UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T X.950

(08/97)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE X: REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

Procesamiento distribuido abierto

Tecnología de la información – Procesamiento distribuido abierto – Función intermediación: Especificación

Recomendación UIT-T X.950

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES DE LA SERIE X DEL UIT-T

REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

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

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

NORMA INTERNACIONAL 13235-1

RECOMENDACIÓN UIT-T X.950

TECNOLOGÍA DE LA INFORMACIÓN – PROCESAMIENTO DISTRIBUIDO ABIERTO – FUNCIÓN INTERMEDIACIÓN: ESPECIFICACIÓN

Resumen

Esta Recomendación | Norma Internacional especifica la función intermediación ODP. La función intermediación es uno de los componentes de la arquitectura del modelo de referencia ODP definido en la Rec. UIT-T X.903 | ISO/CEI 10746-3. La función intermediación proporciona el medio de ofrecer servicios y descubrir servicios que han sido ofrecidos en un sistema ODP. El campo de aplicación de la función intermediación es cualquier sistema ODP en el que es necesario introducir servicios de manera incremental, dinámica y abierta.

Orígenes

El texto de la Recomendación UIT-T X.950 se aprobó el 9 de agosto de 1997. Su texto se publica también, en forma idéntica, como Norma Internacional ISO/CEI 13235-1.

PREFACIO

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

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

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

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

NOTA

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

PROPIEDAD INTELECTUAL

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

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

ÍNDICE

Página

Introducción

El rápido crecimiento del procesamiento distribuido ha creado la necesidad de un marco de coordinación para la normalización del procesamiento distribuido abierto (ODP, *open distributed processing*). El modelo de referencia del procesamiento distribuido abierto (RM-ODP) proporciona este marco. Define una arquitectura dentro de la cual pueden integrarse soporte de distribución, interoperabilidad y portabilidad.

Uno de los componentes de la arquitectura (descrito en el RM-ODP) parte 3: arquitectura) (Rec. UIT-T X.903 | ISO/CEI 10746-3) es la función intermediación ODP. La función intermediación proporciona el medio de ofrecer un servicio y el medio de descubrir servicios que han sido ofrecidos. Esta Recomendación | Norma Internacional presenta una arquitectura para los sistemas que implementan la función de intermediación y la especificación de interfaces dentro de la arquitectura.

NOTA – La especificación de las interfaces computacionales en esta Recomendación | Norma Internacional está técnicamente alineada con el servicio de objetos de intermediación OMG.

Los objetivos de esta Recomendación | Norma Internacional son:

- proporcionar una norma que sea independiente de cualquier implementación;
- asegurar implementaciones que puedan hacerse interoperar (es decir, pueden ser federadas);
- proporcionar suficientes detalles para permitir evaluar las alegaciones de conformidad.

El anexo A es una especificación normativa ODP del IDL de las signaturas de interfaz de la función intermediación.

El anexo B es una especificación del lenguaje de constricciones de la función intermediación ODP.

El anexo C es una especificación normativa del lenguaje de recetas de constricciones de la función intermediación ODP.

El anexo D es una descripción informativa de un depositario de tipos de servicio.

NORMA INTERNACIONAL

RECOMENDACIÓN UIT-T

TECNOLOGÍA DE LA INFORMACIÓN – PROCESAMIENTO DISTRIBUIDO ABIERTO – FUNCIÓN INTERMEDIACIÓN: ESPECIFICACIÓN

1 Alcance y campo de aplicación

El alcance de esta Recomendación | Norma Internacional es:

- una especificación de empresa para la función intermediación;
- una especificación de información para la función intermediación;
- una especificación computacional para los intermediarios (es decir, objetos que proporciona la función intermediación);
- requisitos de conformidad en términos de puntos de conformidad.

No es objetivo de esta Recomendación | Norma Internacional especificar cómo debe realizarse la función intermediación. Por tanto, esta Recomendación | Norma Internacional no incluye una especificación de ingeniería.

El campo de aplicación de esta Recomendación | Norma Internacional es cualquier sistema ODP en el que se necesite introducir y descubrir servicios de manera incremental, dinámica y abierta.

2 Referencias normativas

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

- Recomendación UIT-T X.901 (1997) | ISO/CEI 10746-1:1997, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Sinopsis*.
- Recomendación UIT-T X.902 (1995) | ISO/CEI 10746-2:1996, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Fundamentos*.
- Recomendación UIT-T X.903 (1995) | ISO/CEI 10746-3:1996, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Arquitectura*.
- Recomendación UIT-T X.920 (1997) | ISO/CEI 14750:1998, *Tecnología de la información – Procesamiento distribuido abierto – Lenguaje de definición de interfaces*.
- ISO/CEI 13568: *Information technology – The Z Specification Language*.

3 Notaciones

La especificación de información de la función intermediación se describe utilizando el lenguaje de descripción formal Z. La signatura de la interfaz computacional de la función de comercio se describe utilizando el lenguaje de definición de interfaces ODP en la cláusula 8, y en el anexo A.

4 Definiciones y abreviaturas

4.1 Definiciones de la Rec. UIT-T X.902 | ISO/CEI 10746-2

Esta Especificación se basa en el marco de abstracciones y conceptos desarrollados en el RM-ODP, y utiliza las siguientes definiciones del RM-ODP parte 2: Fundamentos (véase la Rec. UIT-T X.902 | ISO/CEI 10746-2).

a) acción;

b) actividad;

c) comportamiento;

d) compatibilidad de comportamiento;

e) vinculación;

f) objeto cliente;

g) punto de conformidad;

h) contrato;

i) dominio;

j) comportamiento de establecimiento;

k) fallo;

l) identificador;

m) objeto iniciador;

n) instancia;

o) interacción;

p) interfaz;

q) signatura de interfaz;

r) nombre;

s) objeto;

t) obligación;

u) sistema ODP;

v) permiso;

w) política;

x) prohibición;

y) calidad de servicio;

z) punto de referencia;

aa) objeto respondedor;

bb) rol;

cc) objeto servidor;

dd) subtipo;

ee) supertipo;

ff) plantilla;

gg) tipo de plantilla;

hh) intermediación;

ii) transparencia;

jj) tipo;

kk) punto de vista.

4.2 Definiciones de la Rec. UIT-T X.903 | ISO/CEI 10746-3

Esta Especificación se basa en el marco de abstracciones y conceptos desarrollados en el RM-ODP, y utiliza las siguientes definiciones del RM-ODP parte 3: arquitectura (véase la Rec. UIT-T X.903 | ISO/CEI 10746-3).

