un modelo de información. Esta Recomendación especifica las reglas para la definición de estos modelos de información.

4.1.2 Uso común de servicios de objetos comunes de CORBA

Un segundo aspecto de estas directrices es la confianza acerca de un uso y adaptación común del entorno de procesado distribuido elegido. Más que en la redefinición de las capacidades de la interfaz necesarias para soportar las funciones comunes de gestión de red, tales como la denominación de objetos y el filtrado de notificaciones entre cada modelo de información, estas directrices se basan en un conjunto de servicios de soporte. Estos servicios de soporte permiten que los modelos de información sean más sencillos, al tiempo que mejoran su interoperabilidad. Los servicios de soporte necesarios para las interfaces basadas en CORBA están especificados en UIT-T Q.816.

4.1.3 Transparencia del modelo de información

Si se utiliza CORBA en sitios dentro de la arquitectura RGT en los que están bien establecidos otros modelos de información ya existentes (por ejemplo, GDMO), entonces el marco de trabajo debe soportar la reutilización de esos modelos sin hacer modificaciones importantes.

Se necesita una forma única para hacer corresponder estos modelos de información GDMO al IDL del OMG, de forma que el protocolo de aplicación presente siempre los mismos modelos a la aplicación con el mismo conjunto de servicios (capacidades).

4.2 Entidades

Un **tipo entidad** describe a una clase de "cosa" en el mundo real. Cada **tipo entidad** tiene propiedades específicas, llamadas atributos.

Un **ejemplar de entidad** (o **entidad**) (por ejemplo, Conjunto de circuitos #1) es de un **tipo entidad** (por ejemplo, conjunto de circuitos #1). Los atributos de cada **entidad** tienen valores que representan el estado de ese ejemplar. Asimismo, cada **entidad** debe ser inequívocamente identificable.

En CORBA, una **entidad** puede tener acceso por diferentes métodos. Por ejemplo una **entidad** puede ser accedida por medio de una estructura de datos IDL, por un tipo de valor, o un tipo de interfaz. Esta Recomendación presenta la forma en la que se utiliza CORBA para definir **tipos de entidades**.

4.2.1 Granularidad de acceso

En el contexto de las operaciones de la RGT, la granularidad define el nivel de abstracción que se expone entre sistemas. La granularidad de acceso identifica el nivel al que las **entidades** pueden ser accedidas (es decir, la forma en la que se expone una información vía una interfaz). En CORBA,

cada objeto CORBA está provisto de una dirección única, conocida como referencia de objeto interoperable (IOR, *interoperable object reference*). La IOR proporciona una dirección al sistema cliente que identifica a qué sistema servidor se debe conectar para comunicarse con el lado servidor del objeto CORBA.

En CORBA es posible definir diferentes abstracciones de acceso (es decir, granularidad de acceso) a las entidades definidas para la RGT (por ejemplo, las de UIT-T M.3100). Aquí se definen dos diferentes abstracciones de acceso:

1) **Granularidad de ejemplar**: Cada entidad tiene su propia IOR. Para la creación de nuevas entidades, esto implica la ejemplificación de un nuevo objeto CORBA.

   -1 IOR / ejemplar entidad

   Por ejemplo, un **tipo entidad** en el dominio ATM es un *atmLink*. En la técnica de ejemplar granulado, se define un objeto CORBA para que soporte los mismos atributos que tiene el tipo entidad al que representa. Para cada ejemplar del *atmLink*, se crea un objeto CORBA independiente. Por tanto, se puede direccionar inequívocamente a cada *atmLink* por medio de su IOR.

2) **Granularidad específica de aplicación**: Se accede a los ejemplares de un tipo bien definido de tipos entidad vía un único IOR (una única interfaz).

   -1 IOR / familia (conjunto de) tipos de entidad

   Se definen operaciones a granel en interfaces del IDL de CORBA específicos de la aplicación, que pasan las identidades y los estados de las entidades gestionadas por medio de parámetros de operación que emplean listas de tipos estructurados del IDL.

Las directrices para el modelado de objetos CORBA definidas en esta Recomendación son aplicables a la especificación de interfaces de objetos gestionados que soporten granularidad de acceso de ejemplar granulado. Se pueden también definir normas RGT utilizando granularidad de acceso específica de la aplicación. Sin embargo, están fuera del propósito de esta Recomendación dichas especificaciones de interfaz.

4.3 Principios de contenencia y denominación

La contenencia es una representación lógica de la forma en la que entidades de un tipo contienen a entidades de otro tipo. Un árbol de contenencia define la relación entre diversos ejemplares de entidad. Un ejemplar de entidad está contenido por un, y sólo un ejemplar de entidad contenedora. Los ejemplares de entidades contenedoras pueden, ellos mismos, estar contenidos en otro ejemplar de entidad, formando un gráfico directo. El gráfico directo forma lo que se llama el árbol de denominación (o de contenencia).

Se puede utilizar la relación de contenencia para el modelado de jerarquías de elementos del mundo real (por ejemplo, montajes, submontajes y componentes) o a jerarquías organizativas del mundo real (por ejemplo, nombre de una compañía, nombre de una organización).

En la figura 2 que sigue, se presenta un posible árbol de contenencia.

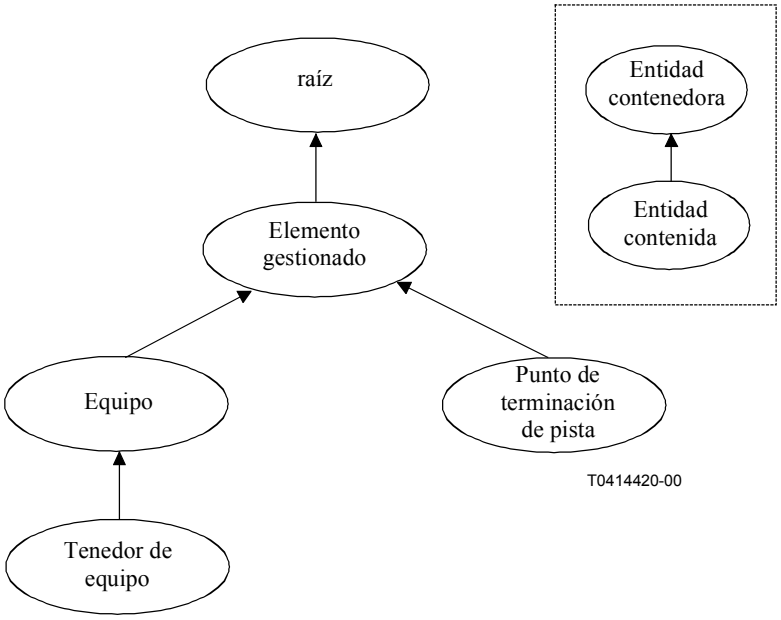

Figura 2/X.780 – Ejemplo de contenencia

4.3.1 Denominación

Una finalidad de las relaciones de contenencia es la de la denominación de entidades. Los nombres se escogen para que sean únicos dentro de un contexto específico; en la RGT este contexto viene determinado por el ejemplar de la entidad contenedora.

Una entidad que recibe un nombre en el contexto de otra entidad, se dice que es una "entidad subordinada". A la entidad que establece el contexto de denominación (este término se utiliza en general y no tendrá una connotación directa de un servicio o de un contexto COS) para otras entidades, se la llama "entidad superior".

Se denomina a una "entidad subordinada" por medio de la combinación de:

– El nombre de su "entidad superior".

– La información que identifica inequívocamente a esta "entidad subordinada" dentro del objetivo de su entidad superior.

El nombre de una entidad que es único dentro de un contexto local de denominación, puede no serlo dentro del contexto de denominación mayor. Sin embargo, si el contexto de denominación local es único dentro del contexto mayor, se puede hacer único al nombre local cualificándolo por su contexto de denominación. Se utiliza como cualificador al nombre del contexto de denominación. Se puede visualizar esta disposición como un gráfico directo con cada una de sus líneas (o flechas) apuntando desde un objeto nombrado a un contexto de denominación.

A su vez, el contexto de denominación puede ser cualificado recurrentemente por otro contexto de denominación, de forma que se puede visualizar la estructura completa de denominación como una jerarquía con una única raíz. A esta jerarquía se le llama árbol de denominación. Por tanto, las "entidades superiores" se convierten en contextos de denominación y sus nombres se convierten en nombres de los contextos. El nombre de un objeto sólo es necesario que sea único dentro del contexto de sus entidades superiores; dentro de un contexto más amplio, su nombre está siempre cualificado por nombres de sus entidades superiores.

4.3.2 Identificación de entidad

Debido a que una "entidad superior" puede contener varias "entidades subordinadas" del mismo tipo, cada una de estas entidades contenidas del mismo tipo deben ser distinguibles en relación con su

entidad contenedora. Al nombre relativo de una entidad dentro de su entidad contenedora se le llama **nombre distinguido relativo** (RDN, *relative distinguished name*) de la entidad. Por ejemplo, puede haber varios tenedores de equipos dentro de un elemento gestionado. Para identificar inequívocamente a cada uno de los tenedores de equipos dentro del elemento gestionado, los tenedores de equipos deben estar provistos de un RDN. El RDN identificará el nombre del tipo entidad (por ejemplo, tenedor de equipos, que es un tipo entidad) y un valor único dentro del alcance de la entidad contenedora.

Un RDN es un elemento básico de un **nombre distinguido** (DN, *distinguished name*), como se especifica en UIT-T X.720. Se define un DN por medio de una secuencia de RDN comenzando desde un contexto específico. El DN entrega un nombre único relativo a este contexto.

4.4 Clases de objetos gestionados

Estas directrices de modelado especifican que cada tipo de entidad se corresponda uno a uno con una interfaz operativa de CORBA. Cuando se hace corresponder de esta manera a un tipo de entidad, el objeto CORBA que representa al tipo de entidad recibe el nombre de *clase de objeto gestionado.* Una clase de objeto gestionado debe tener también la capacidad de emitir notificaciones (véase UIT-T X.703).

El término "clase de objeto gestionado" está definido en UIT-T X.720. Como se explica en UIT-T X.703, las clases de objetos gestionados, y las subclases, se corresponden con interfaces y con interfaces derivadas.

4.5 Lotes

En el IDL de CORBA es necesario tener la noción de lotes. Los lotes son grupos de capacidades (atributos, acciones o notificaciones) que pueden ser soportadas condicionalmente por un ejemplar de objeto gestionado. Un sistema de gestión debe tener la capacidad de determinar qué lotes son soportados por un ejemplar de objeto gestionado. Si se llevan a cabo operaciones sobre un objeto gestionado, y esas operaciones están contenidas por un lote condicional que no está ejemplificado por ese objeto gestionado, entonces el objeto gestionado deberá indicar un error.

4.6 Atributos

Las directrices deben soportar la definición de atributos (es decir, propiedades visibles) sobre las clases de objetos gestionados.

4.6.1 Obtener (GET) y asignar (SET)

El valor de un atributo puede ser observable o modificable a través de una interfaz estándar. Si es observable, el modelador de información debe definir un método "get" para ese atributo. Si es modificable, el modelador de información debe definir un método "set" para ese atributo.

4.6.2 Atributo genérico Get

Los modelos de información RGT basados en CORBA deberán permitir a un sistema de gestión la posibilidad de leer grupos arbitrarios de atributos desde un único objeto gestionado con una sola operación. Este servicio permite que se lleven a cabo con una única operación muchas de las tareas de gestión. Es necesario el soporte del atributo genérico Get.

4.6.3 Atributos con el valor asignado (Set)

Para los atributos que contienen listas de valores, un modelador deberá tener la capacidad de permitir a los sistemas de gestión añadir o retirar valores particulares a/desde listas, sin reenviar toda la información de la lista original.

4.7 Creación y supresión de objetos gestionados

La existencia de objetos gestionados (MO, *managed objects*) está estrechamente relacionada con la relación de contenencia entre los MO. La existencia de MO está ligada a la existencia del ejemplar MO superior de ese MO. Si no existe el ejemplar "MO superior" especificada para un "MO subordinado", entonces no se puede crear ese "MO subordinado". Análogamente, si se suprime el "MO superior" de un MO, entonces no puede seguir existiendo ese "MO subordinado" (ni los "MO subordinados" del subordinado). Dicho esto, hay semánticas de creación y supresión que tienen que ser cumplidas en el marco de trabajo RGT de CORBA.

Las siguientes subcláusulas definen los requisitos de alto nivel que deben ser soportados en la creación y supresión de objetos. La Recomendación UIT-T Q.816 describe los servicios genéricos utilizados para llevar a cabo la creación (es decir, la factoría) y la supresión (es decir, la factoría en coordinación con el que termina el servicio). La cláusula 6 define directrices de modelado sobre la forma en la que se soportan los requisitos definidos en esta cláusula.

4.7.1 Creación

En la creación de un objeto gestionado, se deben identificar tres aspectos de la existencia del MO:

- El nombre del MO.
- Los valores de los atributos del MO.
- Los lotes condicionales del MO que hay que ejemplificar con la creación del nuevo MO.

Téngase en cuenta, que la definición de estos aspectos en la petición de creación puede ser bien explícita o implícita. En las siguientes tres cláusulas se definen las opciones para la identificación de estos aspectos de la existencia de un MO.

4.7.1.1 Identificación del nombre del MO

El nombre del MO a crear se puede determinar de una de las dos formas siguientes:

1) El gestor puede especificar, como un parámetro de la operación de creación, una referencia a un MO existente que vaya a ser el superior del nuevo MO y puede especificar el RDN del nuevo MO en la lista de atributos de la operación de creación. Esto da como resultado que la especificación completa del nombre del MO sea proporcionada por el gestor.
2) El gestor puede especificar, como un parámetro de la operación de creación, una referencia a un MO existente que vaya a que ser el superior del nuevo MO y puede omitir la especificación del RDN del nuevo MO. En este caso, el RDN del nuevo MO es asignado por el sistema gestionado.

En caso de que la información asociada no sea correcta, o que por alguna otra razón no se pueda llevar a cabo la operación de creación, entonces la factoría que trate de llevar a cabo la operación deberá indicar un error.

4.7.1.2 Identificación de los atributos del MO

Cuando se crea un MO, sus atributos son valores asignados que son válidos para el tipo de atributo. Estos valores se derivan de la información de la operación creación y de la definición de la clase del MO de una de las dos siguientes maneras:

1) Se permite que la petición de creación especifique un valor explícito para atributos individuales. Cuando se crea el MO, se asignan valores explícitos a los atributos, de la forma que sea requerida según la definición de la clase del MO.
2) Se permite que la definición de clase del MO especifique la forma en la que se asignen los valores por defecto a los atributos que no hayan sido asignados por la operación de creación.

Si no se especifican valores por defecto para un atributo, entonces el sistema de gestor debe suministrar un valor para ese atributo en la petición de creación. Si no se especifica un valor para ese atributo, se producirá un error.

Si en la petición de creación se define un valor explícito para un atributo en particular, entonces el MO tomará ese valor para el atributo especificado, por encima de cualesquiera valores por defecto que puedan estar especificados para ese atributo.

4.7.1.3 Identificación de lotes de MO para ejemplificación

Con el fin de asegurar que los recursos subyacentes pueden ser ejemplificados con las capacidades solicitadas, el gestor debe ser capaz de especificar las capacidades (es decir, los lotes condicionales) que el objeto gestionado deberá tener ejemplificadas.

La ejemplificación de un lote condicional se producirá si se cumple una condición asociada en el objeto gestionado a ejemplificar. El gestor puede también solicitar la ejemplificación de un lote condicional como parte de la petición de creación incluyéndolo en los atributos de lote de la petición de creación.

4.7.2 Supresión

En la supresión, su semántica puede soportar la supresión de todas las entidades contenidas, mientras que, en otros casos el método de supresión falla inmediatamente si están contenidas entidades subordinadas. Se debe mantener esta semántica para cada tipo de entidad.

4.8 Herencia

Se puede definir una "clase de objeto gestionado" como una especialización de otra "clase de objeto gestionado" por medio de la herencia. La especialización de una "clase de objeto gestionado" implica que todos los métodos y atributos definidos para la superclase, serán también soportados por la subclase.

En el IDL de CORBA, un atributo o una operación no pueden ser heredados por más de una interfaz, ni una subclase puede redefinir una operación o atributo heredado. (Téngase en cuenta, en general, que no se espera que un modelo de información CORBA defina un método o un atributo en una clase, en los casos en los que el mismo método o atributo pueda también estar definido en una superclase. Sin embargo, hay casos en la correspondencia desde GDMO a IDL en los que puede ocurrir esto. Por ejemplo, debido a que los atributos GDMO especifican valores permitidos y valores necesarios, una subclase en GDMO puede algunas veces redefinir el mismo atributo. Se debe tener cuidado que en la correspondencia a IDL no se redefina ese mismo atributo).

En CORBA una subclase no puede heredar el mismo atributo o método (con el mismo nombre) desde más de una superclase (a no ser que a su vez lo haya heredado de la misma clase base). Tampoco, una subclase puede redefinir el mismo atributo o método (con el mismo nombre) definido en una de sus superclases.

Estas directrices no ponen restricciones sobre las herencias de CORBA.

5 El módulo IDL del modelo de objeto

Antes de la descripción de las reglas para la definición de objetos gestionados RGT utilizando el lenguaje de definición de interfaz (IDL) [2] de CORBA, esta cláusula presenta un módulo de gestión de red que contiene un conjunto de interfaces objeto y que soporta estructuras de datos especificadas en el IDL de CORBA. Este módulo IDL tiene la finalidad de jugar un papel en la gestión de red basada en CORBA similar al jugado por las definiciones GDMO y ASN.1 de UIT-T X.721 [6] para el CMIP. Proporciona el conjunto básico de definiciones del IDL sobre las que se construyen después los modelos de información.

En los anexos A y B de esta Recomendación se incluye el IDL. El anexo A contiene las clases base (interfaces), las estructuras de datos y las notificaciones. El anexo B es un fichero independiente que sólo contiene las definiciones de constantes. Ambos están basados en las definiciones GDMO y ASN.1 que se encuentran en la UIT-T X.721.

La Recomendación UIT-T X.721 es una fuente apropiada para las capacidades que deben ser proporcionadas por los modelos de información de gestión de red. La Recomendación UIT-T X.721 define las siguientes clases de objetos gestionados utilizando GDMO:

- 9 tipos de registros (registro de fichero registro cronológico, registro de fichero registro cronológico de eventos, registro de alarmas, registro de cambio de valor de atributo, registro de creación de objeto, registro de supresión de objeto, registro de relación, registro de informe de alarma de seguridad, registro de cambio de estado);
- discriminador y discriminador de retransmisión de eventos;
- fichero registro cronológico;
- sistema;
- tope.

Cada uno de ellos tiene atributos, acciones y tipos de datos y parámetros de soporte. Adicionalmente la Recomendación UIT-T X.721 define 15 notificaciones.

Observando a las clases de objetos gestionados antes listados, está claro que muchos de ellos están tratados por los servicios de objetos comunes CORBA ya incluidos en el marco de trabajo (véase UIT-T Q.816 para detalles sobre servicios RGT basados en la RGT de CORBA):

- El servicio de registro cronológico de eventos de telecomunicación de CORBA define una estructura para mantener registros cronológicos, de forma que no es necesaria la redefinición de las clases de registros. (Téngase en cuenta que especificando la utilización del servicio de registro cronológico de eventos de telecomunicación, el marco de trabajo RGT de CORBA trata los registros cronológicos como estructuras de datos, no como objetos).
- El servicio notificación de CORBA define una capacidad de filtrado, de forma que no es necesario redefinir el discriminador ni el discriminador de retransmisión de eventos.
- El servicio de registro cronológico de eventos de telecomunicación de CORBA define el equivalente al registro cronológico de X.721.

Esto deja sólo a sistema y tope, junto a las notificaciones. Sistema no es realmente una clase del marco de trabajo y pertenece en cambio a un modelo genérico de información (en caso de ser necesaria). El IDL del anexo A, por tanto, define una interfaz de objeto gestionado "tope", llamada "objeto gestionado", que tiene como finalidad la de ser convertida en subclases por todas las demás interfaces de objetos gestionados, de forma similar a la manera en la que la clase de objeto gestionado "tope" es convertida en subclase por todas las clases de objetos gestionados CMIP. También está incluido un objeto genérico "factoría". Se utilizan las factorías de objetos gestionados para la creación de objetos. (Los servicios RGT basados en CORBA definidos en UIT-T Q.816 incluyen un servicio *Terminador* que maneja las supresiones de objetos con independencia del tipo de objeto de que se trate, pero la creación de objetos es llevada a cabo por factorías específicas de clase, de forma que las operaciones de creación de objetos puedan ser muy ligadas al tipo.) Las notificaciones se definen en una tercera interfaz del IDL. Además, están definidos diversos tipos de datos IDL. Finalmente, están definidas algunas macros IDL precompiladas para facilitar la especificación de interfaces de objetos gestionados. Más adelante se trata de cada una de estas características.

5.1 La interfaz de objeto gestionado Base (Tope)

La primera interfaz que se define en el anexo A es la interfaz *ManagedObject*, que se encuentra después de todas las definiciones de tipo de datos. Tiene como finalidad la de ser la interfaz de

objetos gestionados base, a partir de la cual heredan todas las demás interfaces. Define un conjunto de capacidades que todas las instancias de objetos gestionados deben soportar. Estas capacidades son:

- Un método que devuelve el nombre del objeto.
- Un método que devuelve el nombre de la interfaz (clase real) del objeto.
- Un método que devuelve los lotes condicionales soportados por el ejemplar del objeto.
- Un método que devuelve la fuente de creación del objeto (tanto si ha sido creado de forma autónoma por el recurso gestionado, en respuesta a una operación de gestión, o si es desconocida).
- Un método que devuelve la política de supresión del ejemplar. Este es un valor enumerado, e indica si el objeto es no suprimible, si es suprimible sólo si no contiene objetos, o si se suprimirán todos los objetos contenidos cuando se le suprima.
- Un método que devuelve un objeto de tipo de valor de CORBA que contiene todos los atributos legibles del objeto.
- Una operación de destrucción.

El IDL que describe la interfaz *ManagedObject* (sin comentarios) es:

```
interface ManagedObject {
      NameType nameGet()
         raises (ApplicationError);
      ObjectClassType objectClassGet()
         raises (ApplicationError);
      StringSetType packagesGet()
         raises (ApplicationError);
      SourceIndicatorType creationSourceGet()
         raises (ApplicationError);
      DeletePolicyType deletePolicyGet()
         raises (ApplicationError);
      ManagedObjectValueType attributesGet (
         inout StringSetType attributeNames)
         raises (ApplicationError);
      void destroy()
         raises (ApplicationError, DeleteError);

}; // end of ManagedObject interface
```

5.1.1 La operación *nameGet()*

La primera operación, *nameGet*(), devuelve el nombre CORBA del objeto. *NameType* es una definición de tipo para el tipo *Name (nombre)* del servicio de denominación de CORBA. Se utiliza *NameType* para estar de acuerdo con los convenios de IDL definidos más adelante en esta Recomendación. Este método devuelve el nombre compuesto del objeto, comenzando con el nombre asignado al contexto local raíz de denominación bajo el cual está contenido el objeto. Esto es, el método devuelve el nombre "único global" del objeto. Véase UIT-T Q.816 para detalles acerca de la asignación de un nombre único al contexto de denominación raíz de un sistema gestionado. La excepción *ApplicationError* se define para ser provocada por cualquier operación de objeto gestionado si la operación no se puede completar debido a algún problema de recursos. Véase la cláusula 5.5 más adelante, para detalles acerca de esta y todas las demás excepciones.

5.1.2 La operación *objectClassGet()*

La operación *objectClassGet()* devuelve el nombre de la interfaz prevista (nombre de la clase real) del objeto. Los nombres de las interfaces previstas incluyen el, o los, nombres del, o de los, módulos en los que está definida la interfaz. El tipo del valor de retorno, *ObjectClassType*, es una definición de tipo cadena. Si la clase del objeto es una extensión menor de otra clase (por ejemplo, de una clase "R1"), la cadena devuelta es el nombre de la clase real (con "R1"). Por ejemplo, "EquipoR1".

5.1.3 La operación *packagesGet()*

La operación *packagesGet()* devuelve la lista de los lotes condicionales soportados por un ejemplar de objeto. La noción de lotes condicionales, cada uno con un nombre de cadena, está soportada por estas directrices. Véase la cláusula 6.6 para detalles. *StringSetType* es una definición de tipo para una lista de cadenas.

Téngase en cuenta que esto difiere ligeramente del atributo *lotes* de los objetos CMIP ya que este marco de trabajo no soporta la definición de lotes obligatorios, únicamente los condicionales. En CMIP es posible para el atributo lotes el listado de todos los lotes obligatorios. Obviamente, ya que la definición de lotes obligatorios no está soportada por este marco de trabajo, no se pueden listar en el atributo lotes de un objeto gestionado.

5.1.4 La operación *creationSourceGet()*

La operación *creationSourceGet()* devuelve un valor que indica el sistema que originó la creación del objeto. *SourceIndicatorType* es un tipo enumerado con tres valores: *resourceOperation*, *managementOperation*, y *unknown*. Indica si el objeto se creó autónomamente por el recurso, en respuesta a una operación de gestión, o si no se conoce la razón por la que se creó el objeto.

5.1.5 La operación *deletePolicyGet()*

La operación *deletePolicyGet()* devuelve la política de supresión de este ejemplar del objeto. Este es un valor enumerado que indica si el objeto no es suprimible, si es suprimible sólo si no contiene objetos, o si se suprimirán todos los objetos contenidos si es suprimido. (No está permitida la supresión de un objeto pero no la de los objetos que contiene.) Esta política se fija cuando su factoría crea el objeto en base a la información de vinculación de nombres identificada en la operación de creación.