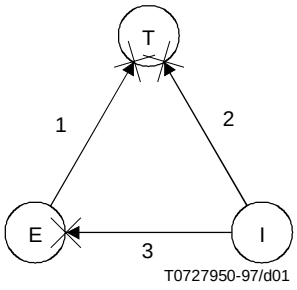
a) comunidad;
b) plantilla de interfaz computacional;
c) punto de vista computacional;
d) esquema dinámico;
e) punto de vista de la ingeniería;
f) punto de vista de la empresa;
g) exportador;
h) punto de vista de información;
i) esquema invariante;
j) esquema;
k) exportación de servicio;
l) importación de servicio;
m) oferta de servicio;
n) esquema estático;
o) punto de vista de la tecnología;
p) federación <X>.

5 Sinopsis de la función intermediación ODP

En el contexto del objetivo ODP de proporcionar utilización transparente de distribución de servicios sobre plataformas y redes heterogéneas, el papel de la función intermediación es permitir a los usuarios encontrar servicios potenciales. Es un corolario de la distribución que el descubrimiento de los servicios se produzca de manera dinámica.

La función intermediación ODP facilita el ofrecimiento y el descubrimiento de instancias de interfaces que proporcionan servicios de determinados tipos. Un intermediario es un objeto que sustenta la función intermediación en un entorno distribuido. Puede considerarse un objeto mediante el cual otros objetos pueden anunciar sus capacidades y concordar sus necesidades con capacidades anunciadas. Anunciar una capacidad u ofrecer un servicio se denomina «exportación». Concordar necesidades o descubrir servicios se denomina «importación». La exportación y la importación facilitan el descubrimiento dinámico de servicio y la posterior vinculación a los mismos.

Para exportar, un objeto hace al intermediario una descripción de un servicio, y le da la posición de una interfaz en la que el servicio está disponible. Para importar, un objeto pide al intermediario un servicio que tenga ciertas características. El intermediario confronta las descripciones de servicios y responde al importador dándole las posiciones de las interfaces del servicio concordado. El importador puede entonces interactuar con un servicio concordado. Estas interacciones se muestran en la figura 1.



Secuencia de interacciones:
1. Exportación
2. Importación
3. Interacción del servicio

Figura 1 – Interacción entre el intermediario y sus clientes

La interacción del servicio podría desligarse de las interacciones de intermediación (importación y exportación) mediante el modelado explícito de un objeto proveedor del servicio y un objeto usuario del servicio, lo cual implicaría interacciones entre el proveedor del servicio y el exportador y entre el importador y el usuario del servicio, que son acciones de intermediación, que se definen en la Rec. UIT-T X.902 | ISO/CEI 10746-2. Sin embargo, estas interacciones implicadas no necesitan conformarse a esta Especificación.

El gran número de ofertas de servicio en todo el mundo y los diferentes requisitos de los usuarios del servicio de intermediación harán inevitable que este servicio se desglose y que las ofertas de servicio se hagan por divisiones.

Cada división, en primera instancia, satisfará las necesidades de intermediación de una comunidad de clientes (exportadores e importadores). Cuando un cliente necesite para sus actividades de intermediación un ámbito más amplio que el proporcionado por una división, accederá a otras divisiones directa e indirectamente. Directamente significa que el cliente interactúa con los intermediarios que tratan estas divisiones. Indirectamente significa que el cliente interactúa con un intermediario solamente, que a su vez interactúa con otros responsables de otras divisiones. Esta última posibilidad se denomina federación de intermediarios. En algunos casos pueden requerirse interceptores entre intermediarios federados.

Un usuario de un intermediario, que interopera con otros, puede asociarse con uno sólo y puede acceder transparentemente a las ofertas de servicio de otros intermediarios con los que aquel intermediario puede interoperar.

Así, la función intermediación en un entorno ODP permite:

- objetos para exportar (anunciar) servicios;
- objetos para importar información sobre uno o más servicios exportados, con arreglo a ciertos criterios;
- federación de intermediarios.

5.1 Diversidad y escalabilidad

El concepto de intermediación para descubrir nuevos servicios es aplicable a una amplia gama de casos. Un intermediario puede contener un gran número de ofertas de servicio y su implementación puede tener tendencia a basarse en una base de datos. O bien, un intermediario puede contener unas pocas ofertas solamente y ser implementable como un intermediario residente en memoria. Estos dos casos presentan diferentes cualidades: disponibilidad e integridad en el primer caso y rendimiento en el segundo. La variación en estos casos ilustra la necesidad de escalabilidad, tanto hacia arriba en sistemas muy grandes como hacia abajo en sistemas pequeños y rápidos.

Para descubrir cualquier oferta arbitraria de servicio, un intermediario necesita que todas las ofertas le resulten, en algún sentido, visibles. Una división no puede contener todas las ofertas, y muchas estarán necesariamente contenidas en otras divisiones, por lo cual, además de un cierto número de ofertas, un intermediario debe poseer información sobre otras divisiones. Sin embargo, no hay necesidad de que un intermediario conozca todas las demás divisiones. Parte de este conocimiento puede obtenerse indirectamente a través de otros intermediarios.

La división del espacio de oferta y el limitado conocimiento contenido en una división sobre otras divisiones constituye la base del cumplimiento de los requisitos de distribución y contextualización de la función intermediación.

5.2 Intermediarios de enlace

Las exigencias de contextualizar el espacio de oferta y de distribuir la función intermediación se satisfacen enlazando los intermediarios entre sí. Al enlazar con otros, cada intermediario pone el espacio de oferta de aquéllos implícitamente disponible a sus propios clientes.

Cada intermediario tiene su horizonte limitado a aquellos otros intermediarios con los que está explícitamente enlazado. Como esos están a su vez enlazados con otros más, el número de intermediarios puede alcanzar un gran número a partir de uno solo. Los intermediarios se enlazan para formar un gráfico directo con la información que describe el gráfico distribuido entre los intermediarios. Este gráfico se denomina gráfico de intermediación.

Los enlaces pueden cruzar fronteras de dominio: administrativa, tecnológica o de otra índole. La intermediación puede por tanto ser un sistema federado, es decir un sistema que abarca muchos dominios.

5.3 Política

Para cumplir los diversos requisitos que probablemente se impongan a la función intermediación, son necesarios algunos grados de libertad cuando se especifica el comportamiento de un objeto intermediario. Para conseguirlo, y seguir

cumpliendo no obstante los objetivos de esta Especificación, se utiliza el concepto de política para establecer un marco que describa el comportamiento de cualquier sistema de intermediación conforme ODP.

Esta Especificación identifica cierto número de políticas y les da una semántica. Cada política determina en parte el comportamiento de un intermediario. Cuando alega conformidad, una implementación puede necesitar especificar qué combinación de políticas asegurarán un comportamiento conforme.