5.1.6 La operación *attributesGet()*

Se utiliza el método *attributesGet()* para devolver todos, o cualquier subconjunto, de los valores de los atributos de un objeto en una operación. En un modelo de información, para cada interfaz de objeto gestionado, se definirá un *valuetype* CORBA que contenga los datos que constituyen cada uno de los atributos legibles en esa interfaz. [Los atributos legibles son aquellos con una operación <nombre de atributo>Get().] Se puede utilizar este método para recuperar este tipo de valor en cualquier objeto gestionado. Se definirán los tipos de valor siguiendo la jerarquía de herencia de las interfaces del objeto gestionado (excepto que los tipos de valor no pueden soportar la herencia múltiple), y cada uno se derivará finalmente desde el tipo *ManagedObjectValue* definido para la interfaz del *ManagedObject*. El objeto gestionado debe devolver un tipo de valor definido por su interfaz en respuesta a este método. Por tanto, cuando un cliente invoque la operación *attributesGet()* en cualquier objeto gestionado, recibirá como respuesta una referencia a un tipo *ManagedObjectValue* que podrá ajustar entonces al tipo de valor definido para la interfaz sobre la que se invocó la operación.

Se complica algo la situación en relación de que un cliente puede no desear entregar todos los valores de los atributos de un ejemplar, y que un ejemplar puede que no soporte todos los atributos que se encuentran en lotes condicionales. (Los tipos de valor incluyen atributos en lotes condicionales.) Se soluciona esta situación con el uso del parámetro *attributeNames* de entrada/salida. Al ser invocada, el cliente puede presentar una lista de los nombres de los atributos en los que está interesado, con una lista nula que tiene el significado especial de que se devolverán todos los atributos soportados. El objeto gestionado ignorará cualesquiera nombres que no sean nombres válidos de atributos. En su respuesta el objeto devolverá la lista real de los atributos para los que están dados los valores. Téngase en cuenta que esta lista puede no concordar con la lista presentada. El objeto debe devolver siempre una lista exacta, aunque la lista presentada fuese nula o tuviese nombres no válidos. Si todos los nombres de la lista presentada son no válidos, el objeto devolverá una lista nula y un tipo de valor vacío.

Ya que la estructura del tipo de valor está predefinida, el objeto debe rellenarla con algún valor para los atributos no pedidos o no soportados. Básicamente, el objeto puede devolver cualesquiera valores para estos atributos, pero los valores serán lo más cortos posible por cuestión de eficacia. Por tanto, se devolverán valores nulos para las cadenas, referencias y listas de cualquier tipo. Para los tipos enteros y enumerados se puede devolver cualquier valor. El cliente debe considerar como no válido cualquier valor para un atributo que no haya sido citado en la lista devuelta por el objeto.

La interfaz base *ManagedObject* actualmente tiene sólo un método que devuelve un tipo de valor CORBA que contiene todos los atributos legibles del objeto. No tiene un método similar para fijar los valores de los atributos, ya que no todos ellos son asignables.

5.1.7 La operación *destroy()*

La operación final sobre el objeto, la operación *destroy()*, se utiliza para liberar cualquier recurso asociado con el objeto gestionado y suprimirlo. La excepción *DeleteError* es provocada por el objeto si tiene una política de supresión *NotDeletable*. La excepción *DeleteError* es también un medio ampliable para la información de problemas en la destrucción de un objeto dependiente del modelo. Por ejemplo, si se trata de suprimir un punto de terminación de camino antes de que se haya suprimido el camino, puede originarse un *DeleteError*. Sin embargo, UIT-T Q.816 define un servicio llamado el "servicio terminador", para implementar la lógica necesaria para el cumplimiento de las políticas de supresión y mantener la integridad del árbol de denominación. La operación de destrucción está en realidad pensada para ser utilizada por este servicio y no deberá ser invocada directamente por un sistema de gestión. Véase UIT-T Q.816 para detalles sobre el servicio terminador.

(R) OBJETO-1. Las interfaces utilizadas para el modelado de recursos en un sistema gestionado heredarán (directa o indirectamente) de la interfaz *ManagedObject* descrita anteriormente y definida en el IDL de CORBA en el anexo A. Se soportarán las capacidades descritas anteriormente.

5.2 La factoría de objetos gestionados

A veces los objetos gestionados son creados automáticamente por el sistema gestionado, a veces se crean como resultado de un acción de otro objeto (tal como un objeto de conexión cruzada creado en respuesta a una acción conexión en una factoría) y, a veces, se crean como respuesta a una solicitud de un gestor para crear un objeto. En este último caso, en los sistemas CMIP, la operación de creación es llevada a cabo en el marco de trabajo del agente CMIP. No puede ser realizada por el propio objeto ya que todavía no ha sido creado. En las implementaciones CORBA, no existe marco de trabajo del agente, de forma que es necesaria la presencia de algún elemento en el sistema gestionado que permita al sistema gestor la creación de objetos. En los sistemas CORBA esta situación es tratada frecuentemente por objetos "factoría". La interfaz *ManagedObjectFactory* está pensada para que sea la interfaz base de la que hereden otras interfaces factoría. Definirá capacidades que se espera sean soportadas por todas las factorías de objetos gestionados. En la actualidad, no se han identificado dichas capacidades, de forma que la interfaz es nula (hereda de la nada y no tiene, ni atributos ni métodos). Es un sitio en el se podrán colocar en el futuro las capacidades en caso necesario. También sirve como una superclase común para todas las factorías.

Se espera que los modelos de información del IDL de CORBA incluyan una interfaz factoría para cada interfaz de objeto gestionado (a no ser que la clase de objeto gestionado no sea ejemplificable). Las factorías contendrán operaciones para la creación de objetos gestionados. Estas operaciones tomarán diversos parámetros, tal como el objeto superior del nuevo objeto, el nombre del nuevo objeto y valores para cada uno de los atributos escribibles o asignables en la creación, etc. Después de la creación correcta del nuevo objeto, la factoría devolverá una referencia del mismo.

Además de la creación de objetos, se espera que las factorías creen también vínculos de nombres para los nuevos objetos en el servicio de denominación de CORBA. Aunque esta funcionalidad podría ser implementada en otro sitio, se cree que su implementación en las factorías simplificará las

implementaciones, liberando a la implementación del objeto gestionado de esta tarea, permitiéndoles que se dediquen a la representación de recursos. Véase UIT-T Q.816 para detalles sobre cómo el marco de trabajo RGT de CORBA utiliza el servicio de denominación de CORBA.

Para ayudar a los clientes a que encuentren factorías, la Recomendación UIT-T Q.816 define un servicio buscador de factoría. Este servicio actúa como un negociador entre clientes y factorías. Básicamente, las factorías se registran ellas mismas con el servicio, entonces los clientes interrogan al servicio bien conocido para encontrar una factoría de un tipo particular. Véase UIT-T Q.816 para detalles sobre el servicio buscador de factoría.

(R) FACTORÍA-1. Los objetos factoría utilizados para la creación de objetos gestionados en un sistema gestionado, heredarán (bien directa o indirectamente) de la interfaz *ManagedObjectFactory* descrita anteriormente y definida en el anexo A del IDL de CORBA.

(R) FACTORÍA-2. Todas las factorías se registrarán en los objetos buscador de factoría ejemplificados en ese sistema.

5.3 La interfaz notificaciones

La tercera interfaz que se define en el anexo A es la interfaz de notificaciones. Cada una de las notificaciones en UIT-T X.721 tiene su operación correspondiente en esta interfaz. Las notificaciones están definidas como llamadas a métodos tipificados, como se exige en UIT-T Q.816. El servicio de notificación del OMG se utiliza para filtrar y difundir notificaciones. Los métodos de notificación tipificada se pueden utilizar directamente con un servicio de notificación que soporte notificaciones tipificadas. En UIT-T Q.816 se dan las correspondencias entre estos métodos de eventos tipificados y los eventos estructurados.

Todas las operaciones de notificación definidas en esta interfaz pasan diversos parámetros, algunos de los cuales son comunes a todas las notificaciones. Algunas de las notificaciones tienen parámetros idénticos, pero son utilizados por razones ligeramente diferentes. La interfaz de notificaciones IDL es la siguiente:

```
interface Notifications {
 void equipmentAlarm (
         in ExternalTimeType                 eventTime,
         in NameType                         source,
         in ObjectClassType                  sourceClass,
         in NotifIDType                      notificationIdentifier,
         in CorrelatedNotificationSetType        correlatedNotifications,
         in AdditionalTextType               additionalText,
         in AdditionalInformationSetType     additionalInfo,
         in ProbableCauseType                probableCause,
         in SpecificProblemSetType           specificProblems,
         in PerceivedSeverityType            perceivedSeverity,
         in BooleanTypeOpt                   backedUpStatus,
         in NameType                         backUpObject,
         in TrendIndicationTypeOpt           trendIndication,
         in ThresholdInfoType                thresholdInfo,
         in AttributeChangeSetType           stateChangeDefinition,
         in AttributeSetType                 monitoredAttributes,
         in ProposedRepairActionSetType      proposedRepairActions,
         in BooleanTypeOpt                   alarmEffectOnService,
         in BooleanTypeOpt                   alarmingResumed,
         in SuspectObjectSetType             suspectObjectList
         );

 ...
}; // fin de la interfaz Notifications
```

Las otras 14 operaciones de notificación son similares a la anterior. Los nombres de las 15 notificaciones definidas son:

- Cambio de valor de atributo
- Alarma de comunicaciones
- Alarma de entorno
- Alarma de equipo
- Violación de la integridad
- Creación de objeto
- Supresión de objeto
- Violación operativa
- Violación física
- Alarma de error de proceso
- Alarma de calidad de servicio
- Cambio de relación
- Violación de seguridad
- Cambio de estado
- Violación en el dominio del tiempo

Este marco CORBA requiere el uso de identificadores de notificación los que pudieran no ser necesarios en otras interfaces (no se requieren en UIT-T X.733). A título ilustrativo se indican cuatro casos posibles en los que se pueda hacer corresponder los identificadores de notificación de alarma de la interfaz elemento de red/EMS con la interfaz EMS/NMS:

1) El elemento de red siempre utiliza identificadores de notificación y el objeto gestionado se representa en ambas interfaces. En este caso, el sistema EMS transmite la alarma (con su identificador de notificación) al sistema NMS.

2) El elemento de red nunca utiliza identificadores de notificación y el objeto gestionado se representa en ambas interfaces. En este caso, el EMS utiliza un contador interno, incluye este valor como el identificador de notificación y transmite la alarma al NMS.

3) El elemento de red utiliza a veces identificadores de notificación y el objeto gestionado se representa en ambas interfaces. En razón que se requiere identificador de notificación, el EMS debe definir un valor cuando alguno no está provisto. Puede ser difícil definir un valor en el EMS debido a que los valores de identificador de notificación debe ser único a través de todas las notificaciones de un determinado ejemplar de objeto gestionado en todo momento en que la correlación es significativa [1]. De esta manera, el EMS debe escoger un valor que no se utiliza en alarmas corrientes y no se deberá utilizar en alarmas subsiguientes.

   Se debe tener extremo cuidado cuando se efectúa esta operación, pues el algoritmo para elegir los valores de identificador de notificación pertenece al sistema de producción (en este caso, el elemento de red).

   Una solución posible, es que el EMS suministre su propio valor para identificador de notificación para todas las alarmas. Esto también requeriría la actualización de listas de notificación correlacionadas de cada alarma, dando por resultado que el EMS mantiene una correspondencia completa de valores de identificador de notificación de elemento de red con los valores de identificador de notificación EMS.

   Otra solución posible, es que el EMS y el elemento de red acordaran soportar diferentes subconjuntos de números de identificador de notificación.

   De manera alternativa el EMS podría suministrar su propio número e ignorar posibles colisiones, permitiendo así que su aparición sea rara.

4) Se pone en correspondencia una alarma de un objeto de interfaz EMS/elemento de red con objetos de interfaz EMS/NMS diferentes. De modo similar al punto anterior, el EMS debe suministrar un valor de identificador de notificación que sea único para el objeto gestionado EMS/NMS. Asimismo, se deben actualizar las listas de notificación correlacionadas.

5.4 Las definiciones de tipos de datos

Precediendo a las definiciones de la interfaz en el anexo A, hay diversas estructuras de datos y definiciones de tipos. La mayor parte de ellas se utilizan en las notificaciones. Proceden del módulo ASN.1 de UIT-T X.721 con pequeños cambios para simplificar la sintaxis. Donde ha sido posible se han empleado conceptos modernos de orientación a objetos, tales como los parámetros de entrada/salida y las excepciones que aparecen reflejados en estos tipos.

Un tipo de datos a destacar es el del tipo hora. Estas directrices adoptan el código del tiempo universal definido para el servicio de hora de CORBA. Este tipo de datos consiste en un entero grande que cuenta las centésimas de nanosegundos que han transcurrido desde la medianoche del 15 de octubre de 1582. Para llevar la cuenta de la hora en todo el mundo, la hora se expresa en relación con la hora en la zona horaria de Greenwich, utilizando un entero corto con signo para expresar la diferencia. Esto significa que los sistemas basados en estas directrices deben conocer su zona horaria local. Este procedimiento hace sencilla la comparación de las horas, ya que la hora se representa como un entero. Posiblemente habrá bibliotecas estándar para la conversión de la representación con entero a otros formatos más familiares.

5.5 Excepciones

El módulo IDL del anexo A define algunas excepciones para ser utilizadas en las operaciones con objetos gestionados. Pueden ser provocadas en algunas operaciones, como se define a continuación. Además, cualquiera de las excepciones estándar de CORBA puede ser provocada en cualquier operación. Por ejemplo, la excepción "CORBA:NO_PERMISSION" puede ser provocada para indicar una violación de seguridad. Las excepciones definidas son:

```
valuetype ApplicationErrorInfoType {
          public UIDType          error;
          public Istring          details;
};
valuetype CreateErrorInfoType : ApplicationErrorInfoType {
          public MOSetType        relatedObjects;
          public AttributeSetType attributeList;
};
valuetype DeleteErrorInfoType : ApplicationErrorInfoType {
          public MOSetType        relatedObjects;
          public AttributeSetType attributeList;
};
valuetype PackageErrorInfoType : CreateErrorInfoType {
          public StringSetType    packages;
};
exception ApplicationError { ApplicationErrorInfoType info; };
exception CreateError { CreateErrorInfoType info; };
exception DeleteError { DeleteErrorInfoType info; };
```

5.5.1 La excepción ApplicationError

Se provoca una excepción *ApplicationError* cuando no se puede completar una operación debido a alguna condición a nivel de aplicación en el sistema gestionado. La información devuelta con la excepción incluye un identificador para una condición específica y una cadena con detalles adicionales o una explicación.

En este marco de trabajo se definen unos pocos identificadores para condiciones de error específicas. Se utilizarán siempre que sea posible. Los modelos de información, sin embargo, pueden definir códigos adicionales de condiciones de error, o crear sus propias excepciones.

Los datos devueltos con la excepción de error de aplicación son un tipo de valor, lo que quiere decir que puede ser expandido. Esto es, para unos determinados códigos de condición de error, el tipo de datos real devuelto puede ser una expansión del tipo de información base de error de aplicación. Ya que el código de error está en el tipo base, el código del cliente puede examinarlo, y si su valor es

uno de los devueltos en una subclase, el cliente puede estrechar (ajustar) el tipo de valor y acceder a la información adicional.

La excepción *ApplicationError* se incluirá en la cláusula *raises (provoca)* de cada operación de objeto gestionado y de factoría de objeto gestionado. En este marco de trabajo se han definido unos pocos valores de códigos de error para la excepción de error de aplicación. Se discute cada uno en las subcláusulas que siguen.

5.5.1.1 invalidParameter

Se provoca una excepción de error de aplicación con código de error *invalidParameter* cuando no es válido para la operación solicitada algún parámetro de la operación. Se devuelve en el campo de detalles el nombre del parámetro equivocado.

5.5.1.2 resourceLimit

Se provoca una excepción de error de aplicación con código de error *resourceLimit* cuando no se puede completar una operación debido a algún error transitorio en el sistema gestionado, como, por ejemplo, una escasez de memoria. Se devuelve en el campo de detalles una cadena conteniendo una explicación.

5.5.1.3 downstreamError

Se provoca una excepción de error de aplicación con código de error *downstreamError* cuando no se puede completar debido a un error hacia abajo desde el sistema gestionado. Un ejemplo de esta situación se produce cuando no se puede completar una operación debido a que un EMS no puede comunicar con un NE.

5.5.2 La excepción *CreateError*

Se provoca la excepción *CreateError* cuando ocurre un error en una operación creación de factoría. Se debería incluir en la cláusula *raises* de cada operación de creación de la factoría de objeto gestionado.

Los datos devueltos con esta excepción sobrepasan a los de un *ApplicationError* general, y añaden una lista de objetos relacionados y los valores de los atributos que habría tenido el objeto si se hubiese creado. Los códigos específicos de error definidos para esta excepción por este marco de trabajo se presentan a continuación. Siempre que sea posible, las implementaciones los deberían utilizar. Los modelos de información pueden añadir nuevos valores o definir nuevas excepciones para casos especiales.

5.5.2.1 invalidNameBinding

Se provoca una excepción de error de creación con un código de error igual a *invalidNameBinding* cuando la vinculación de nombre incluida en la operación de creación no soporta la creación del objeto en esta situación.

5.5.2.2 duplicateName

Se provoca una excepción de error de creación con un código de error igual a *duplicateName* cuando el nombre incluido en la operación de creación está duplicado.

5.5.2.3 unsupportedPackages

Se provoca una excepción de error de creación con un código de error igual a *unsupportedPackages* cuando la implementación no soporta uno o más de los lotes solicitados. Téngase en cuenta que cuando se utiliza este código de error, la estructura de datos devueltos es en realidad una *PackagesErrorInfoType*, que amplía la estructura *CreateErrorInfoType*. La estructura *PackagesErrorInfoType* incluye una lista de lotes que, en este caso serán los lotes no soportados.

5.5.2.4 incompatiblePackages

Se provoca una excepción de error de creación con un código de error igual a *incompatiblePackages* cuando algunos de los lotes solicitados no son compatibles entre sí, o con el recurso para el que se ha creado el objeto. Téngase en cuenta que cuando se utiliza este código de error, la estructura de datos devuelta es en realidad una estructura *PackagesErrorInfoType*, que expande la estructura *CreateErrorInfoType*. La estructura *PackagesErrorInfoType* incluye una lista de lotes que, en este caso serán los lotes incompatibles.

5.5.3 La excepción *DeleteError*

Se provoca una excepción *DeleteError* cuando ocurre un error en una operación de supresión. Se incluye en la cláusula *raises* de la operación de destrucción en la interfaz base *ManagedObject*, que se hereda entonces por cada objeto gestionado.

Los datos devueltos con esta excepción sobrepasan a los de un *ApplicationError* general, y añaden una lista de objetos relacionados y los valores de los atributos que tenía el objeto cuando se intentó suprimirlo. Los códigos específicos de error definidos para esta excepción por este marco de trabajo se presentan a continuación. Siempre que sea posible, las implementaciones los utilizarán. Los modelos de información pueden añadir nuevos valores o definir nuevas excepciones para casos especiales.

5.5.3.1 notDeletable

Se provoca una excepción de error de supresión con el valor constante igual a *notDeletable* cuando se hace un intento de invocar la operación destroy() sobre un objeto gestionado que no debería ser destruido de acuerdo con su política de supresión. (Téngase en cuenta que la operación destroy() de objetos gestionados está definida para ser utilizada en otras partes de este marco de trabajo. Los sistemas gestores que la invocan directamente corren el riesgo de corromper datos en el sistema gestionado.)

También el servicio terminador provocará esta excepción cuando un cliente trata de suprimir un objeto que tiene una política de supresión *notDeletable*.

5.5.3.2 containsObjects

Se provoca una excepción con error de supresión con el valor constante igual a *containsObjects* cuando se hace un intento de suprimir un objeto gestionado que tiene subordinados y una política de supresión *deleteOnlyIfNoContainedObjects*.

Los objetos gestionados no son responsables de la detección de esta condición, pero sí el servicio terminador.

5.6 Definiciones de macros

Después de las interfaces en el anexo A están las definiciones de algunas macros. Estas macros proporcionan sólo notaciones breves para identificar qué notificaciones están soportadas y por qué objetos. Debido a la capacidad limitada del IDL de CORBA para aceptar información de este tipo, se creyó en la utilidad de estas macros.

La macro *MandatoryNotification* identifica notificaciones que deben ser soportadas por un objeto, y la macro *ConditionalNotification* identifica notificaciones que debe emitir un objeto gestionado si soporta un lote particular. Ambas macros toman argumentos que identifican el nombre de una operación (recuérdese que se utilizan las operaciones para transportar notificaciones) y el nombre previsto de la interfaz para la que se define la operación. La macro *ConditionalNotification* acepta también un tercer parámetro, el nombre del lote al que pertenece la notificación.

Las notificaciones de macros se expanden en nada. Desafortunadamente, IDL es simplemente demasiado limitado como para proporcionar una forma de capturar esta información. Se podrían

generar comentarios, pero son inmediatamente descartados por el compilador. Comentarios con formato, como los utilizados en la generación de HTML, desafortunadamente no se pueden utilizar, ya que necesitan de alguna construcción de IDL a la que estar asociados. Se esperaba que el modelo de componentes venidero de CORBA proporcionaría una solución, pero las implementaciones no estarán disponibles a tiempo para estas directrices. En el futuro puede ser posible modificar las macros para que generen un IDL consistente con el modelo de componentes de CORBA. Por ahora, sin embargo, la información acerca de qué notificaciones se emiten y por qué clases de objetos es capturada por estas macros.

5.7 Las definiciones de constantes

Las especificaciones de la interfaz contienen siempre diversas constantes cuyos valores están acordados por todo el mundo para que signifiquen lo mismo. Por ejemplo, todo el mundo está de acuerdo que un "1" en un determinado campo significa una pérdida de señal, un "2" significa una pérdida de trama, etc. La Recomendación UIT-T X.721 no es una excepción, y define diversas constantes. Éstas están reproducidas en formato IDL en el anexo B. Para detalles sobre el mecanismo utilizado para transportar constantes predefinidas véase la cláusula 6.11.

6 Directrices para el modelado de información

Esta cláusula presenta directrices para el desarrollo de modelos de información de la RGT basados en CORBA. En la cláusula siguiente se proporcionan directrices para la traslación de modelos existentes especificados según GDMO.

6.1 Módulos

Los módulos IDL se utilizan para agrupar interfaces, definiciones de tipo, excepciones y otras construcciones IDL. Los módulos fijan también el espacio para los nombres; los identificadores dentro de un módulo deben ser únicos aunque puedan reutilizarse en otros módulos. En casi todos los casos se utilizará un módulo para agrupar las construcciones utilizadas en la especificación de un modelo de información. Los módulos pueden estar anidados dentro de otros módulos y los módulos pueden ocupar varios ficheros. El IDL especificado en estas directrices está contenido dentro de un único módulo llamado "itut_x780". Por ejemplo:

```
module itut_x780 {
…
}; // fin del módulo itut_x780
```

Este módulo tiene submódulos para la definición de constantes.

6.2 Interfaces

Cada *entidad* accesible vía la interfaz de gestión de red de CORBA tendrá una interfaz IDL definida para ella. Las interfaces agrupan a un conjunto de atributos y métodos que se puede suponer como si fuesen proporcionados por un único objeto de soporte lógico. Las interfaces pueden heredar capacidades de otras interfaces y las interfaces definidas para modelar una *entidad* deben heredar (directa o indirectamente) desde la interfaz llamada *ManagedObject* definida en esta Recomendación. Por ejemplo:

```
interface Equipment : ManagedObject {
…
}; // fin de la interfaz Equipment
```

Dichas interfaces reciben el nombre de "interfaces de objeto gestionado". Los objetos que soportan estas interfaces son "objetos gestionados" Debido a que la interfaz *ManagedObject* definida en esta Recomendación tiene un conjunto de capacidades que son heredadas por todas las interfaces de

objetos gestionados, cada objeto gestionado debe implementar un conjunto base de funciones para poder existir en el marco de trabajo RGT de CORBA.

Uno de los asuntos a los que puede que tengan que enfrentarse los modeladores de información es el limitado soporte de CORBA a la herencia múltiple. Una interfaz puede heredar una operación o un atributo desde múltiples superclases sólo si ellas las han heredado a su vez desde la misma superclase. Esta situación se conoce como una herencia en "diamante", y se muestra en la siguiente figura 3.

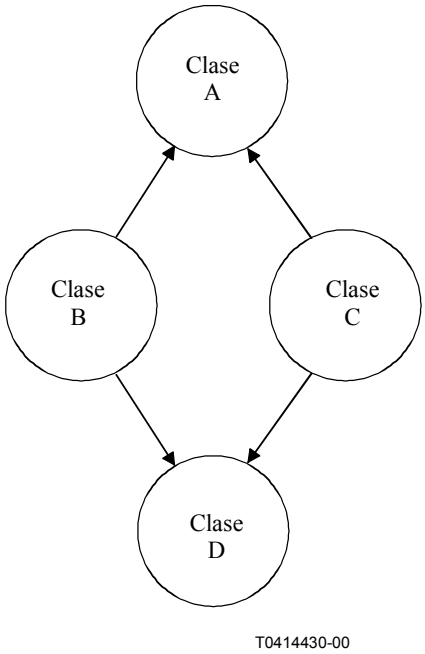

Figura 3/X.780 – Herencia en diamante

Si un modelador de información se encuentra con que tiene que heredar la misma capacidad desde dos clases diferentes que no comparten una superclase común, el modelador puede tener que modificar las clases y crear una superclase virtual a partir de la cual se pueda heredar la capacidad. Por ejemplo, creando "D" a partir de "B" y "C" por encima, pero sin que exista "A", el modelador puede que tenga que modificar las superclases creando una nueva clase virtual ("A") con la capacidad común que entonces es heredada por "B" y "C".

6.3 Atributos

Los atributos se modelan dentro de las interfaces como operaciones utilizadas para acceder al valor del atributo. Los nombres de la operación, así como los tipos de entrada y de salida, indican el nombre del atributo así como el tipo de operación. (El IDL de CORBA soporta atributos además de operaciones, pero por ahora sólo se permite que las operaciones provoquen excepciones definidas por el usuario. Como se verá, las excepciones definidas por el usuario son necesarias en los accesos a los atributos. Por esta razón, las operaciones están definidas para acceder a los atributos, más que para únicamente definir atributos. Futuras versiones de CORBA planean permitir excepciones definidas por el usuario en los accesos a los atributos, y estas directrices pueden cambiar para aprovecharlo.)

6.3.1 Atributos legibles

Los objetos gestionados tendrán una operación llamada "<nombre atributo >Get" *(obtener)* en sus interfaces para cada atributo legible. El tipo devuelto por esta operación refleja el tipo del atributo. Por ejemplo:

```
AdministrativeStateType administrativeStateGet()
        raises (ApplicationError);
```

Los atributos que son asignables pero no legibles, lo cual es raro, no dispondrán de una operación de lectura en la interfaz.

Las operaciones *get* de atributo que puedan devolver grandes cantidades de datos definirán un iterador que permita al sistema cliente el control de la devolución del flujo de información. Para un ejemplo del uso de iteradores véase UIT-T Q.816.

6.3.2 Atributos asignables

Los objetos gestionados tendrán una operación llamada "<nombre atributo>Set" *(asignar)* para cada atributo que sea asignable. El tipo de retorno en la operación será *void (vacío)* y el parámetro de entrada reflejará el tipo del atributo. Por ejemplo:

```
void administrativeStateSet (in AdministrativeStateType adminState)
        raises (ApplicationError);
```

Los atributos que no son asignables no dispondrán de una operación de este tipo en la interfaz.

6.3.3 Atributos valores de conjunto

Muchos atributos de objetos gestionados pueden contener conjuntos de valores. No obstante, en estos casos, se soportarán las operaciones definidas anteriormente (si el atributo es legible y/o escribible). Debido a que CORBA no define explícitamente un tipo complejo para los conjuntos, los tipos de entrada o los de retorno para estas operaciones serán secuencias CORBA. Los valores devueltos por estos atributos no contendrán valores duplicados, y no importa el orden de los valores. También, puede ser necesario soportar la adición o la remoción de valores de estos atributos. Estas operaciones se llamarán "<nombre atributo>Add" *(añadir)* y "<nombre atributo>Remove" *(remover)*. Los tipos de devolución para estas operaciones serán vacíos y el parámetro de entrada para cada una será una secuencia que refleje el tipo del atributo. Por ejemplo:

```
void supportedByObjectsAdd (in ManagedObjectSetType objects)
        raises (ApplicationError);
void supportedByObjectsRemove (in ManagedObjectSetType objects)
        raises (ApplicationError);
```

6.3.4 Excepciones

Las operaciones de acceso a los atributos pueden también provocar excepciones. Están definidas las siguientes excepciones para que sean provocadas en operaciones de acceso a atributos:

1) *ApplicationError*. Esta excepción estará incluida en la cláusula *raises* de cada operación de objeto gestionado, incluyendo las operaciones de acceso a atributos. Puede ser utilizado para indicar diversas situaciones, tales como un valor que esta fuera de rango, una limitación de recursos en el sistema gestionado, etc.

2) Excepciones de lote condicional. Si el atributo es parte de un lote condicional, la excepción definida para ese lote condicional, estará incluida en la cláusula *raises* de las operaciones de acceso a atributo. Se provoca cuando se hace un intento de acceder al atributo, pero el lote al que pertenece no está soportado por el ejemplar. Véase más sobre lotes condicionales en la cláusula 6.6 más adelante.

Además de esto, una implementación puede también provocar cualquiera de las excepciones CORBA estándar. Las operaciones que provocan excepciones no modificarán el valor del atributo. Un ejemplo de una operación de acceso a atributo que provoca una excepción es:

```
void supportedByObjectsRemove (in ManagedObjectSetType objects)
        raises (ApplicationError);
```

6.3.5 Atributos estándar

Los objetos gestionados modelan los recursos, y frecuentemente hay aspectos comunes entre los objetos gestionados. Esta situación se representa utilizando una relación de herencia entre clases de objetos, aunque también puede haber aspectos comunes entre objetos cuando no existe una relación de herencia. Un buen ejemplo de esto son los atributos similares. Muchos objetos gestionados tienen atributos similares. Para facilitar la implementación de las interfaces de gestión, estas directrices definen algunos tipos de datos estándar que se utilizarán para los atributos siempre que sea posible. Esto es, los modeladores tratarán de utilizar estas definiciones tipo en lugar de definir nuevos tipos. También se utilizarán, el nombre del atributo, y los nombres de las operaciones para acceder a la operación. En realidad, cuando se define un nuevo modelo, es una buena práctica reutilizar tipos de atributos y nombres que estén en modelos existentes, siempre que sea posible. Los atributos estándar están definidos en el cuadro 1.

Cuadro 1/X.780 – Atributos estándar

Tipo de datos	Nombre atributo	Método de acceso
AdministrativeStateType	administrativeState	administrativeStateGet()
AvailabilityStatusSetType	availabilityStatus	availabilityStatusGet()
BackedUpStatusType	backedUpStatus	backedUpStatusGet()
ControlStatusSetType	controlStatus	controlStatusGet()
SourceIndicatorType	creationSource (véase la nota)	creationSourceGet()
DeletePolicyType	deletePolicy (véase la nota)	deletePolicyGet()
ExternalTimeType	externalTime	externalTimeGet()
NameType	name (véase la nota)	nameGet()
ObjectClassType	objectClass (véase la nota)	objectClassGet()
OperationalStateType	operationalState	operationalStateGet()
StringSetType	packages (véase la nota)	packagesGet()
ProceduralStatusSetType	proceduralStatus	proceduralStatusGet()
StandbyStatusType	standbyStatus	standbyStatusGet()
SystemLabelType	systemLabel	systemLabelGet()
UnknownStatusType	unknownStatus	unknownStatusGet()
UsageStateType	usageState	usageStateGet()
NOTA – Estos atributos son heredados por todos los objetos gestionados.		

6.4 Acciones

Además de atributos, muchos objetos gestionados tendrán *acciones* – métodos para fines distintos que el acceso a un atributo. Los parámetros y los tipos de retorno en estas operaciones se definen simplemente para cumplir las necesidades de la acción. El nombre de la operación deberá reflejar la finalidad de la operación. Se han definido las siguientes excepciones para que sean provocadas en operaciones acción.

1) *ApplicationError*. Esta excepción estará incluida en la cláusula *raises* de cada operación de objeto gestionado, incluyendo a las operaciones acción. Puede ser utilizada para indicar diversas situaciones, tales como el valor de un parámetro que está fuera de rango, una limitación de recursos en el sistema gestionado, etc.

2) Excepciones de lote condicional. Si la acción es parte de un lote condicional, la excepción definida para ese lote condicional estará incluida en la cláusula *raises* de las operaciones acción. Se provoca cuando se hace un intento de invocar la acción pero el lote al que pertenece la acción no está soportado por el ejemplar. Véase más acerca de lotes condicionales en la cláusula 6.6 más adelante.

Además de esto, una implementación puede también provocar cualquiera de las excepciones CORBA estándar. Otras excepciones específicas de la acción pueden y deberán ser definidas para otras condiciones de error. Por otra parte, un modelo de información puede expandir los puntos de código de error definidos para la excepción *ApplicationError*.

Las acciones que puedan devolver grandes cantidades de datos definirán un iterador que permita al sistema cliente el control de la devolución del flujo de información. Para un ejemplo sobre el uso de iteradores, véase UIT-T Q.816.

6.5 Notificaciones

Se espera que la mayoría de los objetos gestionados emitan notificaciones bajo ciertas circunstancias. En el marco de trabajo RGT de CORBA, se transportan las notificaciones con invocaciones de métodos desde un objeto gestionado hacia atrás a un sistema de gestión, con la ayuda del servicio notificación. Por tanto, la operación notificación está en realidad definida para la interfaz CORBA del sistema de gestión, no para la interfaz del objeto gestionado. Estas directrices definen diversas notificaciones estándar, aunque, si hay que definir una nueva notificación se debería hacer como una operación sobre una interfaz llamada "Notificaciones" dentro del módulo del modelo de información. El nombre de la operación será el nombre de la notificación. Los parámetros de la operación reflejarán los datos de los que informará la notificación. El tipo de retorno en la operación de notificación debe ser vacío, y sólo debe tener parámetros "in" (entrada). Téngase en cuenta que no se utilizará la palabra clave "oneway" (un sentido) que precede la definición de la operación notificación. Las notificaciones que siguen estas directrices son confirmadas. Esto es, cuando un objeto gestionado envía una notificación a un canal, el canal confirmará hacia atrás la recepción de esa notificación al objeto gestionado. Análogamente, al enviar el canal la notificación a cada receptor, el canal recibe una confirmación. Las garantías de calidad del servicio especificadas en UIT-T Q.816 definen la fiabilidad del propio canal. Por tanto, se puede garantizar la entrega de notificaciones a los receptores.

Se necesita también un medio de documentar qué objetos gestionados emiten qué notificaciones. Más que simplemente anotar esto por medio de comentarios en un fichero IDL, se utiliza una declaración de macro. Actualmente, estas directrices definen dos macros, una para ser utilizada cuando la notificación es obligatoria y la otra cuando la notificación es parte de un lote condicional. Las macros están pensadas para ser usadas dentro de una interfaz de objeto gestionado y se definen como sigue:

```
MANDATORY_NOTIFICATION(<interface name>,
        <notification operation name>);
CONDITIONAL_NOTIFICATION(<interface name>,
        <notification operation name>, <package name>);
```

Por ejemplo:

```
interface Equipment : ManagedObject {
…
MANDATORY_NOTIFICATION(itut_x780::Notifications, objectCreation);
CONDITIONAL_NOTIFICATION(itut_x780::Notifications,
        equipmentAlarm, equipmentAlarmPackage);
…
}; // end of Equipment interface
```

El nombre del lote utilizado en la macro de notificación condicional es el mismo al utilizado en los otros sitios. Para detalles sobre lotes, véase la cláusula 6.6. Las macros, en realidad se expanden a nada, ya que no hay en realidad una buena alternativa en el IDL de CORBA. Por tanto, las macros tienen una finalidad de documentación y en realidad no dan como resultado la generación de código. Un ítem para un ulterior estudio es la modificación de las macros para generar el IDL que identifique las notificaciones soportadas por un objeto. La versión de la especificación del modelo de componentes CORBA presenta una oportunidad de hacer esto de una forma consistente con ese modelo. En cada macro sólo se puede listar una notificación. Esto es para hacer más sencilla la posible futura modificación de las macros.

6.6 Lotes condicionales

Estas directrices para el modelado de información soportan la noción de que no todas las capacidades definidas para una clase de objeto gestionado tienen que ser soportadas por todos los ejemplares. En realidad, se pueden definir grupos de capacidades de forma que se soporten bien todas o ninguna de las capacidades. Estos grupos de capacidades reciben el nombre de lotes. Las opciones para la representación de lotes en IDL son limitadas. La definición de una interfaz distinta para cada lote daría como resultado demasiadas interfaces, por lo que se utiliza la técnica que se describe aquí.

Cada operación que es parte de un lote condicional puede provocar una excepción definida para el lote. El nombre de la excepción será NO<nombre lote>. Por ejemplo:

```
exception NOadministrativeStatePackage {};
…
AdmininstrativeStateType administrativeStateGet()
        raises (NOadministrativeStatePackage);
```

Las notificaciones que se emiten como parte de un lote condicional se caracterizan con la declaración *CONDITIONAL_NOTIFICATION* como se ha descrito antes.

Las reglas concernientes a cuando las capacidades incluidas en un lote deberán ser soportadas y cuando no, se colocan en comentarios relativos a la interfaz del objeto gestionado. Una operación se puede incluir en más de un lote condicional por medio de una lista múltiple de excepciones NO<nombre lote> en su cláusula *raises*. Se provocará una excepción sólo si ninguno de los lotes está presente, en cuyo caso se puede provocar cualquiera de las excepciones de lote. Si una operación es obligatoria, no debe listar excepciones de lote en su cláusula *raises*. Una notificación puede listar varios lotes en la macro *CONDITIONAL_NOTIFICATION*.

6.7 Comportamiento

El IDL de CORBA no tiene medios formales para capturar el comportamiento de un objeto. En el futuro es posible que los modelos de información se documentarán con el UML, e incluirán casos de uso y diagramas de interacción de objetos. Sin embargo, IDL está limitado a los comentarios. Por tanto, cuando sea necesario o de ayuda, se deben utilizar los comentarios para la descripción del comportamiento de un objeto.

El IDL en esta Recomendación contiene numerosos comentarios. Están formateados para que sean analizados por los compiladores utilizados en la conversión de IDL a HTML para una lectura más fácil. Un comentario formateado comienza con /** y termina con */ y está asociado con la construcción IDL siguiente. En estos comentarios están permitidas las etiquetas de formateo HTML, como es el caso de algunas palabras clave (precedidas por el símbolo @") que son convertidas por los compiladores de IDL a HTML en formateados adicionales. Al observar IDL con un hojeador HTML, es conveniente tener en cuenta que se verá afectado el uso de las macros antes descritas. Ya que la expansión de las macros es llevada a cabo como parte de la conversión a HTML, se perderá la información que tenía la macro antes de expandirla. Por tanto, las macros utilizadas para identificar las notificaciones soportadas por cada objeto gestionado habrán sido expandidas.

6.8 Información de vinculación de nombres

La contenencia es una relación muy importante en la gestión de red. En el marco de trabajo RGT basado en CORBA, la contenencia se representa por medio de nombres. Esto, desafortunadamente no pone restricciones en las relaciones de contenencia que posiblemente podrían existir. No hay nada para prevenir que, por ejemplo, un objeto de red este contenido por un objeto conexión. Claramente, es deseable la existencia de algún medio que restrinja las posibles relaciones de contenencia a sólo aquellos casos que son sensibles. Estas restricciones, sin embargo, deben ser extensibles bajo control del modelador de información.

Para cumplir estas necesidades, estas directrices requieren que los módulos IDL que especifican los modelos de información RGT basados en CORBA, contengan también información que defina las posibles relaciones de contenencia entre las clases de objetos gestionados. A esta información de relaciones de contenencia se la llama *información de vinculación de nombres de objetos gestionados*. (Desafortunadamente, esto puede ser fácil de confundir con la información de vinculación de nombres del servicio de denominación de CORBA. Los dos no son lo mismo.)

La información de vinculación de nombres de objetos gestionados se representa en el IDL de CORBA utilizando los siguientes convenios:

1) Cada módulo IDL del modelo de información contendrá un submódulo llamado "NameBindings" para la información de vinculación de objetos gestionados.

2) Dentro de este módulo de vinculación de nombres, se definirán submódulos para cada una de las relaciones de contenencia permitidas.

3) Cada submódulo de vinculación de nombres asignará valores a estas 7 constantes:

```
const string            superiorClass
const boolean           superiorSubclassesAllowed
const string            subordinateClass
const boolean           subordinateSubclassesAllowed
const boolean           managersMayCreate
const DeletePolicyType  deletePolicy
const string            kind
```

La constante *superiorClass* contiene el nombre previsto de la clase del objeto superior (contenedor). Si un objeto puede ser el objeto "tope más alto" de un sistema gestionado, esto es, si puede estar contenido directamente bajo un contexto local raíz de denominación, el valor de vinculación de nombre *superiorClass* será una cadena vacía. La constante *superiorSubclassesAllowed* es un campo booleano que tendrá un valor *true (verdadero)* si son aceptables subclases del tipo de la clase superior utilizando esta vinculación de nombres. La constante *subordinateClass* contiene el nombre previsto de la clase del objeto subordinado (el objeto a crear). La constante *subordinateSubclassesAllowed* indica si se pueden crear subclases del objeto subordinado utilizando esta vinculación de nombre. La bandera *managersMayCreate* indica si la creación del objeto está soportada a través de la interfaz de gestión utilizando esta vinculación de nombre. El interés de fijar esta bandera a *false (falso)* es que eso permite que toda la información de relaciones de contenencia se

documente en IDL, aun si el objeto subordinado está creado sólo por el sistema gestionado. La constante *deletePolicy* contiene el valor que se asignará al atributo *deletePolicy* del objeto gestionado cuando sea creado. La constante *kind (género)* contiene el valor que se asignará al campo *kind (género)* en la vinculación de nombres CORBA al objeto cuando se cree.

El valor elegido para el campo *género* en una vinculación de nombres típicamente será el nombre de clase subordinada no previsto. (Los nombres de clases no previstas se utilizarán típicamente para reducir la longitud de los nombres.) El objetivo principal del campo *género* es segmentar el espacio de nombre para mantener colisiones de nombre desde su ocurrencia. Los módulos de vinculación de nombres para las nuevas versiones de interfaces existentes podrían volver a utilizar los valores de *género* empleados para las interfaces antiguas. Por ejemplo, los módulos de vinculación de nombres para las interfaces Equipment y EquipmentR1 podría utilizar el valor "Equipment". De otro modo, probablemente será más seguro utilizar un valor único para cada clase de interfaz.

4) El nombre de un submódulo de vinculación será <subordinateClass>_<superiorClass>, donde <subordinateClass> es el valor asignado a la constante *subordinateClass,* y <superiorClass> es el valor asignado a la constante *superiorClass* en el módulo. Si los dos módulos de vinculación de nombres en el mismo módulo padre comparten los mismos valores de *superiorClass* y *subordinateClass,* pero difieren en otros valores, se añadirá al nombre de uno de los módulos una palabra que denote una diferencia entre los dos. Por ejemplo: "Equipment_Equipment" y "Equipment_Equipment_NotDeleteabe".

Algún ejemplo de vinculación de nombres de objetos gestionado:

```
module itut_m3120 {
…
  /** El siguiente módulo contiene información de vinculación de nombres */
  module NameBindings {
        /** Este módulo de vinculación de nombres permite la creación de
        objetos Equipment debajo de objetos Managed Element objects.
        */
        module Equipment_ManagedElement {
                const string    superiorClass = "itut_m3120::ManagedElement";
                const boolean   superiorSubclassesAllowed = TRUE;
                const string    subordinateClass = "itut_m3120::Equipment";
                const boolean   subordinateSubclassesAllowed = TRUE;
                const boolean   managersMayCreate = TRUE;
                const DeletePolicyType deletePolicy =
                                itut_x780::DeleteOnlyIfNoContainedObjects;
                const string    kind = "Equipment";
        }; // fin del módulo Equipment_ManagedElement de vinculación de nombres
        /** Este módulo de vinculación de nombres permite la creación de
        objetos Equipment debajo de otros objetos Equipment.
        */
        module Equipment_Equipment {
                const string    superiorClass = "itut_m3120::Equipment";
                const boolean   superiorSubclassesAllowed = TRUE;
                const string    subordinateClass = "itut_m3120::Equipment";
                const boolean   subordinateSubclassesAllowed = TRUE;
                const boolean   managersMayCreate = TRUE;
                const DeletePolicyType deletePolicy =
                                itut_x780::DeleteOnlyIfNoContainedObjects;
                const string    kind = "Equipment";
        }; // fin del módulo Equipment_Equipment de vinculación de nombres
  }; fin del módulo de vinculación de nombres
}; fin del módulo itut_m3120
```

Téngase en cuenta que la constante *deletePolicy* es del tipo enumerado, y de acuerdo con las reglas de definición de constantes del IDL de CORBA; si este tipo está definido en otro módulo, el valor asignado a la constante deberá ser el previsto para ese módulo. *DeletePolicyType* está definido en el módulo itut_x780, y el módulo IDL ejemplo es el itut_m3120. Por tanto, el valor de *DeleteOnlyIfNoContained* debe ser el previsto precediéndolo con la cadena "itut_x780::". El propio

tipo, *DeletePolicyType*, debe también estar previsto. Esto se puede hacer con una declaración *typedef* al comienzo del módulo.

6.9 Factorías

El marco de trabajo de la RGT basado en CORBA define un servicio para la supresión de objetos, pero los objetos se crean con una clase específica *factories (factorías)*. Las factorías son objetos con interfaces distintas de las de los objetos que normalmente crean, aunque normalmente relacionadas. Cada clase de objetos gestionados tendrá también una clase factoría. Esto se hace así, de manera que las operaciones de creación de factoría puedan ser fuertemente tipificadas y sean específicas de la clase de objetos que crean. El resultado de esto es que los módulos IDL que definen interfaces de objetos gestionados contendrán también interfaces para las factorías utilizadas en la creación de los objetos. El nombre de la interfaz IDL de la factoría será "<Nombre de la Clase del Objeto Gestionado>Factory".

Esta Recomendación define una interfaz base de factoría de objeto gestionado a partir de la que debe heredar cada interfaz factoría. Las factorías no siguen la misma jerarquía de herencia que la de los objetos que ellas crean. Las factorías heredan simplemente desde la interfaz *ManagedObjectFactory*. Un ejemplo de una definición de una interfaz factoría es:

```
interface EquipmentFactory : ManagedObjectFactory {
…
}; // fin de la interfaz EquipmentFactory
```

Ya que las factorías no pueden crear subclases de objetos, hay que definir nuevas factorías para cada subclase.

Cada clase ejemplificable tendrá una factoría creada para ella, aun si por el momento no están definidos módulos de vinculación de nombres que permitan a los gestores la creación de ejemplares. Esto es para permitir la futura definición de módulos de vinculación de nombres que haga posible que los gestores creen ejemplares.

6.9.1 Operaciones de creación

Cada interfaz factoría definirá una única operación que los clientes utilizarán para crear objetos. El nombre de esta operación será "create" (creación) y devolverá una referencia al tipo de objeto creado por la factoría. Los cuatro primeros parámetros de cada operación creación son siempre los mismos. Después de ellos vienen parámetros para cada atributo escribible o asignable en creación definido para el objeto gestionado. (Un atributo asignable en creación, es uno para el que el objeto no tiene operación "set" (asignar), pero para el que está especificado un valor durante la operación creación.) Los nombres de estos parámetros son los mismos que los del nombre del atributo. (Este es el nombre de una operación de acceso al atributo sin el "Get" o "Set" final.) Cada operación de creación tiene también que aceptar parámetros para asignar valores de cualquier atributo que sea escribible o asignable en creación de todas las superclases del objeto creado por la factoría. A continuación aparece un ejemplo de una operación creación por una factoría llamada equipment:

```
Equipment create(
        in NameBindingType nameBinding, // nombre del módulo que contiene información NB
        in ManagedObject superiorObject,// Referencia al objeto contenedor.
        inout string name,              // Entrada/salida, puede ser nulo en auto
   creación.
        in StringSetType packages,      // Lista de lotes requeridos.
        …                               // Valores escribibles y asignables en creación
                                        // para atributos de la superclase de Equipment.
        …                               // Valores escribibles y asignables en creación
                                        // para atributos de Equipment.
);
```

6.9.1.1 Vinculación de nombres

El parámetro vinculación de nombres transporta el nombre de un módulo que contiene información de vinculación de los nombres de objetos gestionados, como se describe en la cláusula 6.8. Un valor de ejemplo puede ser "itut_m3120::NameBindings::Equipment_Equipment". Dado esto, la factoría puede comprobar si el valor es un identificador de vinculación de nombres es válido. (Una factoría puede ser bien "codificada en firme" con la información de vinculación de nombres disponible cuando se compila el sistema, o puede acceder a la información en la interfaz almacén de CORBA durante su funcionamiento.) Si no se puede encontrar la información de vinculación de nombres, la factoría provocará una excepción *invalidParameter ApplicationError*, devolviendo "nameBinding" como un argumento. (Esto es una excepción *ApplicationError* con el código de *error* asignado a *invalidParameter* y la cadena *details (detalles)* asignada a "nameBinding".) Si se puede encontrar la información de vinculación de nombres, pero está incompleta, la factoría provocará una excepción *invalidNameBinding CreateError*.

La factoría debe también comprobar si el tipo de la clase subordinada especificado en el módulo de vinculación de nombres concuerda con el tipo de objetos que crea. Si no concuerda, la factoría puede comprobar entonces si el tipo de objetos que crea es una subclase del valor constante de la clase subordinada. Si lo es, y si la constante *subordinateSubclassesAllowed* es *true*, puede continuar y crear el objeto. Si no, rechazará la petición provocando una excepción *invalidNameBinding CreateError*.

Finalmente, si la constante *managersMayCreate* en el módulo de vinculación de nombres es *false*, la factoría rechazará también la petición provocando una excepción *invalidNameBinding CreateError*. (Las factorías pueden tener una segunda operación de creación para uso interno por el sistema gestionado que no comprueba este valor y que no se muestra a través de la interfaz de gestión.) La inclusión de módulos de vinculación de nombres con valores de *managersMayCreate* asignados a falso permite la captura de toda la información de contenencia en IDL, como es posible con GDMO, aun si los objetos están creados sólo por el propio sistema gestionado.

La demás información en el módulo de vinculación de nombres será utilizada por la factoría cuando cree el objeto y su vinculación de nombres del servicio de denominación de CORBA. La constante *deletePolicy* estará asignada al atributo del nuevo objeto gestionado del mismo nombre. El valor de la constante *kind* se utilizará cuando la factoría cree la vinculación de nombres del objeto gestionado en el servicio de denominación de CORBA.