Las políticas pueden comunicarse durante la interacción, en cuyo caso guardan relación con una expectativa sobre el comportamiento posterior.

6 Especificación de empresa de la función intermediación

El alcance de una especificación de empresa se define en el RM-ODP parte 3: arquitectura (véase la Rec. UIT-T X.903 | ISO/CEI 10746-3). La especificación de empresa identifica los objetivos y los enunciados de política que rigen las actividades de una función intermediación.

El objetivo de la función intermediación es proporcionar el medio de ofrecer y descubrir instancias de un determinado tipo de servicio, con determinadas características.

Una comunidad de intermediación consta de miembros que tienen diferentes roles, por ejemplo, intermediario, exportador e importador. Un objeto puede tener varios roles dentro de la misma comunidad. Por ejemplo, un objeto puede ser un importador y un exportador.

Las actividades de intermediación de la comunidad son exportaciones de servicios e importaciones de servicios. Estas actividades son regidas por un conjunto de políticas de la comunidad de intermediación. Una actividad de importación de servicios puede propagarse de una comunidad de intermediación a otra. En dicho caso los dominios asociados con esos dos intermediarios están federados. Estas fronteras de dominio de intermediario pueden coincidir con otro dominio (por ejemplo, dominio de tipo o dominio de política de seguridad).

Una política es un conjunto de reglas con un objetivo determinado. Cada regla constriñe algunos aspectos del comportamiento de un intermediario de manera consecuente con el objetivo común. Los miembros de la comunidad de intermediación están obligados a cumplir las reglas de las políticas. Estas reglas proporcionan las directrices para que las decisiones satisfagan los objetivos de la comunidad. Las reglas no se prescriben en esta Especificación. La especificación de empresa identifica el conjunto de políticas que limitan al intermediario en cierto tipo de comportamiento. Las políticas identificadas proporcionan un marco en el cual puede implementarse o configurarse el comportamiento del objeto intermediario.

6.1 Comunidades

6.1.1 comunidad de intermediación: Comunidad de objetos establecidos con el fin de intermediar y regidos por una política de intermediación. Los objetos desempeñan los roles enumerados en 6.2.

Una comunidad de intermediación (a un nivel de abstracción) puede perfeccionarse para conseguir cierto número de comunidades de intermediación interfuncionantes a un segundo nivel de abstracción más detallado. A reserva de la política de la comunidad, el interfuncionamiento de las comunidades de intermediación al nivel detallada es capaz de mantener la impresión de una única comunidad abstracta, permitiendo a objetos con roles de intermediario, importador o exportador de una subcomunidad interactuar con objetos de otra subcomunidad.

6.2 Roles (cometidos)

Los objetos pueden desempeñar los siguientes roles dentro de una comunidad de intermediación:

6.2.1 intermediario: Rol que registra ofertas de servicio de objetos exportadores y retorna ofertas de servicio a petición de objetos importadores con arreglo a algunos criterios.

6.2.2 exportador: Rol que registra ofertas de servicio con el objeto intermediario.

6.2.3 importador: Rol que obtiene ofertas de servicio, satisfaciendo algunos criterios, del objeto intermediario.

6.2.4 administrador intermediario: Rol que define, gestiona e impone la política de intermediación del objeto intermediario. El administrador intermediario es el objeto controlador de un dominio de intermediario (el intermediario y su conjunto de ofertas de servicio).

6.2.5 oferta de servicio: Rol que mantiene la descripción de un servicio.

NOTA – La descripción puede ser la base de un contrato futuro.

6.3 Actividades

Las actividades siguientes son pertinentes para una comunidad de intermediación.

6.3.1 exportación de servicios: Cadena de acciones ejercidas por un objeto exportador y el objeto intermediario que establecen y termina una relación en la que se permite al objeto intermediario proporcionar la oferta de servicios de un objeto exportador a un grupo de objetos importadores.

6.3.2 importación de servicios: Cadena de acciones entre un objeto importador y el objeto intermediario en la que el objeto importador obtiene cierto número de ofertas de servicios que cumplen algunos criterios.

6.4 Políticas

Los comportamientos de los objetos de empresa dentro de una comunidad de intermediación son regidos por las políticas de la comunidad de intermediación. Algunas políticas rigen actividades de intermediación, y algunas políticas imponen constricciones sobre otros aspectos del comportamiento de un intermediario y otros roles en la comunidad de intermediación, de manera consecuente con el objetivo común de la comunidad. Cuando una actividad exige interacciones entre objetos, la política resultante será un compromiso entre las políticas de los objetos interactuantes. El compromiso se alcanzará mediante una forma de arbitraje.

NOTA – Por ejemplo, un objeto intermediario puede ser regido por políticas tales que esté obligado a propagar una búsqueda a una profundidad de 2 enlaces, pero también se le permite terminar una búsqueda después de propagar una búsqueda a una profundidad de 1 enlace. Si se permite (u obliga) al objeto intermediario satisfacer el requisito de un importador en relación con la profundidad de enlaces a atravesar para una búsqueda, se necesitan algunas reglas para ejercer arbitraje entre políticas contrapuestas.

6.4.1 Política de actividad de exportación: La política de actividad de exportación es un conjunto de reglas relacionadas con la actividad de exportación de servicios (es decir, el ofrecimiento de servicios para que puedan posteriormente ser descubiertos por otros objetos).

La política puede incluir, entre otras cosas:

- la obligación de que una oferta de servicios se describa de una determinada forma;
- la prohibición de que determinadas actividades de importación de servicios descubran la oferta de servicios;
- la obligación de que una oferta de servicios que proporcione reglas se evalúe como parte de una actividad de importación de servicios.

Cada exportador puede tener su propia política de exportación, política que describiría las expectativas del exportador en una exportación de servicios. Por tanto, la actividad de exportación de servicios viene regida por la política de exportación del intermediario y por la política de exportación del exportador.

6.4.2 Política de actividad de importación: Conjunto de reglas relacionadas con la actividad de importación de servicios (es decir, el intento de descubrir servicios ofrecidos que cumplan un requisito especificado).

Esta política puede incluir, entre otras cosas:

- la obligación de limitar la utilización de recursos, incluida la duración de la actividad;
- permisos para propagar la importación de servicios a una o más comunidades de intermediación interfuncionantes.

6.4.3 Política de arbitraje: Conjunto de reglas para arbitrar sobre reglas contrapuestas que surjan durante las actividades de intermediación.

Esta política puede incluir, entre otras cosas, la obligación de arbitrar a favor de las reglas del objeto intermediario sobre:

- la utilización de recursos durante la importación de servicios;
- las actividades de propagación de la importación de servicios.