6.9.1.2 Objeto superior

El segundo parámetro en la operación de creación es una referencia al objeto superior, bajo el que se va a crear el nuevo objeto. Utilizando las capacidades estándar de CORBA, la factoría examinará la clase del objeto superior para determinar si concuerda con el tipo especificado en la constante *superiorClass* definida en el módulo de vinculación de nombres. Si no concuerda, la factoría debe comprobar a continuación si la referencia suministrada es la de una subclase del tipo especificado en la constante *superiorClass*. Si lo es, y si la constante *superiorSubclassesAllowed* en la vinculación de nombre es *cierto*, la factoría puede continuar la creación del objeto. Si no, la factoría debe rechazar la petición provocando una excepción *invalidNameBinding CreateError*, devolviendo "superiorObject" en los detalles.

Si la constante *superiorClass* en el módulo de vinculación de nombres es una cadena vacía, entonces se pueden crear objetos de la clase subordinada sin objeto superior (padre), y su nombre está asociado directamente a un contexto local raíz de denominación. Normalmente, estos objetos serán creados por el sistema gestionado, pero en estos casos la referencia al objeto superior será nula.

6.9.1.3 Nombre

El tercer parámetro es el nombre a asignar al nuevo objeto. Esta cadena se convertirá en el campo *ID* de la vinculación de nombres de CORBA creada en el servicio de denominación de CORBA para el

nuevo objeto. Esto será relativo al nombre del objeto superior. Si el parámetro es *inout (entrada salida)*, indica que la factoría debe soportar autonombrado. En este caso, un cliente puede presentar una cadena nula para el nombre, y la factoría elegirá una cadena adecuada y devolverá el valor elegido. Si, en cambio, el cliente presenta una cadena, la factoría utilizará este valor (y lo devolverá como valor de salida). Si el parámetro es sólo *in (de entrada)*, no se soporta el autonombrado y el cliente debe suministrar un nombre. Si no lo hace, la factoría provocará una excepción *badName CreateError*. Si el nombre suministrado está duplicado, la factoría provocará una excepción *duplicateName CreateError*. (Esto significa que ambos campos *ID* y *kind (género)* concuerdan con un objeto existente contenido por el objeto superior).

6.9.1.4 Lotes

El atributo lotes es importante. Dice a la factoría, no sólo qué lotes debe soportar un ejemplar, sino los valores de parámetros que debe ignorar en la operación de creación. Debido a que están fuertemente tipificados, los métodos de creación incluyen un parámetro para cada atributo escribible o asignable en la creación de un objeto, aun si un atributo es parte de un lote condicional. La factoría debe ignorar los valores de cualquier atributo en los lotes que no sean solicitados por el cliente, aun sí la factoría ejemplificase en cualquier caso el objeto con el lote. (Si la factoría ejemplificase un objeto con un lote no solicitado por el cliente, la factoría debe elegir los valores iniciales.) Esto libera al cliente de tener que suministrar valores para atributos en lotes que no desea. En su lugar el cliente puede presentar cualquier valor. Por cuestión de eficacia, los valores presentados para atributos en lotes no solicitados por el cliente serán cortos.

Si el cliente proporciona un nombre de lote no válido en los parámetros de lotes, la factoría provocará una excepción *unsupportedPackage CreateError* y devolverá el nombre del lote como argumento. Se puede también provocar una excepción *incompatiblePackages CreateError* si el cliente solicita la creación de un ejemplar, pero especifica lotes que puede que no coexistan en el mismo ejemplar.

6.9.1.5 Parámetros de superclase

A continuación de estos cuatro parámetros habrá parámetros para cada uno de los atributos escribibles o asignables en la creación para cualquiera de las superclases del tipo de objetos creados por la factoría.

6.9.1.6 Parámetros de clase de objeto

Finalmente, a continuación de los parámetros de superclase, habrá parámetros para cada uno de los atributos escribibles o asignables en la creación para la clase de objetos gestionados creados por la factoría.

6.9.2 Buscador de factoría

Para facilitar la tarea de encontrar una factoría, la Recomendación UIT-T Q.816 define una interfaz buscadora de factoría. (El buscador de factoría es una muestra de diseño común en las aplicaciones CORBA.) Esto permite a un cliente encontrar fácilmente una factoría interactuando con un negociador bien conocido que conozca a todas las factorías presentes en un sistema gestionado.

6.10 Tipos de valores de clase de objeto gestionado

Cada clase de objeto gestionado que cumple con estas directrices hereda una operación de la clase objeto gestionado base, que devuelve todos o algunos de los subconjuntos de los atributos en un único *valuetype*. (CORBA 2.3 introduce el concepto de los *tipos de valores (value type)*, objetos que se pasan por valor en lugar de por referencia.) No sólo debe la implementación de objetos gestionados soportar esta característica, sino que el IDL que describe el objeto gestionado debe incluir un tipo de valor con atributos públicos para cada uno de los atributos soportados por el objeto gestionado. Estas directrices definen un *ManagedObjectValueType* base, y los tipos de valor

definidos para los objetos gestionados deben finalmente derivar desde este tipo de valor base. Los tipos de valor definidos para los objetos gestionados deberán normalmente seguir el modelo de herencia de la interfaz de los objetos gestionados, pero, debido a que los tipos de valores de CORBA soportan sólo la herencia sencilla, esto no será siempre posible. Sin embargo, esto no es una limitación seria. Significa simplemente que los tipos de valores definidos para interfaces que utilizan la herencia múltiple tendrán que heredar simplemente desde uno de los tipos de valores superiores, y los demás atributos tendrán que ser añadidos y mantenidos manualmente.

Como un ejemplo, supóngase que la interfaz del objeto gestionado Equipment hereda directamente desde la clase base *ManagedObject,* y tiene, entre otros, un atributo de función de acceso llamado *userLabelGet* que devuelve un tipo *UserLabelType*. El IDL que describe el tipo de valor para el objeto gestionado Equipment será análogo a:

```
valuetype EquipmentValueType : ManagedObjectValueType {
   public UserLabelType    userLabel;
   …                              // otros atributos
};
```

El nombre del tipo de valor es el nombre de la interfaz con "ValueType" añadido. Téngase en cuenta, además, que el nombre del atributo público en el tipo de valor es el nombre del método en la interfaz del objeto gestionado utilizado para acceder al atributo, sin el "Get" añadido. Se deberá seguir este convenio para todos los atributos en los tipos de valor. El tipo del atributo es el mismo que el tipo devuelto por la función de acceso al atributo.

El código en el lado cliente que pretende recuperar valores de los atributos de un objeto equipment puede parecerse a algo como:

```
ManagedObjectValueType   moValue;
EquipmentValueType       eqValue;
Equipment                eq;
eq = …   // código que asigna a eq un aproximador CORBA que representa un
         // objeto equipment.
moValue = eq.getAttributes();
eqValue = (EquipmentValueType) moValue;  // ajusta el retorno al tipo adecuado
System.out.println("User Label = " + eqValue.userLabel); // imprime etiqueta
```

Cuando se compila el IDL a un lenguaje de programación orientado al objeto, tanto las interfaces (en este caso Equipment) y los tipos de valores (*ManagedObjectValueType* y *EquipmentValueType*) serán trasladados a clases. Para las interfaces, las clases son en realidad aproximadores. Cuando se invocan métodos sobre ellos, hacen uso de ORB para enviar hacia atrás al servidor. Las clases trasladadas desde tipos de valor, sin embargo, no son aproximadores. Son simplemente objetos locales.

Cuando el cliente invoca la llamada al equipo aproximador para obtener atributos, la respuesta desde el servidor será un *EquipmentValueType*. Cuando la ORB recibe esto, creará un ejemplar local de un objeto *EquipmentValueType* con los valores de atributo recibidos desde el servidor. Ya que el tipo de retorno para el método *attributesGet()*, definido en la interfaz base del objeto gestionado, es *ManagedObjectValueType*, la referencia al ejemplar *EquipmentValueType* se pasa hacia atrás como una referencia del tipo *ManagedObjectValueType*. Esto funciona, ya que *EquipmentValueType* se deriva desde *ManagedObjectValueType*. Sin embargo, para acceder a atributos que son específicos de *EquipmentValueType*, el cliente debe estrechar la referencia ajustándola al tipo *EquipmentValueType*.

Ya que el procesado detrás de la escena que lleva a cabo la ORB es un poco complicado, la alternativa sería utilizar listas de los tipos *any (cualquiera)* de CORBA para soportar los valores del atributo. Este tratamiento, sin embargo, necesitaría aun un mayor procesamiento. Además los tipos *any* serían mucho más complicados para los programadores. Como se presenta en el ejemplo anterior, el uso de los tipos de valor es en realidad bastante sencillo.

6.11 Constantes

Los sistemas de gestión de red necesitan ser capaces de intercambiar información con significados previamente acordados. Por ejemplo, una notificación de cambio de estado con una causa probable igual a "1" puede querer decir que fue originada por una pérdida de señal, mientras que si es "2" significa una pérdida de trama, etc. Es bastante simple definir una enumeración o un conjunto de valores enteros a pasar a través de una interfaz dentro de algún campo, pero es un poco más complicado hacer que este mecanismo sea ampliable para muchos grupos, posiblemente trabajando en paralelo. Al mecanismo utilizado por estas directrices para estos casos se le llama "identificador universal" (UID, *universal identifier*).

Un UID es una estructura de datos con dos campos. El primero es una cadena dispuesta para contener el nombre previsto de un módulo IDL que contenga las constantes definidas para algún campo. El segundo es un entero corto (16 bits) con signo que contiene el valor. Por ejemplo, para enviar un valor de "pérdida de señal" en un campo de causa probable, un sistema construirá una estructura UID con una cadena *moduleName* que sea igual a "itut_x780::ProbableCauseConst" y un valor entero igual a 29. (El anexo B contiene las constantes definidas en estas directrices. En él hay un módulo llamado "ProbableCauseConst" que contiene una constante llamada *lossOfSignal* con un valor de 29.)

Téngase en cuenta que este formato es el único utilizado para los valores de constantes que se utiliza dentro de estas directrices. No se utilizan valores "locales".

Estos convenios serán seguidos en la definición de constantes para un modelo de información:

1) Los valores constantes se definirán en módulos separados, uno para cada conjunto de constantes definido para un campo particular. Estos submódulos estarán contenidos dentro del módulo de nivel superior que contiene las demás construcciones definidas para el modelo de información.

2) El nombre del módulo será el nombre del campo añadido con "Const". Por ejemplo, los valores para el campo *probableCause* (definido como tipo UIDType) están contenidos dentro de un módulo llamado "ProbableCauseConst".

3) Las constantes definidas dentro del submódulo deben ser del tipo *const short (constante corta)*. Por ejemplo:

```
const short lossOfSignal = 29;
```

4) Para reducir el tamaño y complejidad del fichero IDL principal, las constantes pueden ser guardadas en un fichero separado. Aun, si las constantes están en un fichero separado, los submódulos estarán dentro de una declaración de módulo IDL con el mismo nombre que el módulo del fichero principal. El fichero principal tendrá una declaración de precompilador *include (incluir)* al comienzo del fichero, para que se incluyan las constantes en cualquier proceso de compilación.

5) El submódulo contendrá también una constante cadena llamada "moduleName" que contenga el nombre previsto para ese módulo. Por ejemplo:

```
module itut_x780 {
        …
        module ProbableCauseConst {
                const string moduleName = "itut_x780::ProbableCauseConst";
                …
        }; // fin del módulo ProbableCauseConst
        …
}; // fin del módulo itut_x780
```

Esto es en realidad una facilidad para permitir a los programadores hacer referencias al nombre del módulo por medio de una constante en lugar de por nombres de cadenas de módulo codificadas en firma.

Téngase en cuenta que otros modelos de información pueden ampliar los valores de causa probable. Puede haber, por ejemplo, un módulo "itut_m3120::ProbableCauseConst" con valores adicionales para el campo causa probable. Estos módulos pueden aun reutilizar el valor 29. El UID será aún único ya que los nombres del módulo diferirán.

6.12 Registro

El IDL de CORBA requiere que todos los identificadores dentro de un módulo sean únicos. Esto quiere decir que, mientras que un nombre de módulo sea único, todos sus contenidos serán nombrados de forma única. El IDL de CORBA define también una declaración de compilador IDL *pragma* que puede ser utilizada para definir un único prefijo para los identificadores de módulo cuando se registran en la interfaz almacén de CORBA, un directorio central de la información de la interfaz utilizado por las ORB de CORBA. Este marco de trabajo requiere que los documentos IDL contengan una declaración de prefijo *pragma* que utilice la organización de los nombres de dominio de Internet como un prefijo para los nombres que contiene.

Esto elimina la necesidad de registrar cada una de las construcciones.

6.13 Versión de las especificaciones del IDL de CORBA

Cuando se utiliza CORBA, una interfaz de gestión se especifica como una o más interfaces de objeto definidas utilizando IDL. Inevitablemente, las interfaces de gestión cambian. La adición de una nueva interfaz de objeto CORBA a una interfaz de gestión es directa. La nueva interfaz CORBA necesita simplemente ser definida en IDL, y añadirla a la especificación que identifica a las interfaces de los objetos que tienen que ser soportados en esa interfaz de gestión en concreto.

La actualización de la interfaz de objeto CORBA, es sin embargo, un poco más complicada. Estas directrices establecen una prioridad en la compatibilidad hacia atrás. Por tanto, en la ampliación de una interfaz existente de un objeto gestionado se aplican las reglas siguientes. Se debe tener en cuenta que estas reglas se aplican sólo a extensiones hechas a una clase base, que no devienen en un cambio de la finalidad pretendida del objeto. Es decir, la nueva clase modela el mismo recurso que la clase antigua, simplemente tiene algunas capacidades adicionales.

1) El nombre de la interfaz del nuevo objeto será el mismo que el de la interfaz existente con la letra "R" y un número añadidos, comenzando con "1". Las subsiguientes extensiones irán incrementando el número. Por tanto, la extensión de una interfaz para los objetos gestionados "Equipment" resultará en una interfaz llamada "EquipmentR1."
2) Se definirá la nueva interfaz dentro del mismo nombre de módulo que el de la interfaz existente. (Los módulos CORBA realmente son espacios de nombres, y se pueden extender a través de varios ficheros.)
3) La nueva interfaz heredará desde la interfaz existente.
4) En la nueva interfaz no se podrán suprimir o modificar las capacidades heredadas de la interfaz existente. Si se tiene que modificar una definición de operación, se debe definir una nueva operación. El nombre de la nueva operación será el mismo que el de la operación existente con la letra "R" y un número añadido, comenzando por el "1". Las extensiones subsiguientes irán incrementando el número.
5) El valor del campo *kind* utilizado en las vinculaciones de nombre, seguirá siendo determinado por una constante en los módulos de vinculación de nombres referenciados cuando se creó el objeto. Cualesquiera de las vinculaciones de nombre válidas para la interfaz existente serán válidas para la nueva interfaz. Esto es, un módulo de vinculación de nombres para un objeto Equipment será también válido para un objeto EquipmentR1, aun si el valor del módulo *subordinateSubclassesAllowed* es falso.

6) Las referencias a las nuevas interfaces serán del tipo más específico. (En caso de no serlo, no se podrá acceder a las nuevas capacidades.) Asimismo, el valor del atributo *objectClass* presentado por un objeto de la nueva clase será del tipo más específico. CORBA proporciona algunos medios para la determinación de la clase real de una referencia sobre la base de la información contenida en la IOR.

Por ejemplo, considérese la siguiente interfaz de objeto:

```
interface Foo {
        void action(in int A, in int B);
}
```

Se puede extender la acción de la siguiente forma:

```
interface FooR1: Foo {
        void actionR1(in int A, in int B, in int C);
}
```

La acción antigua será todavía una operación válida.

Se utilizará un método similar, en el que al nombre se le añade una "R" y un número incrementado, cuando se revisen otras definiciones IDL existentes, incluidas definiciones de constantes, definiciones de tipos, y definiciones de tipos de valores.

7 Traslación desde GDMO

Esta cláusula proporciona directrices para la creación de modelos de información IDL partiendo de modelos de información existentes descritos utilizando las GDMO. Las subcláusulas siguientes describen cómo se debe trasladar cada una de las plantillas GDMO al IDL de CORBA.

7.1 Clases de objetos gestionados

Se tendrá que trasladar cada una de las clases de objetos gestionados de una especificación GDMO a una interfaz de objeto gestionado. Las traslaciones de las clases de objetos gestionados derivadas a partir de la clase Tope de GDMO heredarán desde la interfaz *ManagedObject* del IDL de CORBA. Las traslaciones de clases no derivadas directamente desde la Tope, heredarán desde la traslación de cualesquiera de las clases de la que ha derivado. Todas las interfaces de objetos gestionados deben heredar directa o indirectamente desde la interfaz *ManagedObject.* La herencia múltiple está permitida sujeta a las reglas del IDL de CORBA. Téngase en cuenta, sin embargo, que estas reglas difieren de las del CMIP. En particular, CORBA no permite la herencia de una operación o de un atributo desde varios orígenes, a no ser que ellos a su vez los hayan heredado desde el mismo origen común. Si una traslación de herencia múltiple desde el CMIP no cumple con las reglas de CORBA, quien haga la traslación tendrá que optar por heredar desde una superclase y añadir manualmente las demás capacidades desde otra clase. Otra opción es la de modificar las superclases conflictivas de forma que hereden las capacidades conflictivas desde un origen común. Esto, desde luego, hará necesaria la redefinición de estas superclases.

La incapacidad de heredar desde una potencial superclase significa también que puede ser necesario trabajo manual si se modifica la superclase potencial o alguna de sus superclases. Un aspecto más importante es que el polimorfismo de CORBA está basado en la herencia. Si la subclase no hereda desde una clase, no puede ser polimórfica de ella. Desafortunadamente esta es una limitación de CORBA, no de estas directrices.

Los atributos, acciones y notificaciones en lotes obligatorios y condicionales son trasladados a operaciones en la interfaz de acuerdo con las directrices que siguen. Un comentario que precede a la interfaz describirá las condiciones bajo las que las capacidades de un lote condicional tienen que ser soportadas por un ejemplar, en función de la cláusula PRESENT IF para ese lote. Téngase en cuenta que CORBA no permite la redefinición de una capacidad presente en una superclase. Por tanto, si

una capacidad está definida como condicional en una superclase, no puede ser redefinida como obligatoria en una subclase. (Como se ha descrito antes, las capacidades se dice que son condicionales cuando provocan una excepción *NO<nombre del lote>*. Esta excepción no se puede suprimir en una subclase. La mejor alternativa será un comentario que indique que la subclase no deberá provocar la excepción. Otra alternativa será no considerar la herencia y añadir manualmente la capacidad, al tiempo que se la hace obligatoria. Sin embargo, esto puede acarrear problemas con el polimorfismo y la actualización manual.)

No es preciso el registro de interfaces particulares.

7.2 Lotes

Desafortunadamente, el IDL no proporciona una forma para definir lotes en otra forma que no sea la traslación de un lote a una interfaz. Sin embargo, esto devendría en un número elevado de interfaces extra y aumentaría la complejidad de la interfaz CORBA. En su lugar, estas directrices incluyen el concepto del soporte condicional para grupos de capacidades.

Como se ha descrito antes, siempre que se incluye un lote GDMO en una clase de objeto gestionado, la traslación de esa clase a una interfaz IDL incluye una traslación de cada una de las plantillas del lote.

Los atributos GDMO que son parte de un lote condicional se trasladarán a operaciones de acceso, cada una con una cláusula *raises* que incluya la excepción definida para ese lote. Las acciones GDMO que son parte de un lote condicional se trasladarán a una operación que tenga también una cláusula *raises* que incluya la excepción definida para ese lote. Las notificaciones GDMO que son parte de un lote condicional se trasladarán a una declaración de macro *CONDITIONAL_NOTIFICATION*.

La cláusula *present if* en la declaración de lote condicional del objeto GDMO será trasladada a un comentario que precede la traslación de IDL del objeto.

Las traslaciones desde CMIP pueden encontrar también problemas cuando la misma capacidad está incluida en varios lotes condicionales. Se seguirán las siguientes reglas:

1) Si la capacidad es obligatoria en uno de los orígenes y condicional en otro, será obligatoria en la clase trasladada.

2) Si la capacidad es parte de varios lotes condicionales, la operación trasladada incluirá una excepción para cada lote. Sólo se provocará una excepción si ninguno de los lotes está presente, y entonces se puede provocar cualquiera de las excepciones.

3) Si se incluye el mismo lote condicional desde varias superclases, la condición bajo la que se incluyen los lotes en la nueva clase es una "OR" lógica de las condiciones en la superclase.

4) Las notificaciones que son parte de varios lotes se trasladan a una única declaración de macro. Si alguno de los lotes es obligatorio, se utiliza la declaración de macro *MANDATORY_NOTIFICATION*. En otros casos se utiliza la declaración de macro *CONDITIONAL_NOTIFICATION,* y se listan todas las excepciones del lote.

Si se presenta una plantilla GDMO en varios lotes condicionales incluidos en un único objeto, el modelador puede querer considerar la capacidad como obligatoria, o definir un nuevo lote condicional para la capacidad.

Téngase en cuenta que el uso de excepciones para la representación de lotes sólo soporta a los lotes condicionales. Si en una clase GDMO están presentes varios lotes obligatorios, no serán distinguibles en la interfaz trasladada.

Las declaraciones de comportamiento que acompañan a una definición de lote se trasladarán a comentarios en las definiciones de la interfaz de los objetos IDL trasladados desde los objetos GDMO incluidos en el lote.

No es necesario el registro de los lotes.

7.3 Atributos

Como se ha descrito anteriormente, las clases de objetos gestionados GDMO listan los lotes que tienen que ser incluidos en la definición de clase. El lote lista entonces los atributos, acciones y notificaciones que forman ese lote. Al trasladar una clase de objeto gestionado, cada plantilla de los lotes incluidos se trasladará a una operación en la interfaz del objeto gestionado, y la mayor parte de estas incluirán definiciones de atributos.

Los atributos que soportan las capacidades *GET (OBTENER)* tendrán una operación <Nombre Atributo>Get definida para ellos. El tipo de retorno para la operación será una traslación de la sintaxis ASN.1 del atributo.

Los atributos que soportan las capacidades *REPLACE (REEMPLAZAR)* tendrán una operación <Nombre Atributo>Set definida para ellos. El tipo del parámetro de entrada para la operación será una traslación de la sintaxis ASN.1 del atributo.

Los atributos que soportan las capacidades *ADD (AÑADIR)* tendrán una operación <Nombre Atributo>Add definida para ellos. Los atributos que soportan las capacidades *REMOVE (REMOVER)* tendrán una operación <Nombre Atributo>Remove definida para ellos. El tipo del parámetro de entrada para estas operaciones será una traslación de la sintaxis ASN.1 del atributo.

Los atributos que soportan la capacidad *set-by-create (asignar en creación)* aceptarán un valor inicial para el atributo en los métodos de creación en factoría, pero no tendrán una operación *SET*. (El método creación en factoría aceptará también valores para los atributos que son asignables, pero no para atributos que sólo son legibles.)

Los valores por defecto están definidos como constantes dentro de una interfaz. El identificador de la constante será <Nombre Atributo>Default *(valor por defecto)*. La interfaz puede tener también una operación para asignar al atributo su valor por defecto, o el cliente puede utilizar la operación *SET* con la constante por defecto. La operación asignar valor por defecto, será llamada <Nombre Atributo>SetDefault y no aceptará parámetros y devolverá *void (vacío)*. El IDL de CORBA permite que las constantes sean definidas para tipos de base y tipos enumerados solamente, de modo tal que si el tipo de atributo es complejo, no se podrá definir ningún valor por defecto. En esos casos, se deberá definir la operación de asignación por defecto y un comentario asociado con la operación de asignación por defecto describirá el valor por defecto.

Algunas otras pocas capacidades GDMO relacionadas con el atributo no pueden ser vueltas a crear con el IDL. Los atributos GDMO con una cláusula *DERIVED-FROM* deberán tener las capacidades del otro atributo añadidas manualmente a la especificación de la interfaz. (Se utilizará la sintaxis del atributo derive-from.) Las reglas de concordancia están definidas por las imposiciones del lenguaje del servicio de operación de objeto múltiple, que es parte de los servicios RGT de CORBA definidos en UIT-T Q.816. Estas reglas de concordancia dependen simplemente del tipo básico del atributo. No hay reglas de concordancias específicas según el atributo. No se soportan valores iniciales, valores permitidos ni valores solicitados.

A veces tendrá sentido definir un tipo del IDL para cada atributo. Aun si el atributo es un tipo simple, se puede utilizar una declaración *typedef* del IDL para definir un tipo para él. Un comentario que preceda a la definición de tipo de un atributo es el mejor sitio para poner una traslación de una declaración del comportamiento del atributo. En otro caso, la declaración de comportamiento puede ser trasladada a un comentario que preceda a la operación de acceso al atributo en la interfaz del objeto.

Siempre que sea posible se utilizarán los atributos estándar definidos en estas directrices. Véase la cláusula 6.3.5.

No es necesario el registro de los atributos.

7.4 Grupos de atributos

Estas directrices no soportan el concepto de grupos de atributos. Los grupos de atributos GDMO no tienen una traslación equivalente.

7.5 Acciones

Las acciones se trasladarán a operaciones IDL. Los parámetros de entrada, los parámetros de salida y el tipo de retorno para la operación se trasladarán desde la sintaxis ASN.1 de las acciones de entrada y salida. Esto es, la sintaxis de entrada se trasladará a los parámetros *entrada* de IDL, mientras que la sintaxis de salida se trasladará a una mezcla de parámetros de salida y de valor de retorno. Los parámetros IDL *inout* (*entrada/salida*) se pueden utilizar donde sean pertinentes. Asimismo, se definirán excepciones para devolver valores de condiciones de error, en lugar de devolver uniones de valores normales y de error.

Las acciones GDMO con un modo *sin confirmar* (aquellas a las que las falta la cláusula *MODO CON CONFIRMACIÓN*) pueden ser trasladadas a métodos con la palabra clave de IDL *oneway (sentido único)* precediendo al tipo de retorno. Tales operaciones deben tener, sin embargo, un tipo de retorno *vacío* sin parámetros *salida* o *entrada salida*. Las operaciones IDL sin la palabra clave *oneway* son con confirmación.

7.6 Notificaciones

Estas directrices definen el equivalente IDL de las 15 notificaciones que se tratan en UIT-T X.721, que son las notificaciones utilizadas en la mayoría de los modelos de información GDMO. Típicamente, las notificaciones en los lotes GDMO serán simplemente trasladadas a una declaración de macro de notificación en cada interfaz que incluya el lote. Se utiliza una declaración *MANDATORY_NOTIFICATION* si la notificación es parte de un lote obligatorio y una declaración *CONDITIONAL_NOTIFICATION* si es parte de un lote condicional.

No se soporta la correspondencia de atributos de objeto a campos de notificación dentro de una declaración de notificación. Si es necesaria alguna correspondencia especial, se documentará con un comentario. No se soportan contestaciones a las notificaciones.

Si hay que definir una nueva notificación, se definirá sobre una interfaz llamada "Notificaciones" dentro del nuevo módulo del modelo de información. (Esto no implica que esta interfaz deba heredar de la interfaz itut_x780::Notifications.) El nombre de la operación será el nombre de la notificación. Se trasladarán los parámetros de la operación partiendo de la sintaxis de la información de la notificación. El tipo de retorno en la operación de notificación será vacío, y sólo deberá tener parámetros *entrada.* La Recomendación UIT-T Q.816 proporciona información sobre la forma en la que se coloca la información dentro de una notificación estructurada. Téngase en cuenta que no se necesitan los ID de los atributos. En su lugar, los parámetros se identifican con un nombre y un tipo de datos. El nombre previsto de la interfaz y la operación de notificación pueden utilizarse entonces dentro de las declaraciones de macro de la notificación. Se suministra un nuevo nombre a esta notificación (añadiendo un "R1", etc.) para permitir al sistema de gestión crear una interfaz de herencia múltiple para recibir las alarmas que incluyen las versiones antigua y extendida.

Si se necesita extender una notificación, se debe hacer definiendo una nueva operación. La nueva operación contendrá los mismos parámetros que la antigua. Por ejemplo, el IDL siguiente amplía la alarma equipment añadiendo un parámetro llamado "newData" del tipo "newType".

```
module newModule {
…
        interface Notifications {
                void equipmentAlarmR1 (
                        in ExternalTimeType             eventTime,
                        … (other equipmentAlarm parameters)
                in SuspectObjectSetType         suspectObjectList,
                        in newType                      newData);
        }
}
```

7.7 Comportamientos

Se trasladarán las plantillas de comportamiento GDMO a comentarios IDL formateados que precedan inmediatamente a la construcción IDL con la que se asocia cada comportamiento. Se trasladarán los comportamientos de los atributos a comentarios IDL precediendo a la definición de tipo para el tipo de atributo. Se trasladarán los comportamientos de lote a comentarios IDL precediendo a la excepción definida para el comentario.

7.8 Vinculaciones de nombres

Se trasladará cada vinculación de nombres GDMO a un módulo de vinculación de nombres IDL como se define en la cláusula 6.8. Las diversas construcciones en la vinculación de nombres GDMO se trasladarán como sigue.

Se asignará el nombre de la clase superior en la vinculación de nombres al valor de la constante *superiorClass* del módulo de vinculación de nombres. Si la cláusula de clase superior de GDMO tiene un modificador *AND SUBCLASSES (Y SUBCLASES)*, el valor de la constante de vinculación de nombres IDL *superiorSubclassesAllowed* será *true*. En otro caso será *false*.

Se asignará el nombre de la clase subordinada en la vinculación de nombres al valor de la constante *subordinateClass* del módulo de vinculación de nombres. Si la cláusula de clase subordinada de GDMO tiene un modificador *AND SUBCLASSES*, el valor de la constante de vinculación de nombres IDL *subordinateSubclassesAllowed* será *true*. En otro caso será *false*.

Si la vinculación de nombres de GDMO tiene una cláusula *CREATE (CREACIÓN)*, el valor de la constante de vinculación de nombres IDL *managersMayCreate* será *true*. En otro caso será *false*.

Si la vinculación de nombres de GDMO no tiene cláusula *DELETE (SUPRESIÓN),* el valor de la constante de vinculación de nombres *deletePolicy* será *notDeletable*. Si tiene una cláusula *DELETE* sin modificador, o con un modificador *ONLY-IF-NO-CONTAINED-OBJECTS (sólo si no contiene objetos)*, el valor de *deletePolicy* será *deleteOnlyIfNoContainedObjects*. Si tiene una cláusula *DELETE* con un modificador *CONTAINED-OBJECTS (contiene objetos)*, el valor de *deletePolicy* será *deleteContainedObjects*.

Si la cláusula de creación de vinculación de nombres tiene un modificador *WITH-AUTOMATIC-INSTANCE-NAMING (con denominación automática de ejemplares)*, la operación de creación de factoría de objetos gestionados, definirá el parámetro del nombre como *entrada salida*, e incluirá un comentario que indique que el cliente puede presentar un nombre mulo, y en este caso la factoría elegirá un nombre y lo devolverá.

No es posible la creación de un objeto copiando un conjunto de valores de atributos desde un objeto de referencia con un método de factoría fuertemente tipificado, ya que no hay forma de que la factoría comunique qué valores copiará y cuáles de los parámetros de operación deberá utilizar. Se podría definir una operación fuertemente tipificada que copie todos los valores desde una referencia, aunque su utilidad es limitada. Se puede definir en una factoría una operación débilmente tipificada que aceptase un objeto de referencia, así como una lista parcial de atributos, pero la dificultad de su implementación no parece que compense a los beneficios. Por tanto, no se soporta la traslación del

modificador *WITH-REFERENCE-OBJECT (con objeto de referencia)* en una cláusula de creación de vínculos de nombres.

Se trasladarán los parámetros en las cláusulas de creación a excepciones *CreateError*. Esto puede requerir la definición de un nuevo valor para el ID del error. Se podría poner un comentario en el módulo IDL de vinculación de nombres haciendo notar qué ID de error de excepción *CreateError* es aplicable a objetos creados con esa vinculación de nombres. Si no es posible trasladar un parámetro de cláusula de creación a una excepción *CreateError,* otra alternativa menos deseable es la definición de una nueva factoría y trasladar el parámetro a una excepción en una operación de esa factoría. Sin embargo, debido a la naturaleza de la finalidad general de la excepción *CreateError,* será escasa su necesidad. (Véase más acerca de los parámetros más adelante.)

7.9 Parámetros

Los parámetros GDMO proporcionan extensibilidad a los modelos de información GDMO. Se utilizan plantillas de parámetros para aumentar una especificación existente en las áreas de notificaciones, acciones (peticiones, respuestas y fallos) y errores específicos cuando se definen subclases. Las definiciones GDMO de todas las notificaciones y de muchas acciones contienen un campo de extensibilidad que las subclases definen más ampliamente (si es necesario). En el caso de errores específicos se utilizan errores específicos de clase para aumentar el error general "fallo de proceso" en el CMIP. El formato de esta información es a menudo una lista de pares nombre-valor, en el que el nombre define el tipo de datos del valor.

La traslación de parámetros GDMO a IDL proporciona una buena oportunidad para hacer a las extensiones actualmente definidas que han sido encontradas como útiles con muchas clases de objetos, una parte "normal", fuertemente tipificada del modelo. Por ejemplo, se han incluido en las notificaciones definidas en el IDL tres parámetros GDMO definidos para las alarmas. (Los tres parámetros son: "efecto de alarma en servicio", "lista de objetos sospechosa" y "alarma reanudada".)

Hay varias palabras clave utilizadas en las plantillas de parámetros GDMO para la especificación de la semántica de las extensiones. Más adelante se trata de la traslación de las diversas capacidades de extensión disponibles con plantillas de parámetros basadas en estas palabras clave.

7.9.1 Información de acción (ACTION-INFO) y respuesta a acción (ACTION-REPLY)

Manteniéndose en línea con la fuerte ligazón al tipo recomendada en el marco de trabajo, los parámetros GDMO con las palabras clave "ACTION-INFO" en la plantilla, no se trasladan como un campo de extensión. En su lugar, se convierte en subclase una nueva interfaz partiendo de una interfaz existente que especifica la acción, pero añade las extensiones como parámetros "entrada" normales de ese método. El nombre del parámetro IDL se tomará entonces desde el nombre del parámetro, y el tipo de datos del parámetro se trasladará desde la sintaxis de los parámetros GDMO. De forma similar, los parámetros "ACTION-REPLY" se trasladarán a parámetros "salida" en la operación.

El método anterior implica que no se soporta la posterior adición de un parámetro a una operación IDL existente. En su lugar, el modelador de información puede utilizar los procedimientos más convencionales proporcionados por CORBA para la extensión de una interfaz, tales como la conversión de una interfaz de objeto a una subclase y definiendo en ella un nuevo método, con parámetros adicionales *entrada* y/o *salida,* o excepciones adicionales. Véase la cláusula 6.13 para directrices sobre este tema.

7.9.2 Información de evento (EVENT-INFO) y respuesta a evento (EVENT-REPLY)

En casos en los que ya se han definido parámetros "EVENT-INFO", se les traslada a parámetros "entrada" normales en las operaciones IDL utilizadas para el transporte de una notificación. Estas directrices no soportan respuestas a notificaciones, por tanto, no hay traslación para los parámetros "EVENT-REPLY".

Debido a que este marco de trabajo ya define un conjunto de notificaciones, la traslación de los parámetros EVENT-INFO puede significar la redefinición de una de las operaciones de notificación. Véase la cláusula 7.6.

Sin embargo, en muchos de los casos será preferible la reutilización de una definición de notificación existente. En los casos en los que están predefinidas las extensiones GDMO, como en los de la información de alarmas, se incluirán en las especificaciones IDL de la notificación trasladada. El marco de trabajo de notificación IDL, sin embargo, también soporta un campo "información adicional", que es una lista de pares nombre-valor débilmente tipificada. Se puede utilizar esto para añadir información a estas notificaciones previamente definidas. No cambiará el tipo de evento de la notificación. Sin embargo, la nueva interfaz de objeto gestionado que necesite utilizar la extensión para un parámetro específico, debe tener en cuenta la utilización de este parámetro en los comentarios. Desafortunadamente, no hay otros mecanismos, excepto la utilización de macros mostrada anteriormente, para la especificación de qué notificaciones están soportadas por qué objetos, y esto no soporta tampoco la especificación de parámetros. La ventaja de utilizar el mismo tipo de notificación es la de permitir a los gestores la recepción de notificaciones y que no se tengan que preocupar con el registro de un nuevo tipo de notificaciones. Si no son entendidas las extensiones debido a diferentes versiones del gestor y del agente, entonces se desecha la información adicional.

A continuación se describe la especificación de las extensiones para la información adicional.

Las notificaciones definidas por este marco de trabajo incluyen un campo llamado "additionalInformation" que se parece mucho al campo "additionalInformation" de las notificaciones CMIP. La sintaxis IDL del campo "additionalInformation" es del tipo "AdditionalInformationSetType":

```
struct ManagementExtensionType {
        UIDType id;             // identifica el tipo de información
        any     info;           // el tipo dependerá del identificador
};
typedef sequence <ManagementExtensionType> AdditionalInformationSetType;
```

Los parámetros con las palabras clave EVENT-INFO son trasladados definiendo un identificador único (UID) para cada parámetro. Véase la cláusula 6.11 para detalles acerca de esto. Brevemente, sin embargo, el modelador define un submódulo llamado "AdditionalInformationConst" en el que se define una constante de tipo de valor "corto". Los nombres de estas constantes son los nombres de los parámetros GDMO. El valor de cada constante se puede también derivar desde el GDMO, en base posiblemente al último número del registro del parámetro. En otro caso, se elegirá un entero específico para las constantes de ese módulo. Esta definición debe incluir también un comentario que indique el tipo de datos del valor que acompaña al UID en el campo "additionalInformation". Como un ejemplo, si el parámetro efecto de alarma en servicio no había sido hecho un miembro normal de la estructura de datos de información de alarmas utilizado por las alarmas en este marco de trabajo, se podría haber trasladado de la siguiente forma:

```
module itut_m3100 {
…
  module AdditionalInformationConst {
  /** Los parámetros de efecto de la alarma en servicio están acompañados por un
  valor booleano en el campo "any" , que indica si ha sido afectado el servicio. */
  const short alarmEffectOnService = 1;
  …
  }; // fin del módulo AdditionalInformationConst
  }; // fin del módulo itut_m3100
```

Una interfaz IDL de objeto gestionado puede entonces identificar las notificaciones que soporta como normales, pero un comentario debería indicar los parámetros que estarán incluidos en la notificación.

7.9.3 Palabra clave de contexto (Context-Keyword)

Los parámetros de la palabra clave de contexto identifican la información que tiene que ser pasada en un campo con nombre en una PDU del CMIP. Este campo con nombre es normalmente una secuencia de estructuras de datos que constan de un identificador y un tipo de datos "any" (cualquiera) que guarda un valor cuyo tipo depende del identificador. En CMIP, estos parámetros de palabra clave de contexto pueden ser pasados en parámetros de acción o en notificaciones. No está soportada por este marco de trabajo la traslación de parámetros de palabra clave de contexto debido a la preferencia por la tipificación fuerte. En su lugar, se debería trasladar información adicional para las acciones a parámetros de acción normales. (Véanse parámetros de ACTION-INFO más arriba.)

Para las notificaciones, excepto para las extensiones (explicadas anteriormente), si los campos están definidos como de tipificado débil, entonces se puede utilizar la misma táctica que para el campo de extensión. Sin embargo, esta táctica no ha sido utilizada en la mayoría de normas de las GDMO. La distinción entre el caso que usa como palabra clave EVENT-INFO versus palabra clave de contexto es que la primera está diseñada para facilitar la extensibilidad al ser posible añadir uno o más parámetros. La táctica recomendada en el caso de extensión múltiple es el uso de EVENT-INFO y, por tanto todas las normas tienen definidos parámetros utilizando esta palabra clave.

7.9.4 Error específico (SPECIFIC-ERROR)

Los parámetros "SPECIFIC-ERROR" son devueltos en mensajes CMIP de fallo de proceso. Indican el resultado anormal de una operación. Hay dos opciones para la traslación de estos parámetros. Primera, pueden ser trasladados a excepciones IDL provocadas por la operación para la cual está definido el parámetro de error específico. El nombre de la excepción se tomará desde el nombre del parámetro GDMO, y los tipos de datos devueltos con la excepción se derivarán desde la sintaxis GDMO del parámetro. Debido a que los parámetros de error específico pueden estar definidos para varias clases de plantillas GDMO, los parámetros de error específico sobre las acciones se deberán trasladar a excepciones provocadas por la acción y los parámetros de error específico sobre los atributos se deberán trasladar a excepciones provocadas por la operación de acceso a atributo. Asimismo, los parámetros de error específico sobre la cláusula "Creación" de una vinculación de nombres se trasladarán a excepciones en la operación de creación en la interfaz factoría. No se traslada un parámetro de error específico en una notificación soportada por este marco de trabajo ya que no están permitidas las respuestas a las notificaciones.

La segunda opción para la traslación de parámetros de error específico es la de trasladar el parámetro a un nuevo punto de código para una de las excepciones estándar definidas por el marco de trabajo. El marco de trabajo define tres excepciones estándar: la excepción *CreateError*, provocada en operaciones de creación de factoría, la excepción *DeleteError*, provocada en operaciones de supresión de objetos gestionados y las excepciones *ApplicationError*, provocadas en todas las demás operaciones de objetos gestionados. La excepción *ApplicationError* devuelve un identificador único que identifica al error específico de la aplicación, así como un texto de explicación. Las excepciones de errores de creación y de supresión amplían esta información añadiendo una lista de los objetos relacionados que pueden estar involucrados, así como los atributos del objeto sobre el que se intentó la operación. La lista de los objetos relacionados puede presentar, por ejemplo, algunos objetos que pueden ser suprimidos antes de que el objeto destino pueda ser suprimido. Los atributos pueden contener información sobre el estado del objeto que esté relacionada con el error.

Se deberá utilizar siempre que sea posible la traslación de un error específico a un punto de código de los utilizados por una de estas excepciones estándar. Debido a que los tipos de datos devueltos en las excepciones son tipos de valor, pueden ser extendidos para los puntos de código específicos. Ya que la operación de supresión es heredada desde la interfaz base del objeto gestionado, los parámetros de error específico que aparecen en las cláusulas de supresión de vinculación de nombres de GDMO, deben ser trasladados a puntos de código de excepción *DeleteError*. Esto se hace de forma análoga a la de los parámetros EVENT-INFO descritos anteriormente. Básicamente, el modelador define un submódulo de error de supresión para las constantes del UID. Las definiciones

de las constantes deben incluir un comentario que indique qué datos se pondrán en los campos "relatedObjects" y "attributeList" que acompañan a un error con ese identificador. Asimismo, si el modelador ha extendido el tipo de valor estándar devuelto por el punto de código, un comentario debe hacer notar el tipo de datos reales devuelto, de forma que el sistema de gestión puede estrechar el tipo y el acceder a la información adicional. El marco de trabajo, de hecho, incluye algunos puntos de código de error de supresión que amplían el tipo estándar de valor del error de supresión.

Finalmente, un comentario acerca de la interfaz IDL de los objetos gestionados indica los valores del error de supresión que pueden ser provocados en una excepción cuando se hace un intento incorrecto de suprimir el objeto. Una traslación de ejemplo es la siguiente:

```
module itut_m3100 {
…
 module DeleteErrorConst {
 /** Se provocan errores de supresión de networkTTPTerminatesTrail cuando se
 intenta suprimir un punto de terminación de pista (TTP, trail termination point)
 antes de ue se haya suprimido la pista.
 Incluye una referencia a la pista en el campo "relatedObjects". */
 const short networkTTPTerminatesTrail = 54;
 …
 }; // fin de módulo DeleteErrorConst
}; // fin de módulo itut_m3100
```

7.10 Tipos de datos ASN.1

Las GDMO utilizan el lenguaje ASN.1 para definir la sintaxis de los atributos, así como la de los parámetros de operación y notificación, de forma que en la conversión de plantillas GDMO a IDL, estas definiciones de sintaxis tendrán que trasladarse también. Esta cláusula da directrices sobre la traslación de la sintaxis ASN.1 al IDL de CORBA.

7.10.1 Tipos básicos

El IDL de CORBA define los tipos básicos siguientes a los que pueden ser trasladados los tipos básicos de ASN.1: cualquiera, booleano, carácter, doble (para números de doble precisión en coma flotante), enumerado (para tipos enumerados), fijo, flotante (para números de simple precisión en coma flotante), largo (para enteros largos), objeto (para referencias a objetos), octeto, corto (para enteros cortos), cadenas, carácter ancho (para caracteres "anchos"), y cadenas anchas (para cadenas de caracteres "anchos").

Este marco de trabajo utiliza[1] el tipo *cadena* (*string*) para todas las cadenas, y define un tipo llamado "Istring" para los casos en los que la cadena pueda contener caracteres internacionales de escape. Istring es una definición de tipos cadena "ancha". Estas son cadenas compuestas de caracteres "anchos" (16-bits).

Además, el servicio de hora de CORBA define un tipo hora referido a un "UtcT", que es utilizado por este marco de trabajo.

7.10.2 Secuencia

El IDL de CORBA soporta la definición de estructuras de datos utilizando la palabra clave *struct (estructura)*, similar a los tipos *sequence (secuencia)* de ASN.1.

---

[1] NOTA UIT-T – Se solicitan contribuciones sobre el uso de la alternativa definición de tipos de Istring para cadenas en vez de wstring, ya que las cadenas pueden transportar juegos de caracteres internacionales cuando se utiliza la negociación de juego de caracteres, soportada por la versión 1.1 del GIOP y posteriores. Los tipos cadena ancha (wstring) se corresponden por vinculaciones del lenguaje CORBA al tipo wstring del lenguaje de programación, que frecuentemente está ligado a Unicode.

7.10.3 Secuencia de

El IDL de CORBA soporta la definición de secuencias de tipos, tanto las básicas como las complejas, fundamentalmente en la misma forma que el tipo *sequence of (secuencia de)* de ASN.1.

7.10.4 Conjunto de

El IDL de CORBA no soporta la definición de tipos de conjuntos complejos como lo hace ASN.1. En su lugar, se trasladan los conjuntos a secuencias IDL. Se seguirá el convenio de terminar el nombre del tipo con "SetType". En el manejo de los valores de estos juegos, se deberán eliminar los duplicados e ignorar el orden.

7.10.5 Elección

El IDL de CORBA soporta la definición de uniones discriminadas, que tienen la misma finalidad que los tipos elección de ASN.1.

Con el fin de simplificar la implementación de normas RGT basadas en CORBA, este marco de trabajo recomienda el uso cuidadoso de las uniones discriminadas. A menudo, al hacer traslaciones desde ASN.1 al IDL de CORBA, se puede simplificar el tipo trasladado sin pérdida de la semántica. Por ejemplo, normalmente una elección entre una cadena y un vacío se puede trasladar simplemente a una cadena. Se puede añadir un comentario para identificar esta posibilidad que diga que posiblemente la cadena puede ser nula. De forma análoga se puede trasladar una elección entre una secuencia de (o conjunto de) y de nulo a justamente una secuencia.

7.10.6 Identificador de objeto (OID)

Este marco de trabajo define un tipo llamado "identificador universal" (UID) que está diseñado para que sea un reemplazo de los OID de ASN.1.

7.10.7 Ejemplar de objeto

Este marco de trabajo soporta dos posibles traslaciones para los tipos ejemplar de objeto de ASN.1. Debido a que cada objeto gestionado tiene un nombre, se puede utilizar el tipo de nombre definido por el servicio de denominación de CORBA. [Este marco de trabajo define un *typedef (definición de tipo)* para los nombres del servicio de denominación de CORBA llamado *NameType (tipo de nombre)*.] También se pueden utilizar las referencias a objetos de CORBA. Ya que todas las interfaces de objetos gestionados deben heredar desde la interfaz *ManagedObject,* se utilizará el tipo *ManagedObject* siempre que sea necesaria una referencia general a un objeto. El modelador puede también utilizar un tipo específico para una clase de objetos gestionados, tal como *Equipment*. Esto tiene la ventaja de hacer un modelo más fuertemente tipificado.

7.10.8 Cadena de bits

Esta cláusula define dos correspondencias de CADENA DE BITS DE ASN.1 para ser utilizadas cuando se trasladan especificaciones GDMO a objetos gestionados CORBA, o para especificar sintaxis para su utilización en nuevas especificaciones de objetos gestionados CORBA.

Las dos correspondencias están justificadas, puesto que las cadenas de bits de ASN.1 se utilizan en diferentes formas.

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

Las dos correspondencias son:

- La CADENA DE BITS de ASN.1 se pone en correspondencia con una secuencia de octetos IDL. En el IDL no se transportan rótulos semánticos.

- Una cadena de bits de ASN.1 se pone en correspondencia con el tipo de valor de IDL, con una representación de estado, funciones de ayuda local para manipular valores, y constantes de rótulos semánticos asociados.

7.10.8.1 Representación simple para la CADENA DE BITS de ASN.1

La representación de estado ASN.1 BIT STRING se pone en correspondencia con un IDL typedef de BitString definido como sequence<octet>. La siguiente declaración se incluye en "itut_x780.idl", para utilización con representaciones de cadenas de bits simples:

```
typedef sequence<octet> BitString;
```

La interpretación de un tipo BitString es la de concordar la codificación BER de BIT STRING (CADENA DE BITS). La secuencia de octetos tendrá un octeto inicial seguido de ninguno, uno, o más octetos subsiguientes. Los bits en el bitstring, que comienza con el primer bit y continúa hasta el bit de cola, estará ubicado en los bits 8 a 1 del segundo octeto, seguido de los bits 8 a 1 de cada octeto correspondiente, seguido por tantos bits como sea necesario en el octeto final, comenzando con el bit 8 (la notación "primer bit" y "bit de cola" se especifica en UIT-T X.208 | ISO/CEI 8824). El octeto inicial codificará, como un entero binario sin signos con el bit 1 como el bit menos significativo, el número de bits no utilizados en el octeto final. El número estará en la gama de cero a siete. Si bitstring está vacío, no habrá octetos subsiguientes y el octeto inicial será cero.

Con esta representación simple, cuando se utiliza GIOP para pasar un valor de cadena de bits en un IDL cualquiera, se enviará un ID referencial para que el receptor (es decir, el usuario orb) sepa interpretar la secuencia de octetos como una cadena de bits codificada.

Las constantes de las cadenas de bits de ASN.1 se representan como una secuencia de octetos que utilizan una variante de la forma "bstring" especificada en la Recomendación X.208, con las comillas simples eliminadas. Las constantes ASN.1 definidas mediante la forma "hstring" han de ser transformadas a esta forma "bstring" modificada para la constante IDL asociada.