6.4.4 Política de aceptación de ofertas de servicios: Conjunto de reglas que restringen el conjunto de ofertas de servicios que serán aceptadas por el intermediario.

6.4.5 Política de gestión de tipos: Conjunto de reglas relativas a la especificación de tipos y a la relación entre tipos.

NOTA 1 – La política puede ser remitirse a una función depositaria de tipos con respecto a uno de estos aspectos o a ambos.

NOTA 2 – Ejemplos serían utilizar la equivalencia de nombres o utilizar subtipificación de signaturas en la concordación de tipos.

6.4.6 Política de búsqueda: Conjunto de reglas que guían la búsqueda de ofertas de servicios adecuados mediante el sistema de intermediación.

6.5 Reglas de estructuración

6.5.1 Reglas de comunidad

En una comunidad de intermediación debe haber un objeto que asuma un rol intermediario (un objeto intermediario). Convertirse en miembro de una comunidad de intermediación permite a un objeto interactuar con el objeto intermediario en un rol importador o exportador. Un objeto puede asumir el rol exportador, el rol importador, o ambos.

Una «empresa» puede incluir múltiples comunidades de intermediación. Un objeto puede ser miembro de múltiples comunidades de intermediación. El objeto intermediario de una comunidad puede asumir un rol importador o exportador dentro de otra comunidad de la cual es miembro.

La comunidad puede abarcar varios dominios en aspectos de seguridad, tipos, gestión, remuneración, etc.

Cada intermediario, junto con sus ofertas de servicios, es un dominio de intermediario. De este modo, un conjunto de dominios de intermediario que interopera dentro de una comunidad de intermediación es una federación de intermediarios.

NOTA – Los dominios de intermediarios federados no siempre requieren interceptores situados en su frontera desde un punto de vista de ingeniería.

6.5.2 Reglas de transferencia

Los objetos exportadores pueden exportar ofertas de servicios que proporcionan en sus propias interfaces, o exportar ofertas de servicios proporcionados por otro objeto proveedor de servicios.

Los objetos importadores pueden importar ofertas de servicios para su propio uso o para su uso por otros objetos usuarios de servicios.

6.5.3 Delimitación de las reglas de autoridad

Cada objeto administrador intermediario de una comunidad de intermediación tiene control completo sobre su propio objeto intermediario.

El objeto exportador es responsable de la exactitud de sus ofertas de servicios.

Para que los intermediarios sean miembros de una federación establecida de intermediarios:

- un intermediario no está obligado a efectuar una actividad iniciada por otro intermediario;
- cada intermediario debe tener autonomía completa con relación a sus propias políticas de intermediario.

En particular, cada intermediario determina sus propias políticas de búsqueda de intermediarios entre el grupo de intermediarios interfuncionantes.

6.5.4 Reglas de calidad de servicio

El objeto intermediario no es responsable de la calidad de los servicios descritos en las ofertas de servicios.

Un objeto intermediario puede estar obligado a asegurar la supresión de las ofertas de servicios en el momento oportuno.

NOTA – Dos ejemplos para conseguirlo son:

1) Una política de aceptación de ofertas de servicios de un intermediario puede obligar a que las ofertas de servicios tengan una fecha de expiración. Al objeto intermediario se le permite suprimir ofertas de servicios expiradas.
2) Una política de importación de un intermediario puede prohibir al objeto intermediario retornar ofertas de servicios que han expirado en el momento de una importación.

6.5.5 Reglas de concordancia

Una importación de servicios requiere una comprobación de los tipos de signaturas de la interfaz computacional. Además, puede exigir otros niveles de comprobación de relaciones de subtipos o supertipos, compatibilidad de comportamientos y constricciones de entorno. Puede también preverse otra comprobación de aspectos de empresa, de información, de ingeniería y de tecnología.

7 Especificación de información de la función intermediación

7.1 Sinopsis

El alcance de la especificación de información se describe en RM-ODP parte 3: arquitectura (véase la Rec. UIT-T X.903 | ISO/CEI 10746-3). Esta especificación de información describe los tipos de información y las relaciones entre los mismos que se requieren para definir la función intermediación ODP. Utiliza el lenguaje de información definido

en RM-ODP y, cuando es apropiado, interpreta el lenguaje en términos de la notación de especificación formal Z. Los párrafos de la notación formal están entremezclados con texto inglés en el estilo Z habitual.

La especificación de información de esta cláusula define:

- conceptos básicos para la información utilizada en esta Especificación;
- esquemática estática, invariante y dinámica para esta Especificación.

7.2 Conceptos básicos

7.2.1 Interfaces

Un servicio es ofrecido en una interfaz. Existe necesidad de que una oferta de servicio identifique el tipo de signatura de interfaz y el identificador de interfaz de la interfaz del servicio.

7.2.1.1 Tipo de signatura de interfaz

El tipo de signatura de interfaz identifica la signatura de las interfaces de objetos.

En Z, los tipos de signaturas de interfaz se definen formalmente introduciendo un determinado conjunto para representar los valores que pueden adoptar:

[InterfaceSignatureType]

7.2.1.2 Identificador de interfaz

Un identificador de interfaz identifica una interfaz en la que está disponible o se requiere un servicio.

En Z, los identificadores de interfaces vienen definidos formalmente por un determinado conjunto.

7.2.2 Tipo de servicio

Un servicio es un conjunto de capacidades proporcionado por un objeto en una interfaz computacional. Un servicio es una instancia de un tipo de servicio.

Una definición de tipo de servicio consta de un tipo de signatura de interfaz, un conjunto de definiciones de propiedad de servicio, y un conjunto de reglas acerca de los modos de las propiedades del servicio.

Las definiciones de propiedad de servicio se describen explícitamente en la especificación formal en términos de nombres, tipos de valores y modos. Los modos válidos son:

- normal (lectura y escritura pero presencia opcional) ;
- lectura solamente (lectura pero presencia opcional) ;
- obligatorio (lectura y escritura presencia obligatoria) ; y
- lectura solamente, obligatorio (lectura solamente, presencia obligatoria).

Un tipo de valor es un conjunto de valores.
[Name, Value]

ValueType == ⦶Value

Mode ::= { normal, readonly, mandatory, readonly_mandatory }

Las propiedades del servicio contienen información sobre aspectos computacionales (tales como el comportamiento del entorno de una interfaz) y también describen los aspectos de tecnología, ingeniería, información y empresa del servicio.

La definición formal del tipo de servicio se da en el esquema Z *ServiceType*. Agrupa un tipo de signatura de interfaz y un conjunto de definiciones de propiedades del servicio.

ServiceType

signature : *InterfaceSignatureType*
prop_defs : *Name* ⇻ (*ValueType* ⇸ *Mode*)

En Z, se definen funciones para extraer el conjunto de nombres de propiedades que deben estar presentes (es decir, son *obligatorio* o *lectura solamente_obligatorio*) y el conjunto de nombres de propiedades que no pueden modificarse (son *lectura solamente*, o *lectura solamente_obligatorio*). Estas dos funciones se definen formalmente como sigue:

$$
\begin{array}{l}
\textbf{\textit{mandatory\_props}} : \textit{ServiceType} \rightarrow \mathbb{P}\ \textit{Name} \\
\textbf{\textit{readonly\_props}} : \textit{ServiceType} \rightarrow \mathbb{P}\ \textit{Name} \\
\hline
\forall\ s : \textit{ServiceType} \bullet \textbf{\textit{mandatory\_props}}\ s = \\
\quad \{\ n : \textit{Name} \mid \textit{second}\ (s.\textit{prop\_defs}\ n) \in \{\textit{mandatory}, \textit{readonly\_mandatory}\}\} \\
\forall\ s : \textit{ServiceType} \bullet \textbf{\textit{readonly\_props}}\ s = \\
\quad \{n : \textit{Name} \mid \textit{second}\ (s.\textit{prop\_defs}\ n) \in \{\textit{readonly}, \textit{readonly\_mandatory}\}\}
\end{array}
$$

7.2.3 Reglas de subtipificación de tipos de servicios

Las reglas generales para la subtipificación de servicios para un intermediario conforme son las siguientes:

En el caso más general, un tipo de servicio **b** es un subtipo del tipo de servicio **a**, si y sólo si:

- el tipo de signatura de interfaz de **b** es un subtipo de signatura de interfaz de **a**;
- todas las propiedades denominadas de **a** están en **b**;
- todas las propiedades denominadas de **a** tienen un valor tipo que es un supertipo de la propiedad idénticamente denominada en **b**;
- todas las propiedades denominadas de **a** tienen un modo que es un supertipo del modo de la propiedad idénticamente denominada en **b**.

NOTA – Las reglas anteriores son equivalentes a las reglas normales de subtipificación de interfaz ODP, si las propiedades son vistas como operaciones, con el tipo y el modo como argumentos de retorno de las operaciones.

La representación Z exige que se definan tres relaciones para representar la subtipificación de signaturas de interfaz, la supertipificación de valores y la supertipificación de modos. Las reglas de subtipificación de signaturas de interfaz son las indicadas en la Rec. UIT-T X.903 | ISO/CEI 10746-3, y no siguen definiéndose aquí. Se dan definiciones formales para las relaciones de supertipificación a través de los modos y tipos de valores.

$$
\begin{array}{l}
\_\ \textbf{\textit{is\_sig\_subtype\_of}}\ \_ : \textit{InterfaceSignatureType} \leftrightarrow \textit{InterfaceSignatureType} \\
\_\ \textbf{\textit{is\_value\_supertype\_of}}\ \_ : \textit{ValueType} \leftrightarrow \textit{ValueType} \\
\_\ \textbf{\textit{is\_mode\_supertype\_of}}\ \_ : \textit{Mode} \leftrightarrow \textit{Mode} \\
\hline
\forall\ a,b : \textit{Mode} \bullet a\ \textbf{\textit{is\_mode\_supertype\_of}}\ b \subseteq \\
\quad (a,b) \in \{(\textit{normal},\textit{readonly}),(\textit{normal},\textit{mandatory}),(\textit{normal}, \textit{readonly\_mandatory}), \\
\qquad (\textit{readonly},\textit{readonly\_mandatory}), (\textit{mandatory},\textit{readonly\_mandatory})\} \\
\forall\ a,b : \textit{ValueType} \bullet a\ \textbf{\textit{is\_value\_supertype\_of}}\ b \subseteq b \supseteq a
\end{array}
$$

$$
\begin{array}{l}
\_\_\ \textbf{\textit{is\_subtype\_of}}\ \_\_ : \textit{ServiceType} \leftrightarrow \textit{ServiceType} \\
\hline
\forall\ a,b : \textit{ServiceType} \bullet b\ \textbf{\textit{is\_subtype\_of}}\ a \subseteq \\
\quad b.\textit{signature}\ \textbf{\textit{is\_sig\_subtype\_of}}\ a.\textit{signature}\ \wedge \\
\quad \textit{dom}\ a.\textit{prop\_defs} \supseteq \textit{dom}\ b.\textit{prop\_defs}\ \wedge \\
\quad (\forall\ n : \textit{dom}\ a.\textit{prop\_defs} \bullet \\
\qquad \textit{first}\ (a.\textit{prop\_defs}\ n)\ \textbf{\textit{is\_value\_supertype\_of}}\ \textit{first}\ (b.\textit{prop\_defs}\ n) \wedge \\
\qquad \textit{second}\ (a.\textit{prop\_defs}\ n)\ \textbf{\textit{is\_mode\_supertype\_of}}\ \textit{second}\ (b.\textit{prop\_defs}\ n))
\end{array}
$$

7.2.4 Oferta de servicio

Una oferta de servicio anuncia un servicio. Es una aserción efectuada por un exportador acerca de un servicio que se ofrece para su utilización por otros objetos en la interfaz computacional. Consta de una instanciación de un tipo de servicio, un identificador de la oferta de servicio y un identificador de la interfaz mediante la cual puede utilizarse el servicio. Una oferta de servicio puede también incluir un conjunto de valores de propiedades de oferta de servicio.

Para Z, se introduce un conjunto determinado para representar los identificadores de oferta de servicio que son inequívocos dentro de una comunidad de intermediación.

[ServiceOfferIdentifier]

La definición Z formal de una oferta de servicio viene dada por el esquema *ServiceOffer*. Cada oferta de servicio debe satisfacer su tipo de servicio, es decir, la oferta de servicio debe tener un valor para todas las propiedades definidas por el tipo de servicio como obligatorio y lectura solamente obligatorio; y todas estas propiedades que son propiedades del tipo de servicio deben tener valores extraídos de los conjuntos definidos en el tipo de servicio.

ServiceOffer

$$
\begin{array}{|l}
\hline
service\_type : ServiceType \\
prop\_vals : Name \pfun Value \\
interface\_identifier : InterfaceIdentifier \\
service\_offer\_identifier : ServiceOfferIdentifier \\
\hline
\textbf{mandatory\_props}\ service\_type \finset dom\ prop\_vals \\
\forall n : dom\ service\_type.prop\_defs \rightarrowtail dom\ prop\_vals \bullet \\
\quad prop\_vals\ n \in first\ (service\_type.prop\_defs\ n) \\
\hline
\end{array}
$$

7.2.5 Criterios y constricciones

Las políticas desde el punto de vista de empresa son representadas por criterios y constricciones en el punto de vista de información.