Cuadro 2/X.780 – Ejemplo de correspondencia de CADENA DE BITS de ASN.1 simple

ASN.1	IDL
CCDScan ::= BIT STRING scan CCDScan ::= '100110100100001110110'B	typedef ITUT_X780::BitString CCDScanType; const string scan = "100110100100001110110B" ;
G3FacsimilePage ::= BIT STRING -- a seq of bits conforming to Rec. T.4 image G3FacsimilePage ::= '100110100100001110110'B trailer BIT STRING ::= '01'H	typedef ITUT_X780::BitString G3FacsimilePageType; //string constants generated for BIT STRING constants const string image = "100110100100001110110B"; const string trailer = "00000001B";

7.10.8.2 Representación del tipo valor de CADENA DE BITS de ASN.1

La representación de estado para el tipo valor es ITUT_X780::BitString. La representación de las constantes para el tipo BitStringValType es la misma que para el tipo BitString. En el tipo valor definido se incluyen métodos de ayuda (que están en correspondencia con llamadas locales de la función objeto de lenguaje de programación).

```
exception InvalidLength { long length } ;
valuetype BitStringValue {
  public BitString bitStringVal;
  factory initValue (in unsigned long number_of_bits);
  factory InitFromBitString (in BitString desiredValue);
```

```
  // local operations
  short getBit (in unsigned long position)
        raises (InvalidLength);
  void setBit (in unsigned long position, in short new_bit_value)
        raises (InvalidLength);
  unsigned long length ();
  string asString ();   // produces a string with binary values ("1001011B")
  // input a string with binary values ("1001011B")
  void setFromString (in string string_value)
        raises (InvalidString);
};
```

Las constantes IDL se generan en valuetypes (tipos de valores) heredados del BitStringValue (valor de cadena de bits), por rótulos semánticos asociados con las posiciones de bits de denominación. Cada bit de denominación se pone en correspondencia como una constante IDL del tipo largo sin signo con valor igual al valor de desplazamiento, con la cadena de bits. La denominación de la constante es el nombre dado, por las reglas JIDM sin ambigüedades (es decir, "a_n" para la primera utilización n + 1 del identificador "a" en el mismo contexto) si hay colisión de identificadores dentro del alcance de la definición de tipo de valor derivado.

```
const unsigned long <bitname> = <offset>;
```

Cuadro 3 /X.780 – Ejemplo de correspondencias para el tipo valor de cadena de bits

ASN.1	IDL
T0 ::= BIT STRING	valuetype T0Type : ITUT_X780::BitStringValue {} //could have used typedef to BitString for simpler mapping

MessageFlag ::= BIT STRING {
posResp (0), negResp (1),
doNotForward (2) }
	valuetype MessageFlagType : ITUT_X780::BitStringValue {
	const unsigned long posResp = 0;
	const unsigned long negResp = 1;
	const unsigned long doNotForward = 2;
	}

a INTEGER ::= 1
T1::= INTEGER { a(2) }
T2 ::= BIT STRING { a(3), b(a) }
	const long a = 1;
	typedef long T1Type;
	const T1Type a_1 = 2;
	valuetype T2Type : ITUT_X780::BitStringValue {
	const unsigned long a = 3;
	const unsigned long b = a;
	}

NOTA – En algunos casos de traslación, puede ser necesario solucionar el caso de ambigüedades de nombres constantes, aun dentro del alcance de una declaración tipo de valor. Para evitar colisión de denominaciones, se podrán utilizar las reglas de colisión JIDM (demostrada para T1 en el ejemplo) dentro del alcance del campo valuetype.

8 Modismos de estilo para la especificación del IDL de CORBA

Esta cláusula define un conjunto de modismos de estilo para el lenguaje de definición de interfaz (IDL) de la arquitectura de intermediario de petición de objeto común (CORBA, *common object request broker architecture*) para ser utilizada en la definición de interfaces. Tener un conjunto de modismos de estilo conducirá a unas especificaciones del IDL de CORBA que tengan un estilo consistente. Esto puede hacer necesario un trabajo adicional por parte de los editores, pero este esfuerzo adicional tiene como ventaja la mejora de la legibilidad de las especificaciones del IDL de

CORBA. Es importante mantener la idea de que los convenios de estilo benefician al lector, no necesariamente en beneficio del autor.

8.1 Utilizar sangrado consistente

Esta cláusula demuestra el estilo de sangrado que puede ser utilizado en los módulos del IDL. Como un ejemplo, se muestra a continuación un pasaje seleccionado del módulo de no repudio del servicio de seguridad de CORBA.

```
enum EvidenceType {
  SecProofofCreation,
  SecProofofReceipt,
  SecProofofApproval,
  SecProofofRetrieval,
  SecProofofOrigin,
  SecProofofDelivery,
  SecNoEvidence // used when request-only token desired
};
interface NRPolicy {
  void get_NR_policy_info (
    out Security::ExtensibleFamily NR_policy_id,
    out unsigned long        policy_version,
    out Security::TimeT       policy_effective_time,
    out Security::TimeT       policy_expiry_time,
    out EvidenceDescriptorListType supported_evidence_types,
    out MechanismDescriptorListType supported_mechanisms
  );
};
```

8.2 Uso consistente del tamaño de letra en los identificadores

Algunos lenguajes imponen reglas sobre el tamaño de las letras de los textos (como el ASN.1) mientras que otros tienen reglas de-facto. Estas reglas permiten que los lectores distingan identificadores de tipo diferente con el fin de aumentar la legibilidad. IDL no impone el tamaño de las letras, por lo que se proponen las siguientes reglas:

- Las operaciones, parámetros, atributos, miembros y constantes tendrán cada una de las palabras embebidas con su inicio en mayúsculas, excepto para la primera palabra que comienza por mayúscula.
- Todos los demás identificadores tendrán la primera letra de cada palabra embebida escrita en mayúsculas.

```
module CarModule {
  struct EngineType {
    PistonType piston;
    RodType  pistonRod;
  };
  typedef string KeyType;
  enum WontStartReasonType {
    BatteryIsDead,
    NoGas
  };
  exception WontStart {
    WontStartReasonType reasonEngineWontStart;
  };
  interface FordRanger {
    void startEngine(
      in KeyType key
    )
    raises (
      WontStart;
    );
    attribute EngineType engine;
  };
};
```

8.3 Seguir la técnica de la gestión conjunta entre dominios (JIDM) en las IMPORTACIONES

Al comienzo de un módulo que importa un tipo desde otro módulo, crear una definición de tipo local. Éste, lista explícitamente el tipo del que depende el módulo importador desde el módulo exportador. (NOTA – No es necesario que el nombre del identificador local sea el mismo nombre que el del identificador del módulo exportador.) Esta utilización del campo typedef no se debe utilizar para interfaces importadas o definiciones de tipo de valor. Por el contrario, se deben utilizar para ellas nombres de alcance completos.

```
module ImportingModule {
  // Imports
  typedef ExportingModule::SomeType             SomeType;
  typedef ExportingModule::SomeOtherType        SomeOtherType;
  typedef ExportingModule::SomethingElse        SomethingElseType;
  ...
};
```

8.4 Seguir la técnica JIDM para opcional (OPTIONAL) y elección (CHOICE)

En los tipos enumerados y numéricos (enteros y flotantes), utilizar en las correspondencias a IDL de OPTIONAL y CHOICE de ASN, como se prescribe en la Especificación de Traslación [3] de la gestión entre dominios del grupo de gestión conjunta entre dominios del Foro de gestión de redes abiertas y del grupo abierto. A continuación se presenta un ejemplo:

```
// Choice
enum CarChoiceType {
  Ford,
  Cheverolet,
  Chrysler
};
union CarType switch (CarChoiceType) {
  case Ford:            FordType                fordValue;
  case Cheverolet:      ChevroletType           chevroletValue;
  case Chrysler:        ChryslertType           chryslerValue;
}
// Optional
union SunRoofTypeOpt switch(boolean) {case TRUE: SunRoofType the_value};
```

Para las cadenas, secuencias y referencias a objetos, se puede utilizar normalmente un valor nulo para representar los casos opcionales cuando no está presente ningún valor. En los casos en los que hay una diferencia semántica entre un nulo y un no presente, se puede utilizar el método anterior.

Para las estructuras y uniones, se puede utilizar el método anterior, o se puede tomar la decisión de utilizar valores nulos dentro de la estructura para la representación de valores opcionales que no están presentes. Por ejemplo, para una estructura compuesta de dos cadenas, dos nulos podrían representar a un valor opcional que no está presente. Si un valor es opcional será marcado como opcional con un comentario.

Como siempre, es necesario utilizar las directrices con sentido común. Se deberá evaluar la traslación resultante para ver su claridad y utilidad. Si la traslación es demasiado compleja, el modelador puede querer intentar simplificarla.

8.5 Utilizar un sufijo de tipo consecuente

Añádase el sufijo "Type" a todos los tipos IDL. Esto permite que los identificadores de tipo y los miembros utilicen el mismo nombre sin que haya discrepancias, ya que el IDL es insensible al tamaño de las letras. Además, este modismo aumenta la legibilidad al separar los identificadores de tipo de otros identificadores.

8.6 Utilizar un sufijo consecuente para los tipos secuencia

Para las secuencias (ordenadas, duplicados permitidos) utilícese un sufijo "SeqType" para distinguir secuencias de nombres simples.

8.7 Utilizar un sufijo consecuente para los tipos conjunto

Para los conjuntos (no ordenados, duplicados no permitidos) utilícese un sufijo "SetType" para distinguir conjuntos de nombres simples.

8.8 Utilizar un sufijo consecuente para los tipos opcionales

Para los tipos opcionales utilícese un sufijo "TypeOpt" para distinguirlos de los tipos no opcionales.

8.9 Disponer los parámetros de operación de una forma consecuente

Una ordenación consecuente de los parámetros aumenta la legibilidad. Ordene los parámetros según las operaciones, entrada, entrada-salida y seguidamente salida.

8.10 Suponga espacios de identificación no globales

Para reducir las colisiones de nombres y fomentar la reutilización, todos los identificadores estarán previstos para un contesto particular (por ejemplo, módulo e interfaz).

8.11 Definiciones a nivel de módulo

Todas las definiciones de tipos serán a nivel de módulo. El anidado de definiciones de tipo dentro de un contexto inferior conduce a dificultades en la reutilización y duplicación.

8.12 Utilizar las excepciones y los códigos de retorno

Se utilizarán las excepciones para condiciones excepcionales, tales como las situaciones de error. Los retornos normales serán manejados como códigos de retorno y parámetros de salida.

8.13 Operaciones explícitas versus operaciones implícitas

Una operación deberá llevar a cabo una función explícita. El uso de parámetros como una bandera para cambiar implícitamente el comportamiento de la operación, puede llevar a confusiones. Divida cada comportamiento en una operación explícita distinta.

8.14 No cree un número grande de excepciones

Tener un número grande de excepciones aumenta la dificultad para entender una definición de interfaz. Agrupe las excepciones por categorías, o utilice las excepciones estándar (*ApplicationError*, *CreateError*, y *DeleteError*) definiendo, en caso necesario, nuevos puntos de códigos de error para ellas.

9 Cumplimiento y conformidad

Esta cláusula define los criterios que se deben cumplir por parte de otros documentos de normas que pretendan el cumplimiento acorde con estas directrices y las funciones que deben estar implementadas por los sistemas que pretendan la conformidad con esta Recomendación.

9.1 Cumplimiento de los documentos de normas

Cualquier especificación que pretenda el cumplimiento de estas directrices deberá:

1) Derivar (directa o indirectamente) todas las interfaces que modelan los recursos partiendo de la interfaz *ManagedObject* descrita en la cláusula 5.1 y definida para el IDL de CORBA en el anexo A.
2) Definir, para cada clase de objeto gestionado que pueda ser ejemplificado, una interfaz factoría derivada (directa o indirectamente) partiendo de la interfaz *ManagedObjectFactory* descrita en la cláusula 5.2 y definida en el IDL de CORBA del anexo A.
3) Utilizar las constantes definidas en el IDL de CORBA en el anexo B, siempre que sean las apropiadas.
4) Utilizar las notificaciones descritas en la cláusula 5.3 y definidas en el IDL de CORBA en el anexo A, siempre que sean las apropiadas.
5) Adherirse a los convenios para la definición de los objetos gestionados de la RGT basada en CORBA especificada en la cláusula 6.
6) Adherirse a los convenios del IDL especificados en la cláusula 8.
7) Especificar las notificaciones como métodos en una interfaz "Notificaciones" en caso de que no sea aplicable ninguna de las notificaciones definidas en esta Recomendación.
8) Definir y utilizar una excepción NO<nombre lote> para la identificación de atributos y acciones que son parte de un lote condicional.
9) Utilizar las macros definidas en esta Recomendación para la identificación de notificaciones que no son soportadas por un objeto gestionado.
10) Utilizar, donde sean aplicables, las definiciones para los tipos de atributos genéricos que se encuentran en la cláusula 6.3.5.
11) Definir módulos IDL de vinculación de nombres para la identificación de relaciones de contenencia que se puedan permitir.
12) Manifestar, en su cláusula de cumplimiento una referencia al módulo (o módulos) a partir de los que se utilizan otros atributos genéricos.
13) Seguir las reglas de correspondencia de GDMO a IDL definidas en la cláusula 7 en caso de que el modelo IDL sea una traslación desde GDMO.

9.2 Conformidad del sistema

Una implementación que pretenda la conformidad con esta Recomendación deberá:

1) Soportar todas las capacidades de la interfaz *ManagedObject* descritas en la cláusula 5.1.
2) Soportar el comportamiento de la operación creación descrito en la cláusula 6.9.

9.3 Directrices para la declaración de conformidad

Los usuarios de estas directrices deben tener cuidado en la redacción de declaraciones de conformidad. Debido a que los módulos IDL se están utilizando como espacios de nombre, pueden, como lo permiten las reglas del IDL del OMG, ser partidos entre varios ficheros. Por tanto, cuando se extiende un módulo no cambiará su nombre. En su lugar, simplemente se añadirá un nuevo fichero. La simple declaración del nombre de un módulo en una declaración de conformidad, por tanto, no será suficiente para la identificación de un conjunto de interfaces IDL. La declaración de conformidad debe identificar un documento y el año de su publicación, para tener seguridad de que se identifica la versión correcta del IDL.

ANEXO A

El módulo modelo de objeto del IDL de CORBA

```
/* Este código IDL está hecho para ser almacenado en un fichero con nombre "itut_x780.idl"
situado en el camino de búsqueda utilizado en su sistema por los compiladores IDL. */
#ifndef ITUT_X780_IDL
#define ITUT_X780_IDL
#include <CosNaming.idl>
#include <CosTime.idl>
#include <itut_x780Const.idl>
#pragma prefix "itu.int"
/* La mayoría de los comentarios en este fichero están formateados para ser analizados por
un convertidor de IDL a HTML, tales como idldoc u orbacus hidl. */
```

// MÓDULO itut_x780

```
/** Este módulo proporciona las capacidades fundamentales para la implementación de
interfaces de gestión de red y define la interfaz objeto gestionado ("managed object").
Las interfaces a continuación están modeladas a partir de las especificaciones de objetos
gestionados que se encuentran en la especificación del CMIP en el documento X.721 del
UIT-T. */
module itut_x780 {
```

// TIPOS IMPORTADOS

```
        // Tipos importados desde CosNaming
        typedef CosNaming::Name NameType;
        // Tipos importados desde CosTime
        typedef TimeBase::UtcT UtcT;
```

// DECLARACIONES PREVIAS Y DEFINICIONES DE TIPOS

```
        /** Las cadenas internacionales son cadenas de caracteres amplios (unicode de
        16 bits). */
        typedef wstring Istring;
        /** Los conjuntos de cadenas internacionales (Istrings Sets) son grupos de cadenas
        internacionales */
        typedef sequence <Istring> IstringSetType;
        /**  El tipo de texto adicional es utilizado a menudo para transmitir un texto de
        aplicación para la notificación.
        typedef Istring AdditionalTextType;
        /** El tipo disponibilidad (availability) se usa en una secuencia para indicar la
        disponibilidad de un recurso. Se puede indicar ninguna, o cualquier número de
        estas condiciones.
        */
        typedef short AvailabilityStatusType;
        const AvailabilityStatusType inTest = 0;
        const AvailabilityStatusType failed = 1;
        const AvailabilityStatusType powerOff = 2;
        const AvailabilityStatusType offLine = 3;
        const AvailabilityStatusType offDuty = 4;
        const AvailabilityStatusType dependency = 5;
        const AvailabilityStatusType degraded = 6;
        const AvailabilityStatusType notInstalled = 7;
        const AvailabilityStatusType logFull = 8;
        /**  El estado disponibilidad (availability) se utiliza para indicar la
        disponibilidad de un recurso. Está representado por una secuencia de enteros ya
        que algunas de las condiciones pueden existir en un momento.
        */
        typedef sequence<AvailabilityStatusType> AvailabilityStatusSetType;
        /** El tipo estado de copia de respaldo (backed up status) se utiliza para indicar
        si un objeto tiene una copia de respaldo. */
        typedef boolean BackedUpStatusType;
        /** BitStrings (cadenas de bits) se utilizan para retener cadenas de bits. Pueden
        tener cualquier longitud. */
        typedef sequence<octet> BitString;
```

```
/** El tipo estado del control (control status) se utiliza en una secuencia para
indicar el estado del control de un recurso. Se pueden indicar cero o más de
ellos.
*/
typedef short ControlStatusType;
const ControlStatusType subjectToTest = 0;
const ControlStatusType partOfServicesLocked = 1;
const ControlStatusType reservedForTest = 2;
const ControlStatusType suspended = 3;
/**  Se utiliza conjunto del estado del control (control status set) para indicar
el estado del control de un recurso. Está representado por una secuencia de
enteros ya que pueden existir en un momento varias condiciones.
*/
typedef sequence<ControlStatusType> ControlStatusSetType;
/** La hora generalizada es un tipo ASN.1 básico. Normalmente se representa en los
lenguajes de computación como una cadena, pero tiene ciertos formatos que son
analizables. Los tres formatos posibles son: <ol><li>
Sólo hora local. "YYYYMMDDHHMMSS.fff", donde el opcional fff tiene la precisión
de tres decimales, <li>
Sólo hora universal (hora UTC). "YYYYMMDDHHMMSS.fffZ", y <li>
Diferencia entre las horas local y UTC. "YYYYMMDDHHMMSS.fff+-HHMM".
</ol>
Las opciones para representar esto en IDL parecen ser bien una cadena o la
estructura UtcT del servicio de hora de CORBA. UtcT hace un poco más fácil la
comparación de las horas de las diferentes zonas, pero precisa que los sistemas
gestionados conozcan sus zonas horarias. Se ha elegido UtcT.
*/
typedef UtcT GeneralizedTimeType;
/** La hora externa (external time) es una hora generalizada. */
typedef GeneralizedTimeType ExternalTimeType;
/** Declaraciones previas. CORBA utiliza referencias de objeto del tipo "object"
para identificar objetos. Se utilizan en lugar de ejemplares de objetos de ASN.1.
Para las interfaces de gestión de red, todos los objetos heredan de la interfaz
"ManagedObject". */
interface ManagedObject;
/** Un conjunto de MO (MO set) es un conjunto de referencias ManagedObject (objeto
gestionado). */

typedef sequence <ManagedObject> MOSetType;
/** Una secuencia MO (MO seq) es una secuencia de referencias ManagedObject
(objeto gestionado). */
typedef sequence <ManagedObject> MOSeqType;
/** Un conjunto de nombres se define como una secuencia de nombres. */
typedef sequence <NameType> NameSetType;
/** Los ID de notificación son enteros largos. */
typedef long NotifIDType;
/** Esto define un conjunto de ID de notificación. */
typedef sequence <long> NotifIDSetType;
/** El tipo de estado de procedimiento (procedural status type) se utiliza en una
secuencia para indicar el estado del procedimiento de un recurso. Se pueden
indicar cero o más de ellos.
*/
typedef short ProceduralStatusType;
const ProceduralStatusType initializationRequired = 0;
const ProceduralStatusType notInitialized = 1;
const ProceduralStatusType initializing = 2;
const ProceduralStatusType reporting = 3;
const ProceduralStatusType terminating = 4;
/** El conjunto de estados de procedimiento (procedural status set) se utiliza
para indicar el estado del procedimiento de un recurso. Se representa como una
secuencia de enteros ya que algunas de las condiciones pueden existir en un
momento.
*/
typedef sequence<ProceduralStatusType> ProceduralStatusSetType;
/** ScopedName es una cadena. */
typedef string ScopedNameType;
/** Los conjuntos de nombres previstos (scoped name sets) son simplemente
conjuntos de nombres previstos. */
typedef sequence <ScopedNameType> ScopedNameSetType;
/** En CORBA se utilizan cadenas que contienen nombres previstos para identificar
clases de objetos, (en realidad "interfaces").
```

```
typedef ScopedNameType ObjectClassType;
/** Object Class Set es un conjunto de clases de objeto. */
typedef sequence <ObjectClassType> ObjectClassSetType;
/** Los módulos de vinculación de nombres se identifican con nombres previstos. */
typedef ScopedNameType NameBindingType;
/** El campo StartTimeType se utiliza para especificar el tiempo en que comienza
una operación. Está a menudo apareado con StopTimeType para controlar la
activación de algunas funciones.
*/
typedef GeneralizedTimeType StartTimeType;
/** Los conjuntos de cadenas (string sets) son conjuntos de cadenas. */
typedef sequence <string> StringSetType;
/** Los rótulos del sistema son cadenas utilizadas para identificar sistemas.  */
typedef string SystemLabelType;
/** El estado desconocido (unknown status) se utiliza para indicar si es
desconocido el estado de un recurso. Un valor "cierto" indica que el estado es
desconocido.  */
typedef boolean UnknownStatusType;
```

// TIPOS ENUMERADOS

```
/* Los siguientes objetos de estado se utilizan en muchas interfaces y son
paralelos a los objetos de estado de las normas CMIP. */
/** El estado administrativo (administrative state) es de lectura/escritura. Un
objeto "cerrado" es normalmente uno que no puede ser cambiado o uno que no está
suministrando servicio. Asignando el estado administrativo de un objeto a
"apagado" comienza el proceso de apagado para ese objeto. */
enum AdministrativeStateType {locked, unlocked, shuttingDown};
/** El estado operativo (operational state) es de sólo lectura. Simplemente
informa de la capacidad real del objeto para prestar servicio. */
enum OperationalStateType {disabled, enabled};
/** El estado utilización (usage) es de sólo lectura. Si "inactivo", el recurso
está completamente sin utilizar. Si "ocupado", se está utilizando la capacidad
total del recurso. "Activo" es una situación intermedia.  */
enum UsageStateType {idle, active, busy};
/** La política de supresión (delete policy) indica si se puede suprimir un
objeto, y en caso afirmativo si se puede suprimir alguno de los objetos
contenidos. Debido a que los objetos no deben estar huérfanos, si un objeto tiene
una política de supresión "deleteOnlyIfNoContainedObjects" (suprimir sólo si no
contiene objetos) el objeto no debe ser suprimido si tiene objetos contenidos. Un
valor "deleteContainedObjects" (suprimir objetos contenidos) significa que si se
suprime el objeto se suprimirán también los objetos contenidos.  */
enum DeletePolicyType {notDeletable, deleteOnlyIfNoContainedObjects,
                deleteContainedObjects};
/** PerceivedSeverity informa de la severidad de una alarma. Se utiliza
"Indeterminate" (indeterminada) cuando no es posible asignar uno de los demás
valores. */
enum PerceivedSeverityType {indeterminate, critical, major, minor,
                                warning, cleared};
/** Se utiliza indicador de origen (source indicator) en muchas notificaciones.
Identifica si la notificación es el resultado de una operación de gestión o de
algo que ha ocurrido en el sistema gestionado.  */
enum SourceIndicatorType {resourceOperation, managementOperation,
                                unknown};
/** El atributo a estado en espera (standby) es un valor único de lectura
solamente. Este valor es sólo significativo cuando existe un cometido de relación
de respaldo. En el estado "reserva activa", el recurso no proporciona servicio
sino que funciona en sincronismo con otro recurso que se debe respaldar.
En el estado "reserva pasiva", el recurso respalda a otro recurso, pero no está
sincronizado con ese recurso. En el estado "prestando servicio" el recurso de
respaldo está prestando servicio y respalda a otro recurso.
*/
enum StandbyStatusType {hotStandby, coldStandby, providingService};
/** Los tiempos de parada (stop times) se utilizan para especificar cuándo debe
cesar una función. Normalmente hay dos elecciones, la función circula
continuamente (en cuyo caso no se especifica el tiempo real) o la función finaliza
en un tiempo especificado.
*/
enum StopTimeChoice {specific, continual};
/** La indicación umbral (threshold) describe si el cruce del umbral fue en la
dirección hacia arriba o hacia abajo. */
```

```
enum ThresholdIndicationType {up, down};
/** Los valores de TrendIndication indican si mejora, empeora, o no cambia la
tendencia de alguna condición observada.  */
enum TrendIndicationType {lessSevere, noChange, moreSevere};
```