Hay tres aspectos para especificar el conjunto de ofertas de servicio que son resultados aceptables de una acción buscar o seleccionar. Dos de éstas son relaciones de filtrado sobre las propiedades del servicio y las propiedades de oferta de servicio. La primera define las propiedades esenciales que no pueden violarse para concordar. La segunda es una preferencia que actúa como filtro de selección cuando hay muchas ofertas que concuerdan con las propiedades esenciales. El tercer aspecto es una relación de determinación de alcance que restringe el conjunto de ofertas de servicio que han de compararse con las reglas de concordancia. Estas especificaciones pueden ser proporcionadas por el importador y por el sistema de intermediación, siendo la especificación final una combinación de las dos. El sistema de intermediación se define en 7.3.

7.2.5.1 Criterios de concordancia

Reglas que se aplican al conjunto total de ofertas de servicio para obtener un conjunto más pequeño de ofertas de servicio aceptables.

En Z los criterios de concordancia pueden expresarse como el conjunto de todas las ofertas de servicio posibles que satisfarían las reglas de concordancia. El proceso de aplicar estas reglas a un conjunto de ofertas de servicio puede representarse tomando la intersección de dos conjuntos.

$MatchingCriteria == \mathbb{P} ServiceOffer$

7.2.5.2 Criterios de preferencia

Reglas que se aplican al conjunto de ofertas de servicio aceptables para obtener una secuencia ordenada de ofertas de servicio.

En Z los criterios de preferencia pueden expresarse como una función total a partir de conjuntos de ofertas de servicio a una secuencia de ofertas de servicio. La aplicación de estas reglas se describe formalmente en 7.5.10.

PreferenceCriteria == ⦶*ServiceOffer* ⤖ **seq** *ServiceOffer*

7.2.5.3 Criterios de alcance

Reglas que restringen el conjunto de ofertas de servicio que han de compararse con las reglas de concordancia.

En Z los criterios de alcance pueden expresarse como un conjunto de ofertas de servicio.

ScopeCriteria == ⦶*ServiceOffer*

7.2.5.4 Criterios de bordes

Reglas que restringen el conjunto de nodos alcanzables desde un nodo determinado.

En Z, los criterios de bordes pueden expresarse como asociaciones:

EdgeCriteria == Node x Node

7.2.5.5 Constricción de concordancia de intermediario

Constricción sobre los criterios de concordancia impuesta por la política del sistema de intermediación.

En Z las constricciones de concordancia pueden especificarse de la misma manera que los criterios de concordancia. La aplicación de estas constricciones puede representarse por intersección de conjuntos.

7.2.5.6 Constricción de preferencia de intermediario

Constricción sobre los criterios de preferencia impuesta por la política del sistema de intermediación.

En Z, las constricciones de preferencia pueden especificarse de la misma manera que los criterios de preferencia. La aplicación de esta función se describe en 7.5.10.

PreferenceConstraint == PreferenceCriteria ⤖ *PreferenceCriteria*

7.2.5.7 Constricción de alcance de intermediario

Constricción sobre los criterios de alcance impuesta por la política de intermediario.

En Z, las constricciones de alcance son representadas por un conjunto de ofertas de servicio como para los criterios de alcance.

7.2.5.8 Constricciones del sistema de intermediación

Para la especificación Z formal es conveniente agrupar las constricciones del sistema de intermediación en el esquema *TradingSystemConstraints*. Esta definición se utiliza en 7.5.9.

TradingSystemConstraints

trader_matching : *MatchingCriteria*
trader_scope : *ScopeCriteria*
trader_preference : *PreferenceConstraint*

7.2.6 Petición de búsqueda

Una petición de búsqueda es una especificación de la política de importador que se aplica a una determinada acción buscar.

En Z una petición de búsqueda puede modelarse como un esquema compuesto por los criterios de concordancia, alcance y preferencia del importador, y el tipo de servicio del servicio deseado. Todas las ofertas que satisfacen los requisitos de concordancia del importador deben tener un tipo de servicio que es un subtipo del tipo de servicio especificado. Esta definición se utiliza en 7.5.9.

Search Request

importer_matching : *MatchingCriteria*
importer_scope : *ScopeCriteria*
importer_preference : *PreferenceCriteria*
importer_service_type : *ServiceType*

$\forall s$:*importer_matching* • *s.service_type* **is_subtype_of** *importer_service_type*

7.3 Esquema invariante

El espacio ofertas de servicio será dividido y asociado con cualquier otra división será conocimiento de un número limitado de otras divisiones. Este patrón se repite centrándose en cualquier división dada. En la especificación de información esto se modela como un gráfico directo, en el que los nodos representan las divisiones y los bordes representan el conocimiento relativo a las divisiones. Este gráfico se denomina gráfico de intermediación.

Puede haber cualquier número de bases sobre las cuales dividir el espacio de oferta, además de la distribución. Por ejemplo, la división puede basarse en:

- propiedades de la ubicación (por ejemplo, arquitectura de máquina);
- propiedades de las ofertas de servicios (por ejemplo, una clasificación de seguridad);
- propiedades del servicio (por ejemplo, su disponibilidad).

Un borde puede tener propiedades que describan la percepción de una división cuando se ve desde otra división.

Los componentes del sistema de intermediación son:

- un conjunto (*offers*) de ofertas de servicio disponibles para importación;
- un conjunto (*nodes*) de nodos en los que se dividen estas ofertas de servicio;
- una relación (*edges*) entre nodos para representar los bordes del gráfico de intermediación, que rigen la propagación de las búsquedas;
- conjuntos (*edge_properties*) de propiedades asociadas con los bordes;
- las ofertas de servicio se distinguen por sus identificadores de oferta de servicio, que son únicos e inequívocos. Esto es captado por los invariantes del esquema *TradingSystem* en Z.

Los distintos nodos se definen formalmente en Z introduciendo un determinado conjunto.

[Node]

Todas las ofertas de servicios deben asignarse a un nodo, y sólo uno. No se permite superposición entre los nodos ni relaciones de subconjuntos. Esta restricción es captada por la función parcial *partition* (*división*) que hace corresponder cada oferta de servicio en el sistema de intermediación a un solo nodo. Un exportador puede exportar el mismo servicio a más de un nodo, creando otra oferta de servicio con un identificador de oferta de servicio diferente.

El punto de vista computacional establece correspondencia de nodos a intermediarios. Esta correspondencia es de uno a uno (cada nodo en el punto de vista de información es un objeto intermediario único). Por tanto, en el punto de vista computacional, un borde corresponde a una interfaz computacional entre intermediarios.

El siguiente esquema Z representa el estado del sistema de intermediación.