// ESTRUCTURAS Y UNIONES

```
/* Las estructuras definidas a continuación se utilizan para pasar valores que se
pueden incluir opcionalmente. Para algunos tipos de valores, como las cadenas,
listas y punteros es fácil decir si está incluido el valor. Para otros, como las
enumeraciones, números y estructuras, no lo es. */
/** AdministrativeStateTypeOpt es un tipo opcional del estado administrativo. Si
el discriminador es cierto, el valor está presente, en otro caso el valor es nulo.
*/
union AdministrativeStateTypeOpt switch (boolean) {
        case TRUE:      AdministrativeStateType value;
};
/** BooleanTypeOpt es un tipo opcional booleano. Si el discriminador es cierto,
el valor está presente, en otro caso el valor es nulo. */
union BooleanTypeOpt switch (boolean) {
        case TRUE:      boolean value;
};
/** FloatTypeOpt es un tipo opcional flotante. Si el discriminador es cierto,
el valor está presente, en otro caso el valor es nulo. */
union FloatTypeOpt switch (boolean) {
        case TRUE:      float   value;
};
/** LongTypeOpt es un tipo opcional largo. Si el discriminador es cierto,
el valor está presente, en otro caso el valor es nulo. */
union LongTypeOpt switch (boolean) {
        case TRUE:      long    value;
};
/** OperationalStateTypeOpt es un tipo opcional de estado operativo.
Si el discriminador es cierto, el valor está presente, en otro caso el valor es
nulo. */
union OperationalStateTypeOpt switch (boolean) {
        case TRUE:      OperationalStateType    value;
};
/** ShortTypeOpt es un tipo opcional corto. Si el discriminador es cierto,
el valor está presente, en otro caso el valor es nulo. */
union ShortTypeOpt switch (boolean) {
        case TRUE:      short   value;
};
/** TrendIndicationTypeOpt es un tipo opcional de indicación de tendencia.
Si el discriminador es cierto, el valor está presente, en otro caso el valor es
nulo. */
union TrendIndicationTypeOpt switch (boolean) {
        case TRUE:      TrendIndicationType     value;
};
/** UnsignedShortTypeOpt es un tipo opcional. Si el discriminador es cierto, el
valor está presente, en otro caso el valor es nulo. */
union UnsignedShortTypeOpt switch (boolean) {
        case TRUE:      unsigned short value;
};
/** UsageStateTypeOpt es un tipo opcional de estado de utilización.
Si el discriminador es cierto, el valor está presente, en otro caso el valor es
nulo. */
union UsageStateTypeOpt switch (boolean) {
        case TRUE:      UsageStateType  value;
};
/** Muchas veces las especificaciones de la interfaz necesitan definir valores
estándar que sean pasados a través de la interfaz. También, a menudo, el esquema
utilizado para definir estos valores necesita ser extensible ya que se obtienen
subclases, de forma que las enumeraciones no funcionan correctamente. El CMIP
utiliza en sus normas los OID, cadenas de números que se añaden a veces. Con esta
finalidad se utiliza el ID único. Consta de dos partes, una cadena que contiene un
nombre previsto para el módulo, y un valor entero definido como una constante
dentro de ese módulo. Estos UID, y el tipo ObjectClass definido anteriormente,
reemplazan a los OID de la ASN.1. Se espera que cada módulo contendrá una cadena
constante llamada "moduleName" que contenga el nombre del módulo a utilizar por el
```

```
programador para estar libre de errores. Un nombre de módulo nulo indica un valor
nulo para el UID. <p>
El código para interpretar un UID puede ser análogo al del pequeño recorte de
código que sigue:<br>
<code><pre>
UIDType pc;     // causa probable
...
if (pc.moduleName ==
        itut_x780::ProbableCauseConst::moduleName) //string compare
        switch (pc.value) {
        case itut_x780::ProbableCauseConst::adapterError:
        ...
        case
        itut_x780::ProbableCauseConst::applicationSubsystemFailure:
        ...
        case itut_x780::ProbableCauseConst::bandwidthReduced:
        ...
        }
else if (pc.moduleName == MyLocal::ProbableCauseConst::moduleName)
        switch (pc.value) {
        ...
        }
</pre></code>
@member moduleName      El nombre de módulo previsto en el que están definidos los
                        valores.
@member value           El valor definido como una constante dentro del módulo.
*/
struct UIDType {
        string moduleName;      // módulo donde se define el valor
        short value;            // constante dentro del módulo
};
typedef sequence <UIDType> UIDSetType;
/** La extensión de gestión es una estructura para la comunicación flexible de
información. Se utiliza normalmente en el campo Información Adicional de las
notificaciones.
@see <a href="#AdditionalInformationSetType">
AdditionalInformationSetType </a>
@member id      identifica el tipo de información
@member any     contiene la información real; el tipo dependerá del valor del
miembro id.
*/
struct ManagementExtensionType {
        UIDType id;             // identifica el tipo de información
        any     info;           // el tipo dependerá del id
};
/** La información adicional es una forma flexible de comunicar información que no
se ajusta dentro de la estructura de una notificación. Contiene una secuencia de
una estructura llamada "Management Extension" (extensión de gestión). */
typedef sequence <ManagementExtensionType>
                        AdditionalInformationSetType;
/** En una notificación se utiliza una estructura valor de atributo (attribute
value) para informar del valor de cualquier atributo. La cadena utilizada para el
nombre del atributo es la misma que el nombre del miembro datos del objeto valor
definido para el objeto. En otras palabras, es el nombre del método de acceso a
atributos sin el "get" o "set".
@member attributeName   el nombre del atributo
@member value           contiene el valor del atributo; el tipo dependerá del
                        attributeName.
*/
struct AttributeValueType {
        string  attributeName;
        any     value;          // el tipo dependerá del atributo
};
/** Los conjuntos de valores de atributos (attribute value sets) se utilizan para
informar genéricamente de los atributos en un modo por lotes. */
typedef sequence <AttributeValueType> AttributeSetType;
/** Se utiliza en una notificación una estructura cambio de valor de atributo
(attribute value change) para informar que un atributo ha sido cambiado.
@see <a href="#AttributeValueType">AttributeValueType</a>
@member attributeName   el nombre del atributo
@member oldValue        el valor viejo; el tipo dependerá del attributeName
```

```
@member newValue        el nuevo valor; el tipo dependerá del attributeName.
*/
struct AttributeValueChangeType {
        string          attributeName;
        any             oldValue;       // el tipo depende del atributo
        any             newValue;       // el tipo depende del atributo
};
/** Se utiliza el conjunto de cambios de atributo (attribute change set) para
informar de los atributos que han sido cambiados en una notificación de cambio de
valor de atributo. */
typedef sequence <AttributeValueChangeType> AttributeChangeSetType;
/** Una notificación correlacionada (correlated notification) se identifica por el
objeto que emitió la notificación y por el ID de la notificación. Se incluyen
ambos en el caso de que los ID no sean únicos entre los objetos.
@member source   Referencia al objeto que emitió la notificación correlacionada.
                 Si es nulo, las notificaciones correlacionadas proceden del mismo
                 origen que las notificaciones que contienen esta estructura de
                 datos.
@member notifIDs Los ID de las notificaciones correlacionadas. Se deben elegir los
                  identificadores de notificación para que sean únicos en todas
                  las notificaciones de un determinado objeto gestionado durante
                  el tiempo en el que tiene sentido la correlación.
*/
struct CorrelatedNotificationType {
        NameType        source;
        NotifIDSetType  notifIDs;
};
/** Los conjuntos de notificaciones correlacionadas son conjuntos de estructuras
de notificaciones correlacionadas. */
typedef sequence <CorrelatedNotificationType>
                CorrelatedNotificationSetType;
/** En las normas CMIP, ProbableCause, puede ser un entero o una OID de GDMO, una
cadena con notación de puntos. Se utiliza en su lugar el tipo UID. */
typedef UIDType ProbableCauseType;
/** Las acciones de reparación propuestas (proposed repair actions) son conjuntos
de identificadores únicos. */
typedef UIDSetType ProposedRepairActionSetType;
/** Las causas de alarmas de seguridad (security alarm causes) son identificadores
únicos. */
typedef UIDType SecurityAlarmCauseType;
/** El Security Alarm Detector (detector de alarma de seguridad) puede indicar
bien un mecanismo o un objeto específico. De acuerdo con X.721 se elige entre uno
o el otro, aunque no está claro el porqué. (En realidad, X.721 añade una tercera
elección de un título AE que no tiene equivalencia aquí). A no ser que se indique
otra cosa, entonces, al menos uno de los miembros no será nulo. Se pueden enviar
dos nulos si el sistema gestionado no soporta esta propiedad.
@member mechanism       el esquema o la función que detecta la alarma; puede ser
                        nulo
@member obj             el objeto que detecta la alarma; puede ser nulo
*/
struct SecurityAlarmDetectorType {
        UIDType         mechanism;      // puede ser nulo
        NameType        obj;            // puede ser nulo
};
/** Usuario del servicio
@member id    el id del usuario del servicio
@member details detalles del usuario del servicio; el tipo depende del id
*/
struct ServiceUserType {
        UIDType id;
        any     details;        // el valor dependerá del id
};
/** Los suministradores del servicio comparten la misma representación que los
usuarios del servicio.
*/
typedef ServiceUserType ServiceProviderType;
/** Los problemas específicos (specific problems) son conjuntos de identificadores
únicos. */
typedef UIDSetType SpecificProblemSetType;
```

```
/** El tipo detener tiempo (Stop Time) se utiliza para indicar cuándo debe cesar
alguna función. En el caso específico, se da el tiempo real. En el caso continuo,
la función circula continuamente y en esta unión no se transporta ningún valor.
*/
union StopTimeType switch (StopTimeChoice) {
        case specific:  GeneralizedTimeType time;
        /* case continual carries NULL value */
};
/** Un SuspectObject identifica a un objeto sospechoso que puede ser la causa de
un fallo. Normalmente es un componente de una SuspectObjectList.
@member objectClass             Clase de objeto del objeto sospechoso
@member suspectObjectInstance   Ejemplar de objeto del objeto sospechoso
@member failureProbability      Responsabilidad de fallo opcional
                                Probabilidad de 1 a 100
*/
struct SuspectObjectType {
        ObjectClassType         objectClass;
        ManagedObject           suspectObjectInstance;
        UnsignedShortTypeOpt    failureProbability;
};
/** Las listas de objetos sospechosos (suspect object lists) se utilizan para
identificar objetos que pueden ser causa de un fallo. */
typedef sequence<SuspectObjectType> SuspectObjectSetType;
/** La indicación de nivel umbral (threshold level indication) describe los cruces
de umbrales multinivel. Hacia arriba es la única opción permitida para un
contador. En ASN.1, si la indicación es "up" (hacia arriba), el valor de comienzo
es opcional.
@member indication      indica dirección hacia arriba o hacia abajo del cruce.
@member low             el valor bajo observado.
@member high            el valor alto observado.
*/
struct ThresholdLevelIndType {
        ThresholdIndicationType         indication;
        FloatTypeOpt                    low;    // valor observado
        float                           high;   // valor observado
};
/** El tipo opcional de indicador de nivel de umbral (threshold level ind type
opt) es un tipo opcional. Si el discriminador es cierto, el valor está presente,
en otro caso el valor es nulo. */
union ThresholdLevelIndTypeOpt switch (boolean) {
        case TRUE:      ThresholdLevelIndType   value;
};
/** La información de umbral (threshold information) indica que algún atributo
de calibre o de contador pasó un umbral fijado. La estructura difiere de la
de X.721 para simplificar la sintaxis.
@member attributeID     Identifica al atributo que cruzó el umbral. En realidad es
                        un nombre de una operación sobre una interfaz, sin el
                        "get" o "set". La interfaz para la que se define la
                        operación se incluye en otro sitio de la notificación como
                        una ObjectClass. Un valor nulo indica que toda la
                        estructura es nula.
@member observedValue   Los atributos que son del tipo entero serán convertidos a
                        flotante.
@member thresholdlevel  Este parámetro es para umbrales multinivel. Opcional.
@member armTime         Puede ser nulo(0). */
struct ThresholdInfoType {
        string                          attributeID;
        float                           observedValue;
        ThresholdLevelIndTypeOpt        thresholdLevel;
        ExternalTimeType                armTime;
};
```

// EXCEPCIONES

```
/** Los tipos de información de error de aplicación se pasan hacia atrás en
excepciones de objetos gestionados.
@member error   Identificador único que identifica el problema.
@member details Mensaje de texto con información adicional acerca del problema.
*/
```

```
valuetype ApplicationErrorInfoType {
        public UIDType          error;
        public Istring          details;
};
/** Los tipos de información de error de creación se pasan hacia atrás en
excepciones de creación de objetos gestionados. Extienden los tipos de información
de error de aplicación.
@member relatedObjects  objetos que tienen alguna relación con el objeto a crear y
                        que de alguna manera impedían su creación.
@member attributeList   valores que se habrían asignado al objeto creado. Pueden
                        contener información del porqué no se pudo crear el
                        objeto.
*/
valuetype CreateErrorInfoType : ApplicationErrorInfoType {
        public MOSetType        relatedObjects;
        public AttributeSetType attributeList;
};
/** Los tipos de información error de supresión se pasan hacia atrás en
excepciones de supresión de objetos gestionados. Extienden los tipos de
información de error de aplicación.
@member relatedObjects  objetos que tienen alguna relación con el objeto a
                        suprimir que de alguna manera impedían la supresión.
@member attributeList   valores de los atributos asignados al objeto a suprimir
                        Pueden contener información del porqué no se pudo suprimir
                        el objeto.
*/
valuetype DeleteErrorInfoType : ApplicationErrorInfoType {
        public MOSetType        relatedObjects;
        public AttributeSetType attributeList;
};
/** Un tipo de información de error de lote es un error especial de creación. Se
pasará hacia atrás en una excepción de creación de objeto gestionado como un error
de creación. Si el código de error del UID concuerda con el tipo de información de
error de lote, la aplicación cliente puede estrechar el tipo de valor desde el
tipo de información de error de creación a tipo de información de error de lote
para acceder a la información adicional.
@member packages        lista de los lotes solicitados que originaban conflictos
                        o que no se podían soportar.
*/
valuetype PackageErrorInfoType : CreateErrorInfoType {
        public StringSetType    packages;
};
/** Las excepciones de error de aplicación pueden ser provocadas en cualquier
operación de objeto gestionado para identificar un problema que impide que se
complete la operación. */
exception ApplicationError { ApplicationErrorInfoType info; };
/** Las excepciones de error de creación pueden ser provocadas en cualquier
operación de creación de objeto gestionado para identificar un problema que impide
que se complete el objeto. */
exception CreateError { CreateErrorInfoType info; };
/** Las excepciones de error de supresión pueden ser provocadas por un objeto
gestionado en respuesta a un intento de suprimir el objeto. Pueden ser provocadas
también por el servicio terminador.
 */
exception DeleteError { DeleteErrorInfoType info; };
/** Las excepciones de longitud no válida son provocadas cuando se suministra una
longitud no válida en una invocación de operación. */
exception InvalidLength { long length; };
/** Las excepciones de cadena no válida son provocadas cuando se suministra una
cadena no válida en una invocación de operación. */
exception InvalidString {};
```

// TIPOS DE VALOR

```
/** Los tipos de valores cadena de bits se utilizan para representar cadenas de
bits con rótulos semánticos asociados que representan las posiciones de cadena de
bits. */
valuetype BitStringValue {
        /** El estado de una cadena de bits se mantiene en un miembro de datos de
        tipo BitString (secuencia de octetos). El primer octeto es la cuenta de
        bits no utilizados en el último octeto. */
```

```
        public BitString bitStringVal;
        /** Este inicializador fijará todas las posiciones de bits en '0' */
        factory initValue (in unsigned long number_of_bits);
        /** Este inicializador creará un nuevo BitStringValue con la misma
        longitud y valor que el BitString .*/
        factory InitFromBitString (in BitString desiredValue);
        // operaciones locales
        /** Esta operación local devuelve 0 ó 1 para los valores binarios en la
        posición especificada. Si la posición solicitada está más allá de la
        longitud de la cadena de bits provocará una excepción de longitud no
        válida. */
        short getBit (in unsigned long position)
                raises (InvalidLength);
        /** Esta operación local se utiliza para fijar el valor del bit en la
        posición solicitada. Si la posición solicitada está más allá de la
        longitud de la cadena de bits, provocará una excepción de longitud no
        válida. Si el new_bit_value es 0, el bit se pone a 0, de otro modo se pone
        a 1. */

        void setBit (in unsigned long position, in short new_bit_value)
                raises (InvalidLength);
        /** Esta operación local devuelve el número de bits en la cadena de bits.
        Puesto que el primer octeto contiene el número de bits no utilizados en el
        último octeto, y el último octeto puede contener bits no utilizados, el
        valor devuelto es
        (NumberOfOctets – 2)*8 + (8 – firstOctetVal) */
        unsigned long length ();
        /** Esta operación local devuelve el valor de una cadena de bits como una
        cadena de caracteres. Cada bit de valor 0 se debe representar como un
        carácter '0', y cada bit de valor '1' se representará como un carácter
        '1'. Se añadirá un carácter 'B' al final de la cadena ("1001011B").
        string asString ();
        /** Esta operación local fija el valor de la cadena de bits dando una
        cadena de caracteres. Cada carácter '0' en la cadena se convierte a un bit
        de valor 0, y cada carácter '1' se convierte a un 1. Si la cadena tiene
        algún carácter distinto de '0', '1', o 'B' de terminación, se provoca una
        excepción InvalidString. */
        void setFromString (in string string_value)
                raises (InvalidString);
};
```

// INTERFAZ DE OBJETO GESTIONADO

```
/** Este objeto del tipo valuetype contiene miembros para cada uno de los
atributos accessibles en esta interfaz. */
valuetype ManagedObjectValueType {
        public NameType                 name;
        public ObjectClassType          objectClass;
        public StringSetType            packages;
        public SourceIndicatorType      creationSource;
        public DeletePolicyType         deletePolicy;
};
/** La interfaz de objeto gestionado tiene por finalidad ser la interfaz base de
la que heredan todas las demás interfaces de objeto gestionado. Es un sitio
central para especificar las funcione básicas que se espera soporten todos los
objetos gestionados. */
interface ManagedObject {
        /** Este método devuelve el nombre completamente cualificado del objeto.
        Se utiliza este método en lugar de tener un método "get*ID" (obtener ID)
        para cada interfaz, como se hace en las especificaciones del CMIP. Esto
        asegura que los objetos sólo tienen una única operación para recuperar
        nombres cuando se les pasa a subclases.
        <p>
        La respuesta es una secuencia de estructuras de componentes de nombres,
        comenzando por el nombre asignado al contexto de denominación de la "raíz
        local" bajo la que está contenido este objeto. El cliente puede encontrar
        los superiores de este objeto retirando componentes desde el final de esta
        secuencia y llevando a cabo una operación de adecuación en la primera
        parte del nombre.
        NameType nameGet()
                raises (ApplicationError);
```

```
        /** Este método devuelve el nombre previsto de la clase más específica de
        la interfaz (por ejemplo, "EquipmentR1"). */
        ObjectClassType objectClassGet()
                raises (ApplicationError);
        /** Este método devuelve una lista de todos los lotes condicionales
        soportados por este ejemplar. */
        StringSetType packagesGet ()
                raises (ApplicationError);
        /** Este método devuelve una indicación de cómo fue creado el objeto. */
        SourceIndicatorType creationSourceGet()
                raises (ApplicationError);
        /** Este método devuelve un valor que indica si el objeto puede ser
        suprimido, y en caso de poderse, si todos los objetos contenidos se
        suprimen automáticamente. */
        DeletePolicyType deletePolicyGet ()
                raises (ApplicationError);
        /** Se puede utilizar este método para obtener genéricamente todos los
        atributos soportados por un ejemplar. Se supone que cada ejemplar hace
        subclases del tipo de valor del objeto gestionado y añade los demás
        atributos soportados por esa interfaz. El objeto gestionado debe devolver
        un objeto de valor de ese tipo. El cliente debe entonces estrechar la
        referencia para acceder a todos los atributos.
        <p>
        El cliente puede también presentar una lista de nombres que indique los
        atributos que desea recibir. Estos nombres deben concordar con los nombres
        de los miembros en el objeto valor. En el caso de los miembros que no
        están en la lista y de los miembros que son parte de lotes no soportados,
        el servidor puede devolver cualquier valor, que será lo más corto posible.
        El servidor devuelve también la lista de los atributos que puede ser más
        corta debido a la exclusión de atributos en los lotes no soportados. El
        cliente debe considerar al valor de cada miembro que no esté en la lista
        como desechable. <p>
        Una lista nula de nombres de atributos indica que hay que devolver todos
        los atributos soportados. El servidor debe devolver la lista real.  */
        ManagedObjectValueType attributesGet (
                inout StringSetType attributeNames)
                raises (ApplicationError);
        /** Este método destruye el objeto. Se utiliza simplemente para liberar
        cualquier recurso asociado con el objeto gestionado. No comprueba los
        objetos contenidos ni elimina vinculaciones de nombres del árbol de
        denominación. <p>
        La finalidad de esta operación es permitir servicios de soporte para la
        destrucción del objeto gestionado.  <p><b>
        NOTA: La invocación directa de esta operación por un sistema de gestión
        podría corromper el árbol de denominación y está recomendada sólo bajo
        circunstancias extraordinarias. Los clientes que deseen suprimir un
        objeto, deberán, en su lugar utilizar el servicio terminador. </b>  */
        (tipo opcional de estado de utilización)
        void destroy()
                raises (ApplicationError, DeleteError);
};  // fin de la interfaz ManagedObject.
```

// INTERFAZ DE FACTORÍA DE OBJETOS GESTIONADOS

```
/** Esta interfaz define la interfaz genérica de la factoría de objetos
gestionados. Todas las factorías de objetos gestionados deberán heredar desde esta
interfaz.  <p>
Además de proporcionar medios para la creación de objetos por medio de la
operación de gestión, se supone que las factorías asumen responsabilidades para el
mantenimiento de la integridad del árbol de denominación creando vinculaciones de
nombres para los objetos que han creado.  <p>
En la actualidad, esta interfaz es nula. Sin embargo, se incluye como un sitio
para colocar las capacidades que deben ser soportadas por todas las factorías de
objetos gestionados.
*/
interface ManagedObjectFactory {

};  // fin de la interfaz ManagedObjectFactory
```

// INTERFAZ DE NOTIFICACIONES

```
/** Esta interfaz contiene las definiciones de las notificaciones emitidas por
muchos objetos gestionados. <p>
El uso de notificaciones "tipificadas" se hace aquí de forma que se puedan
documentar en IDL y para soportar notificaciones las tipificadas para aquellos
gestores y sistemas de gestión que deseen utilizarlas. Téngase en cuenta que el
servicio de notificación del OMG soporta tanto las notificaciones estructuradas
como las tipificadas. No está claro si las implementaciones del servicio de
notificación soportarán la traslación entre ellas. Se espera que el acuerdo de
implementación entre el sistema gestor y el gestionado especificará la utilización
de las notificaciones estructuradas o tipificadas.  <p>
Los usuarios de notificaciones que deseen utilizar notificaciones tipificadas sólo
necesitan soportar las interfaces que siguen. Los editores de notificaciones y los
subscriptores que deseen utilizar notificaciones estructuradas basadas en las
operaciones definidas a continuación deberán seguir estas reglas para la
construcción y lectura de la estructura de la notificación: <ul><li>
La cadena domain_type en el encabezamiento fijo de la estructura se asignará a
"telecommunications".  <li>
La cadena event_type en el encabezamiento fijo de la estructura se asignará al
nombre previsto para la operación. Por ejemplo, para la notificación cambio de
valor de atributo definida más adelante este campo será
"itut_x780::Notifications::attributeValueChange".  <li>
La cadena event_name en el encabezamiento fijo de la estructura no es utilizada
por este marco. Se debe fijar en nula o utilizarse para otros fines.  <li>
Los campos opcionales de encabezamiento se pueden incluir para el soporte de
características como calidad del servicio según sea pertinente.  <li>
Se colocará cada parámetro de la operación en un par nombre-valor dentro de la
parte filtrable del cuerpo de la notificación. La cadena fd_name de este par será
asignada al nombre del parámetro, y el tipo colocado en el fd_value asociado será
el tipo especificado para el parámetro. Por ejemplo, cada una de las
notificaciones definidas más abajo tiene un parámetro llamado "eventTime" que es
un "ExternalTimeType". Este parámetro se colocará en la parte de datos filtrable
del evento. La cadena fd_name de este par se asignará a "eventTime" y el fd_value
contendrá un valor ExternalTimeType.  <li>
El resto del cuerpo de la notificación (la parte no filtrable) será nulo.  </ul>
Desafortunadamente, las notificaciones tipificadas se hacen corresponder a
estructuras de notificación de forma diferente, de manera que si uno de los
sistemas desea utilizar notificaciones tipificadas y el otro notificaciones
estructuradas, el usuario de las notificaciones estructuradas debe ser consciente
de la forma en la que el servicio de notificaciones de CORBA hace la traslación de
las notificaciones tipificadas a notificaciones estructuradas. Véase la
especificación para los detalles. Sin embargo, de forma breve, cada uno de los
parámetros de las operaciones que se presentan más abajo, se convertirá a un par
nombre-valor en la parte de datos filtrable de la notificación estructurada.
Asimismo, al campo event_type del encabezamiento fijo de la notificación
estructurada se la asignará al valor especial "%TYPED" y el campo domain_type será
una cadena vacía. Finalmente, se añadirá un par nombre-valor como primer elemento
en la parte de datos filtrables de la notificación con el nombre "operation". El
valor asociado con este nombre será una cadena con el valor asignado al nombre
previsto para la operación utilizada para emitir la notificación (por ejemplo,
itut_x780::Notifications::attributeValueChange).  <p>
Asimismo, los editores de notificaciones estructuradas pueden excluir parámetros
de las notificaciones que estén marcados como "opcional" o sean de un tipo
opcional (un nombre de tipo que termine en "TypeOpt"). Se deberá hacer esto por
motivos de eficacia. Sin embargo esto imposibilita la conversión automática de
notificaciones estructuradas a tipificadas, por lo que los gestores deberán ser
capaces de aceptar notificaciones estructuradas. (Estrictamente, no tienen por que
soportar notificaciones tipificadas, pero si los sistemas gestionados emiten
notificaciones tipificadas los gestores deberán aceptarlas, mejor que hacer
traslaciones, ya que será más eficaz). Si se incluye un parámetro "opcional" en
una notificación, se deberá utilizar el tipo "opcional" (unión discriminada).  <p>
En las operaciones de notificación se deberán evitar los parámetros con el nombre
"operación" con el fin de soportar el uso de notificaciones tipificadas. Mientras
que el canal de notificación deberá ser capaz de diferenciar el parámetro real del
añadido en base a sus posiciones dentro de la lista de datos filtrables, esto
podría repercutir en el filtrado, ya que el lenguaje por defecto de filtrado no
tiene posibilidad de diferenciar parámetros en base a su posición.  <p>
Debido a que se coloca el nombre previsto para la operación bien en la cadena
type_name (cuando se utilizan las notificaciones estructuradas) o en el cuerpo de
un par nombre-valor filtrable con el nombre "operación" (cuando se utilizan
```

```
        notificaciones tipificadas), no hay parámetro "event type" incluido explícitamente
        en ninguna de las notificaciones de las estructuras de datos. */
interface Notifications {
        /** Se utiliza una notificación cambio de valor de atributo para informar acerca
        de cambios en los atributos de un objeto, tales como la añadidura o la supresión
        de miembros de uno o más atributos con valor asignado y el reemplazo del valor de
        uno o más atributos.
        @param eventTime                Hora actual del sistema gestionado.
        @param source                   Objeto emisor de notificación.
        @param sourceClass              Clase real del objeto origen.
        @param notificationIdentifier   Identificador único para esta notificación. Debe
                                        ser único para un ejemplar de objeto.(Opcional en
                                        X.721, pero no aquí. Para el estudio de las
                                        repercusiones posibles véase el texto).
        @param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                        Una secuencia de longitud cero indica la ausencia
                                        de este parámetro.
        @param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                        cero indica la ausencia de este parámetro.
        @param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                        ausencia de este parámetro.
        param sourceIndicator           Causa del evento. Opcional. Utilizar "unknown"
                                        (desconocido) si no se soporta.
        @param attributeChanges         Atributos cambiados.
        */
        void attributeValueChange (
                in ExternalTimeType                     eventTime,
                in NameType                             source,
                in ObjectClassType                      sourceClass,
                in NotifIDType                          notificationIdentifier,
                in CorrelatedNotificationSetType        correlatedNotifications,
                in AdditionalTextType                   additionalText,
                in AdditionalInformationSetType         additionalInfo,
                in SourceIndicatorType                  sourceIndicator,
                in AttributeChangeSetType               attributeChanges
        );
        /** Se utiliza una notificación alarma de comunicaciones para informar de cuándo
        un objeto detecta un error de comunicaciones.
        @param eventTime                Hora actual del sistema gestionado.
        @param source                   Objeto emisor de notificación.
        @param sourceClass              Clase real del objeto origen.
        @param notificationIdentifier   Identificador único para esta notificación. Debe
                                        ser único para un ejemplar de objeto. (Opcional en
                                        X.721, pero no aquí. Para el estudio de las
                                        repercusiones posibles véase el texto).
        @param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                        Una secuencia de longitud cero indica la ausencia
                                        de este parámetro.
        @param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                        cero indica la ausencia de este parámetro.
        @param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                        ausencia de este parámetro.
        @param probableCause
        @param specificProblems         Opcional. Una secuencia de longitud cero indica la
                                        ausencia de este parámetro.
        @param perceivedSeverity
        @param backedUpStatus           "Verdadero" ("True") si tiene copia de respaldo.
        @param backUpObject             Será nulo si backedUpStatus es "falso" ("false").
        @param trendIndication          Opcional. Para detalles, véase tipo.
        @param thresholdInfo            Opcional. Para detalles, véase tipo.
        @param stateChangeDefinition    Opcional. Una secuencia de longitud cero indica la
                                        ausencia de este parámetro.
        @param monitoredAttributes      Opcional. Una secuencia de longitud cero indica la
                                        ausencia de este parámetro.
        @param proposedRepairActions    Opcional. Una secuencia de longitud cero indica la
                                        ausencia de este parámetro.
        @param alarmEffectOnService     Verdadero si la alarma presta servicio.
        @param alarmingResumed          Verdadero si se acaba de reanudar el sistema de
                                        alarma lo que posiblemente causará un retardo en
                                        la información de una alarma.
```

```
@param suspectObjectList        Objetos posiblemente involucrados en el fallo.
*/
void communicationsAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType                perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** Se utiliza una notificación alarma de entorno para informar de un problema en
el entorno.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto.(Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param probableCause
@param specificProblems         Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param perceivedSeverity
@param backedUpStatus           "Verdadero" ("True") si tiene copia de respaldo.
@param backUpObject             Será nulo si backedUpStatus es "falso" ("false").
@param trendIndication          Opcional. Para detalles, véase tipo.
@param thresholdInfo            Opcional. Para detalles, véase tipo.
@param stateChangeDefinition    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param monitoredAttributes      Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param proposedRepairActions    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param alarmEffectOnService     Verdadero si la alarma presta servicio.
@param alarmingResumed          Verdadero si se acaba de reanudar el sistema de
                                alarma lo que posiblemente causará un retardo en
                                la información de una alarma.
@param suspectObjectList        Objetos posiblemente involucrados en el fallo.
*/
void environmentalAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType                perceivedSeverity,
```

```
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** Se utiliza una notificación alarma de equipo para informar de un fallo en el
equipo.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param probableCause
@param specificProblems         Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param perceivedSeverity
@param backedUpStatus           "Verdadero" ("True") si tiene copia de respaldo.
@param backUpObject             Será nulo si backedUpStatus es "falso" ("false").
@param trendIndication          Opcional. Para detalles, véase tipo.
@param thresholdInfo            Opcional. Para detalles, véase tipo.
@param stateChangeDefinition    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param monitoredAttributes      Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param proposedRepairActions    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param alarmEffectOnService     Verdadero si la alarma presta servicio.
@param alarmingResumed          Verdadero si se acaba de reanudar el sistema de
                                alarma lo que posiblemente causará un retardo en
                                la información de una alarma.
@param suspectObjectList        Objetos posiblemente involucrados en el fallo.
*/
void equipmentAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType                perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
```

```
/** Se utiliza una notificación violación de integridad para informar de que se ha
producido una posible interrupción en el flujo de información, de forma que la
información ha podido ser modificada, insertada o suprimida ilegalmente.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param securityAlarmCause
@param securityAlarmSeverity    ¿Se permiten los borrados? Parece que X.721
                                restringe el valor "cleared" (borrado) en esta
                                alarma, pero debería estar permitido.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void integrityViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType                securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** Se utiliza una notificación creación de objeto para informar a otro sistema
abierto acerca de la creación de un objeto gestionado. Téngase en cuenta que el
campo origen se deberá asignar al objeto creado, no a la factoría.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param sourceIndicator          Causa del evento. Opcional. Utilizar "unknown"
                                (desconocida) si no se soporta.
@param attributeSet             Valores de atributo. Opcional. Una secuencia de
                                longitud cero indica la ausencia de este
                                parámetro.
*/
void objectCreation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
```

```
        in SourceIndicatorType                  sourceIndicator,
        in AttributeSetType                     attributeList
);
/** Se utiliza una notificación supresión de objeto para informar de la supresión
de un objeto gestionado. Téngase en cuenta que el campo origen se deberá asignar
al objeto que se suprime.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param sourceIndicator          Causa del evento. Opcional. Utilizar "unknown"
                                (desconocida) si no se soporta.
@param attributeSet             Valores de atributo. Opcional. Una secuencia de
                                longitud cero indica la ausencia de este
                                parámetro.
*/
void objectDeletion (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SourceIndicatorType                  sourceIndicator,
        in AttributeSetType                     attributeList
);
/** Se utiliza una notificación violación operativa para informar de que no fue
posible la prestación del servicio solicitado debido a la no disponibilidad, mal
funcionamiento o a la invocación incorrecta del servicio.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param securityAlarmCause
@param securityAlarmSeverity    ¿Se permiten los borrados? Parece que X.721
                                restringe el valor "cleared" (borrado) en esta
                                alarma, pero debería estar permitido.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void operationalViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
```

```
        in PerceivedSeverityType                securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** Se utiliza una notificación violación física para informar de que ha sido
violado un recurso físico, de forma tal que indica un ataque potencial a la
seguridad.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param securityAlarmCause
@param securityAlarmSeverity    ¿Se permiten los borrados? Parece que X.721
                                restringe el valor "cleared" (borrado) en esta
                                alarma, pero debería estar permitido.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void physicalViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType                securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** Se utiliza una notificación alarma de error de proceso para informar de un
fallo de proceso en un objeto gestionado.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param probableCause
@param specificProblems         Opcional. Una secuencia de longitud cero
                                indica la ausencia de este parámetro.
@param perceivedSeverity
@param backedUpStatus           "Verdadero" ("True") si tiene copia de respaldo.
@param backUpObject             Será nulo si backedUpStatus es "falso" ("false").
@param trendIndication          Opcional. Para detalles, véase tipo.
@param thresholdInfo            Opcional. Para detalles, véase tipo.
@param stateChangeDefinition    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
```

```
@param monitoredAttributes      Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param proposedRepairActions    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param alarmEffectOnService     Verdadero si la alarma presta servicio.
@param alarmingResumed          Verdadero si se acaba de reanudar el sistema de
                                alarma lo que posiblemente causará un retardo en
                                la información de una alarma.
@param suspectObjectList        Objetos posiblemente involucrados en el fallo.
*/
void processingErrorAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType                perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** Se utiliza una notificación alarma de calidad del servicio para informar de un
fallo en la calidad de servicio del objeto gestionado.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param probableCause
@param specificProblems         Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param perceivedSeverity
@param backedUpStatus           "Verdadero" ("True") si tiene copia de respaldo.
@param backUpObject             Será nulo si backedUpStatus es falso ("false").
@param trendIndication          Opcional. Para detalles, véase tipo.
@param thresholdInfo            Opcional. Para detalles, véase tipo.
@param stateChangeDefinition    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param monitoredAttributes      Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param proposedRepairActions    Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param alarmEffectOnService     Verdadero si la alarma presta servicio.
@param alarmingResumed          Verdadero si se acaba de reanudar el sistema de
                                alarma lo que posiblemente causará un retardo en
                                la información de una alarma.
@param suspectObjectList        Objetos posiblemente involucrados en el fallo.
*/
void qualityOfServiceAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
```

```
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType                perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** Se utiliza una notificación cambio de relación para informar del cambio de uno
o más de los atributos de relación de un objeto gestionado, que se produce bien
por operaciones internas del objeto gestionado o por operaciones de gestión.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param sourceIndicator          Causa del evento. Opcional. Utilizar "unknown"
                                (desconocida) si no se soporta.
@param relationshipChanges      Atributos de relación cambiados.
*/
void relationshipChange (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SourceIndicatorType                  sourceIndicator,
        in AttributeChangeSetType               relationshipChanges
);
/** Se utiliza una notificación violación de seguridad para informar de que se ha
detectado un ataque a la seguridad que ha sido detectado por un servicio o
mecanismo de seguridad.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param securityAlarmCause
```

```
@param securityAlarmSeverity    ¿Se permiten los borrados? Parece que X.721
                                restringe el valor "cleared" (borrado) en esta
                                alarma, pero debería estar permitido.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void securityViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType                securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** Se utiliza una notificación cambio de estado para informar del cambio de valor
de uno o más atributos de estado de un objeto gestionado, que se produce bien por
una operación interna del objeto gestionado o por una operación de gestión.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
@param sourceIndicator          Causa del evento. Opcional. Utilizar "unknown"
                                (desconocida) si no se soporta.
@param stateChanges             Atributos de estado cambiados.
*/
void stateChange (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SourceIndicatorType                  sourceIndicator,
        in AttributeChangeSetType               stateChanges
);
/** Se utiliza una notificación violación en el dominio del tiempo para informar
de que se ha producido un evento en un momento inesperado o prohibido.
@param eventTime                Hora actual del sistema gestionado.
@param source                   Objeto emisor de notificación.
@param sourceClass              Clase real del objeto origen.
@param notificationIdentifier   Identificador único para esta notificación. Debe
                                ser único para un ejemplar de objeto. (Opcional en
                                X.721, pero no aquí. Para el estudio de las
                                repercusiones posibles véase el texto).
@param correlatedNotifications  Lista de notificaciones correlacionadas. Opcional.
                                Una secuencia de longitud cero indica la ausencia
                                de este parámetro.
@param additionalText           Mensaje de texto. Opcional. Una cadena de longitud
                                cero indica la ausencia de este parámetro.
@param additionalInfo           Opcional. Una secuencia de longitud cero indica la
                                ausencia de este parámetro.
```

```
@param securityAlarmCause
@param securityAlarmSeverity    ¿Se permiten los borrados? Parece que X.721
                                restringe el valor "cleared" (borrado) en esta
                                alarma, pero debería estar permitido.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void timeDomainViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetType        correlatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType                securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** Estas constantes definen los nombres de las notificaciones declaradas
anteriormente y se proporcionan para ayudar a reducir los errores.  */
const string attributeValueChangeTypeName =
        "itut_x780::Notifications::attributeValueChange";
const string communicationsAlarmTypeName =
        "itut_x780::Notifications::communicationsAlarm";
const string environmentalAlarmTypeName =
        "itut_x780::Notifications::environmentalAlarm";
const string equipmentAlarmTypeName =
        "itut_x780::Notifications::equipmentAlarm";
const string integrityViolationTypeName =
        "itut_x780::Notifications::integrityViolation";
const string objectCreationTypeName =
        "itut_x780::Notifications::objectCreation";
const string objectDeletionTypeName =
        "itut_x780::Notifications::objectDeletion";
const string operationalViolationTypeName =
        "itut_x780::Notifications::operationalViolation";
const string physicalViolationTypeName =
        "itut_x780::Notifications::physicalViolation";
const string processingErrorAlarmTypeName =
        "itut_x780::Notifications::processingErrorAlarm";
const string qualityOfServiceAlarmTypeName =
        "itut_x780::Notifications::qualityOfServiceAlarm";
const string relationshipChangeTypeName =
        "itut_x780::Notifications::relationshipChange";
const string securityViolationTypeName =
        "itut_x780::Notifications::securityViolation";
const string stateChangeTypeName =
        "itut_x780::Notifications::stateChange";
const string timeDomainViolationTypeName =
        "itut_x780::Notifications::timeDomainViolation";
/** Estas constantes definen los nombres de los parámetros utilizados en las
notificaciones declaradas anteriormente que se suministran para ayudar a
reducir los errores.
*/
const string additionalInfoName = "additionalInfo";
const string additionalTextName = "additionalText";
const string alarmEffectOnServiceName = "alarmEffectOnService";
const string alarmingResumedName = "alarmingResumed";
const string attributeChangesName = "attributeChanges";
const string attributeListName = "attributeList";
const string backedUpStatusName = "backedUpStatus";
const string backUpObjectName = "backUpObject";
const string correlatedNotificationsName = "correlatedNotifications";
const string eventTimeName = "eventTime";
const string monitoredAttributesName = "monitoredAttributes";
const string notificationIdentifierName = "notificationIdentifier";
const string perceivedSeverityName = "perceivedSeverity";
```

```
        const string probableCauseName = "probableCause";
        const string proposedRepairActionsName = "proposedRepairActions";
        const string relationshipChangesName = "relationshipChanges";
        const string securityAlarmCauseName = "securityAlarmCause";
        const string securityAlarmDetectorName = "securityAlarmDetector";
        const string securityAlarmSeverityName = "securityAlarmSeverity";
        const string serviceProviderName = "serviceProvider";
        const string serviceUserName = "serviceUser";
        const string sourceName = "source";
        const string sourceClassName = "sourceClass";
        const string sourceIndicatorName = "sourceIndicator";
        const string specificProblemsName = "specificProblems";
        const string stateChangeDefinitionName = "stateChangeDefinition";
        const string stateChangesName = "stateChanges";
        const string suspectObjectListName = "suspectObjectList";
        const string thresholdInfoName = "thresholdInfo";
        const string trendIndicationName = "trendIndication";
  }; // fin de la interfaz Notifications

}; // fin del módulo itut_x780
```

// MACROS

```
/* Se presentan las siguientes macros como una definición rápida y concisa de las
notificaciones que deben ser soportadas por un objeto. Ejemplo de uso (dentro de una
interfaz):
MANDATORY_NOTIFICATION(itut_x780::Notifications, objectCreation);
CONDITIONAL_NOTIFICATION(itut_x780::Notifications, stateChange, statePackage);
Las macros simplemente se expanden a nada, ya que el IDL de CORBA no tiene en realidad
nada en lo que expandirlas que tenga sentido. Eventualmente se pueden cambiar las macros
en el IDL que soporte el modelo componente de CORBA.
*/
#undef MANDATORY_NOTIFICATION
#define MANDATORY_NOTIFICATION(InterfaceName, NotificationName)
#undef CONDITIONAL_NOTIFICATION
#define CONDITIONAL_NOTIFICATION(InterfaceName, NotificationName, PackageName)
#endif // fin de ifndef itut_x780_IDL
```

ANEXO B

Definiciones de constantes de gestión de red

```
/* Este código IDL tiene como finalidad la de ser almacenado en un fichero con nombre
"itut_x780Const.idl"
y situado en el mismo directorio que el que alberga al fichero del anexo A  */
#ifndef ITUT_X780Const_IDL
#define ITUT_X780Const_IDL
#pragma prefix "itu.int"
module itut_x780 {
```

// Módulo ApplicationErrorConst

```
/** Este módulo contiene las constantes definidas para el código de error contenido en las
estructuras información de error de aplicación devueltas con excepciones de error de
aplicación.
*/
module ApplicationErrorConst {
        const string moduleName = "itut_x780::ApplicationErrorConst";
        /** Este código de excepción de error de aplicación indica que la operación falló
        debido a un problema hacia abajo desde el sistema gestionado, posiblemente un
        problema de comunicación entre el sistema gestionado y el recurso */
        const short downstreamError = 1;
        /** Una excepción de error de aplicación que devuelva este código devolverá el
        nombre del parámetro causante en el campo de detalles. */
        const short invalidParameter = 2;
```

```
        /** Este código de excepción de error de aplicación indica que la operación falló
        debido a un problema transitorio en el sistema gestionado. */
        const short resourceLimit = 3;
}; // fin del módulo ApplicationErrorConst
```

// Módulo CreateErrorConst

```
/** Este módulo contiene las constantes definidas para el código de error contenido en las
estructuras información de error de creación devueltas con las excepciones error de
creación.
*/
module CreateErrorConst {
        const string moduleName = "itut_x780::CreateErrorConst";
        /** Este código de excepción de error de creación indica que no es válido el
        nombre incluido en la operación de creación. */
        const short badName = 1;
        /** Este código de excepción de error de creación indica que el nombre incluido en
        la operación de creación está duplicado. */
        const short duplicateName = 2;
        /** Este código de excepción de error de creación indica que algunos de los lotes
        solicitados en la operación de creación son incompatibles entre sí. Se debe
        incluir en una estructura PackageErrorInfoType (subclase de CreateErrorInfoType).
        La lista de los lotes contiene los nombres de los lotes no soportados. */
        const short incompatiblePackages = 3;
        /** Este código de excepción de error de creación indica que no es válida la
        vinculación de nombres referenciada en la operación de creación. */
        const short invalidNameBinding = 4;
        /** Este código de excepción de error de creación indica que no se soporta un lote
        solicitado en una operación de creación. Debe ser incluido en una estructura
        PackageErrorInfoType (subclase de CreateErrorInfoType). La lista de los lotes
        contiene los nombres de los lotes no soportados. */
        const short unsupportedPackages = 5;
}; // fin del módulo CreateErrorConst
```

// Módulo DeleteErrorConst

```
/** Este módulo contiene las constantes definidas en el código de error contenido en las
estructuras información de error de supresión devueltas con las excepciones de error de
supresión.
*/
module DeleteErrorConst {
        const string moduleName = "itut_x780::DeleteErrorConst";
        /** Este código de excepción de error de supresión indica que el objeto tiene al
        mismo tiempo subordinados y una política de supresión deleteOnlyIfNoContained. */
        const short containsObjects = 1;
        /** Este código de excepción de error de supresión indica que el objeto tiene una
        política de supresión notDeletable, y no puede ser suprimido. */
        const short notDeletable = 2;
        /** Este código de excepción de error de supresión indica que el objeto tenía un
        objeto subordinado que no se pudo suprimir, por lo que el objeto u objetos
        superiores no se podían suprimir. */
        const short undeletableContainedObject = 3;
        /** Este código de excepción de error de supresión indica que el objeto está en un
        estado en que no puede ser suprimido. */
        const short invalidStateForDestroy = 4;
}; // fin del módulo DeleteErrorConst
```

// Módulo ProbableCauseConst

```
/** Este módulo contiene los valores de las constantes definidas para el UID
ProbableCause. Estos valores están tomados de X.721. */
module ProbableCauseConst {
const string moduleName = "itut_x780::ProbableCauseConst";
        const short indeterminate = 0;
        const short adapterError = 1;
        const short applicationSubsystemFailure = 2;
        const short bandwidthReduced = 3;
        const short callEstablishmentError = 4;
        const short communicationsProtocolError = 5;
        const short communicationsSubsystemFailure = 6;
        const short configurationOrCustomizationError = 7;
        const short congestion = 8;
```

```
        const short corruptData = 9;
        const short cpuCyclesLimitExceeded = 10;
        const short dataSetOrModemError = 11;
        const short degradedSignal = 12;
        const short dTE_DCEInterfaceError = 13;
        const short enclosureDoorOpen = 14;
        const short equipmentMalfunction = 15;
        const short excessiveVibration = 16;
        const short fileError = 17;
        const short fireDetected = 18;
        const short floodDetected = 19;
        const short framingError = 20;
        const short heatingOrVentilationOrCoolingSystemProblem = 21;
        const short humidityUnacceptable = 22;
        const short inputOutputDeviceError = 23;
        const short inputDeviceError = 24;
        const short lANError = 25;
        const short leakDetected = 26;
        const short localNodeTransmissionError = 27;
        const short lossOfFrame = 28;
        const short lossOfSignal = 29;
        const short materialSupplyExhausted = 30;
        const short multiplexerProblem = 31;
        const short outOfMemory = 32;
        const short ouputDeviceError = 33;
        const short performanceDegraded = 34;
        const short powerProblem = 35;
        const short pressureUnacceptable = 36;
        const short processorProblem = 37;
        const short pumpFailure = 38;
        const short queueSizeExceeded = 39;
        const short receiveFailure = 40;
        const short receiverFailure = 41;
        const short remoteNodeTransmissionError = 42;
        const short resourceAtOrNearingCapacity = 43;
        const short responseTimeExcessive = 44;
        const short retransmissionRateExcessive = 45;
        const short softwareError = 46;
        const short softwareProgramAbnormallyTerminated = 47;
        const short softwareProgramError = 48;
        const short storageCapacityProblem = 49;
        const short temperatureUnacceptable = 50;
        const short thresholdCrossed = 51;
        const short timingProblem = 52;
        const short toxicLeakDetected = 53;
        const short transmitFailure = 54;
        const short transmitterFailure = 55;
        const short underlyingResourceUnavailable = 56;
        const short versionMismatch = 57;
}; // fin del módulo ProbableCauseConst.

/** Este módulo contiene los valores de las constantes definidos para el UID de la
SecurityAlarmCause. Estos valores están tomados de X.721 */
module SecurityAlarmCauseConst {
const string moduleName = "itut_x780::SecurityAlarmCauseConst";
        const short authenticationFailure = 1;
        const short breachOfConfidentiality = 2;
        const short cableTamper = 3;
        const short delayedInformation = 4;
        const short denialOfService = 5;
        const short duplicateInformation = 6;
        const short informationMissing = 7;
        const short informationModificationDetected = 8;
        const short informationOutOfSequence = 9;
        const short intrusionDetection = 10;
        const short keyExpired = 11;
        const short nonRepudiationFailure = 12;
        const short outOfHoursActivity = 13;
        const short outOfService = 14;
        const short proceduralError = 15;
        const short unauthorizedAccessAttempt = 16;
```

```
        const short unexpectedInformation = 17;
        const short unspecifiedReason = 18;
}; // fin del módulo SecurityAlarmCauseConst
}; // fin del módulo itut_x780
#endif // fin de ifndef ITUT_X780Const_IDL
```

APÉNDICE I

Bibliografía

Las siguientes Recomendaciones y otras referencias contienen información que fue utilizada en el desarrollo de las presentes directrices. Como se ha mencionado en la introducción, uno de los principales objetivos de estas directrices es el permitir la reutilización de modelos de información de gestión de red ya existentes, al menos sin cambios semánticos significativos. Estos documentos proporcionan muchos de los detalles acerca del marco de trabajo CMIP del UIT-T y, por tanto, definen algunas de las funcionalidades que deben soportar las directrices de modelado de objetos CORBA.

[8] UIT-T X.720 (1992) | ISO/CEI 10165-1:1993, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Modelo de información de gestión*.

[9] UIT-T X.733 (1992) | ISO/CEI 10164-4:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función señaladora de alarmas*.

[10] UIT-T M.3010 (2000), *Principios para una red de gestión de las telecomunicaciones*.

[11] UIT-T M.3120 (2001), *Modelo de información de red genérico basado en la arquitectura de intermediario de petición de objeto común y a nivel de elemento de red*.

[12] UIT-T Q.821 (2000), Descripción de las etapas 2 y 3 para la interfaz Q3 – Vigilancia de alarmas.

SERIES DE RECOMENDACIONES DEL UIT-T

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

Ginebra, 2001