TradingSystem

$offers : \mathbb{P}\ ServiceOffer$
$nodes : \mathbb{P}\ Node$
$partition : ServiceOffer \rightarrow Node$
$edges : Node \leftrightarrow Node$
$edge\_properties : (Node \times Node) \rightarrow \mathbb{P}\ Property$

$dom\ partition = offers$
$ran\ partition \subseteq nodes$
$dom\ edges \triangleleft ran\ edges \subseteq nodes$
$dom\ edge\_properties = edges$
$\forall p,q : ServiceOffer \bullet p.service\_offer\_identifier = q.service\_offer\_identifier \Rightarrow p = q$

7.4 Esquema estático

El esquema estático aplica el estado del sistema de intermediación en un determinado lugar de tiempo.

Para la coherencia y la compleción de la especificación Z formal, se incluye la acción de inicializar un sistema de intermediación.

Cuando se crea un sistema, tiene el valor nulo para cada componente de estado:

Initialize

$TradingSystem'$

$offers' = \emptyset$
$nodes' = \emptyset$
$partition' = \emptyset$
$edges' = \emptyset$
$edge\_properties' = \emptyset$

7.5 Esquemas dinámicos

Se presentan aquí los esquemas de información dinámicos que describen los cambios de estado asociados con las siguientes acciones:

- Exportar (Export) – Añadir una oferta de servicio al espacio de oferta de servicio del sistema de intermediación.
- Retirar (Withdraw) – Retirar una oferta de servicio del espacio de oferta de servicio del sistema de intermediación.
- Modificar oferta (Modify Offer) – Cambiar los valores propiedad del servicio y propiedad de oferta de servicio asociados con una oferta de servicio pero preservando el identificador de oferta de servicio.
- Añadir borde (Add Edge) – Añadir a un borde al conjunto de bordes del sistema de intermediación.
- Suprimir borde (Remove Edge) – Suprimir un borde del conjunto de bordes del sistema de intermediación
- Modificar borde (Modify Edge) – Cambiar las propiedades de un borde.
- Añadir nodo (Add Node) – Añadir un nodo al conjunto de nodos del sistema de intermediación.
- Suprimir nodo (Remove Node) – Suprimir un nodo del conjunto de nodos del sistema de intermediación.
- Buscar (Search) – Buscar el subconjunto de ofertas de servicio que satisfacen algunos criterios de concordancia y criterios de determinación de alcance.
- Seleccionar (Select) – Especialización útil de buscar que retorna una secuencia de ofertas de servicio con arreglo a alguna constricción de preferencia.

7.5.1 Exportar

El esquema dinámico Export describe el comportamiento de la adición de una oferta de servicio al sistema de intermediación.

La exportación exitosa de una oferta de servicio es como sigue. La nueva oferta de servicio se añade al conjunto existente de ofertas de servicio y se asocia con un solo nodo. La fuente del componente identificador de oferta de servicio de la oferta de servicio no se identifica: es generada por el intermediario, en lugar del exportador. El identificador de oferta de servicio se devuelve al exportador. Las relaciones de bordes, y las propiedades asociadas con los bordes, no varían.

La condición previa para esta acción es que el nuevo identificador de oferta de servicio no se utilice en ese momento en el sistema de intermediación. Otra condición previa exige que el nodo con el que ha de asociarse la oferta exista en el sistema de intermediación.

En Z, el esquema *Export* representa el comportamiento correspondiente.

ExportOK

ΔTradingSystem
new_offer? : *ServiceOffer*
node? : *Node*
service_offer_identifier! : *ServiceOfferIdentifier*

$\forall s : offers \bullet s.service\_offer\_identifier \neq new\_offer?.service\_offer\_identifier$
$node? \notin nodes$
$offers' = offers \cup \{new\_offer?\}$
$partition' = partition \cup \{new\_offer? \mapsto node?\}$
$service\_offer\_identifier! = new\_offer?.service\_offer\_identifier$
$nodes' = nodes$
$edge\_properties' = edge\_properties$
$edges' = edges$

Si las condiciones previas del esquema *ExportOK* no se cumplen, el estado del sistema de intermediación permanece invariable. En Z, éste se capta en el siguiente esquema de error, la condición previa del cual es la negación de la condición previa de *ExportOK*.

ExportError

ΞTradingSystem
new_offer? : *ServiceOffer*
node? : *Node*

$\exists s : offers \bullet s.service\_offer\_identifier = new\_offer?.service\_offer\_identifier \lor$
$node? \notin nodes$

Export prosperará o fracasará según las condiciones previas de los esquemas Z arriba indicados.

Export ≙ *ExportOK* ∨ *ExportError*

7.5.2 Retirar

El esquema dinámico Withdraw suprime una oferta de servicio del sistema de intermediación.

La oferta es retirada del conjunto de ofertas. Los bordes y propiedades de bordes no varían. Adviértase que este comportamiento no especifica cómo se inició la acción. Así, el comportamiento se aplica siempre que una oferta es retirada por el intermediario, el exportador o algún otro agente autorizado.

La condición previa para esta acción es que la oferta de servicio debe existir en el sistema de intermediación.

En Z, el esquema *Withdraw* representa el comportamiento correspondiente.

WithdrawOfferOK

ΔTradingSystem
old_offer? : *ServiceOffer*

---

old_offer? ∉ *offers*
offers' = *offers* \ {*old_offer?*}
partition' = {*old_offer?*} ⨟ *partition*
nodes' = *nodes*
edges' = *edges*
edge_properties' = *edge_properties*

Si no se cumplen las condiciones previas de *WithdrawOfferOK*, el estado del sistema de intermediación permanece invariable.

WithdrawOfferError

ΞTradingSystem
old_offer? : *ServiceOffer*

---

old_offer? ∈ *offers*

WithdrawOffer prosperará o fracasará según las condiciones previas de los esquemas Z arriba indicados.

WithdrawOffer ≙ *WithdrawOfferOK* ▷ *WithdrawOfferError*

7.5.3 Modificar oferta

El esquema dinámico ModifyOffer define el comportamiento de modificar las propiedades del servicio y las propiedades de oferta de servicio asociadas con una oferta de servicio. Este comportamiento no es simplemente Withdraw seguido por Export, ya que también garantiza que se preserva el identificador de oferta de servicio.

Una oferta de servicio puede modificarse de tres maneras:

- las propiedades pueden suprimirse, en la medida en que no sean propiedades obligatorias;
- pueden añadirse nuevas propiedades, a condición de que aún no tengan asignado un valor en la oferta de servicio;
- las propiedades existentes pueden ser actualizadas, a condición de que no sean de lectura solamente.

En Z, el esquema *ModifyServiceOffer* representa el comportamiento correspondiente. La definición se da en varios pasos. En primer lugar, se da una definición que modifique una oferta de servicio sustituyendo los valores de propiedades del servicio, a condición de que se cumplan las constricciones citadas. Los otros componentes de la oferta no son afectados: en particular se preserva el identificador de oferta de servicio. Adviértase que este comportamiento no especifica cómo se inició la acción. Así, el comportamiento se aplica siempre que una oferta es modificada por el intermediario, el exportador o algún otro agente autorizado.

ModifyServiceOfferOK

ΔServiceOffer
delete? : ℙ *Name*
new? : *Name* ⇸ *Value*
update? : *Name* ⇸ *Value*

---

delete? ◁ ***mandatory_props*** *service_type* = ∅
dom update? ◁ ***readonly_props*** *service_type* = ∅
delete? ⩤ *dom update?* ⩥ *dom prop_vals*
dom new? ◁ *dom prop_vals* = ∅
prop_vals' = (*delete?* ⨟ *prop_vals*) ⩊ *update?* ⩤ *new?*
service_type'.*signature* = *service_type*.*signature*
interface_identifier' = *interface_identifier*
service_offer_identifier' = *service_offer_identifier*

Las condiciones de error para este comportamiento surgen cuando se hace un intento de:

- actualizar o modificar una propiedad inexistente;
- modificar una propiedad lectura solamente u obligatorio lectura solamente; o
- se añade una nueva propiedad que ya existe en las propiedades del servicio.

Si ocurre así, todas las propiedades de la oferta de servicio permanecen invariables, y la signatura de la interfaz tampoco varía.

ModifyServiceOfferError

ServiceOffer
delete? : *Name*
new? : Name *Value*
update? : Name *Value*

---

n : delete? dom update? • n dom prop_vals
delete? ***mandatory_props*** *service_type*
dom update? ***readonly_props*** *service_type*
dom new? *dom prop_vals*

La técnica Z de promoción se emplea ahora para promover el esquema *ModifyServiceOffer* para que sea aplicado a una oferta específica en el sistema de intermediación. La oferta a modificar es identificada por la definición de un esquema de entramación, *ModifyOffer*. El esquema de entramación especifica que:

- el identificador de la oferta a cambiar es conocido por el sistema de intermediación;
- tras la acción, la oferta seleccionada se ha cambiado a un nuevo valor. Todas las demás ofertas no varían y la oferta permanece en el nodo original;
- los bordes y las propiedades de los bordes no varían.

ModifyOffer

TradingSystem
ServiceOffer
modified_offer? : *ServiceOffer*

---

modified_offer? offers
ServiceOffer = modified_offer?
offers = (*offers* \ {*ServiceOffer*}) {*ServiceOffer* }
partition =
({*ServiceOffer*} *partition*) {*ServiceOffer* *partition ServiceOffer*}
edges = *edges*
edge_properties = *edge_properties*
nodes = *nodes*

Las condiciones de error para este comportamiento surgen cuando no se cumple la condición previa de la *ModifyOffer*. Así ocurre cuando se hace un intento por modificar una oferta que no está presente en el sistema de intermediación. El sistema de intermediación permanece en el mismo estado.

ModifyOfferError

TradingSystem
modified_offer? : *ServiceOffer*

---

modified_offer? offers

Por último, el comportamiento *ModifyOffer* se define en términos de *ModifyServiceOffer*, el esquema de entramación ☞*ModifyOffer* (que identifica una determinada oferta a modificar) y las condiciones de error definidas en *ModifyOfferError*. Los componentes de ✍*ServiceOffer* están ocultos para que no aparezcan en la parte declaración de *ModifyOffer*, en línea con los convenios Z habituales.

ModifyOffer ✧ ((*ModifyServiceOfferOK* ⅄ ☞*ModifyOffer*) ⋎ *ModifyOfferError*) \
(*service_type, prop_vals, service_offer_identifier, interface_identifier ,*
*service_type*O*, prop_vals*O *, service_offer_identifier*O*,interface_identifier'*)

7.5.4 Añadir borde

El esquema dinámico AddEdge define el comportamiento asociado con la adición de un borde al gráfico de intermediación. Se añade un borde entre los dos nodos suministrados, y se asocian con este borde las propiedades del nuevo borde. El conjunto de ofertas de servicio y los nodos dentro del sistema de intermediación permanecen invariables.

Las condiciones previas de este esquema dinámico son que existan ambos nodos en el sistema de intermediación, y que no exista todavía ningún borde entre estos dos nodos en el mismo sentido.

En Z, el esquema *AddEdge* representa el comportamiento correspondiente.

AddEdgeOK

✍*TradingSystem*
node1?, node2? : *Node*
new_edge_properties? : ⨀*Property*

{*node1?, node2??*} ⊄ *nodes*
(*node1?, node2??*) ⊄ *edges*
*edges*O = *edges* ⩤ {*node1?* ➇ *node2??*}
*edge_properties*O = *edge_properties* ⩤ {(*node1?, node2??*) ➇ *new_edge_properties?*}
*offers*O = *offers*
*nodes*O = *nodes*
*partition*O = *partition*

Si no se cumplen las condiciones previas de *AddEdgeOK*, el sistema de intermediación permanece invariable.

AddEdgeError

✠*TradingSystem*
node1?, node2? : *Node*

---

node1? ⊄ *nodes* ⋎
node2? ⊄ *nodes* ⋎
(*node1?, node2??*) ⊅ *edges*

AddEdge prosperará o fracasará según las condiciones previas de los esquemas Z arriba indicados.

AddEdge ✧ *AddEdgeOK* ⋎ *AddEdgeError*

7.5.5 Suprimir borde

El esquema dinámico RemoveEdge suprime un borde del gráfico de intermediación. El conjunto de ofertas de servicio y nodos dentro del sistema de intermediación permanece invariable con este esquema dinámico. Las propiedades asociadas con el borde se suprimen de las *edge_properties*.

Las condiciones previas de esta acción son que el borde suministrado esté en el conjunto de bordes existente en ese momento.

En Z, el esquema *RemoveEdge* representa el comportamiento correspondiente.

RemoveEdgeOK

Δ*TradingSystem*
old_edge? : *Node* ⇸ *Node*

---

old_edge? ⦏ *edges*
edges′ = *edges* \ {*old_edge?*}
edge_properties′ = {*old_edge?*} ⩤ *edge_properties*
offers′ = *offers*
nodes′ = *nodes*
partition′ = *partition*

Si no se cumplen las condiciones previas de *RemoveEdgeOK*, el sistema de intermediación permanece invariable.

RemoveEdgeError

Ξ*TradingSystem*
old_edge? : *Node* ⇸ *Node*

---

old_edge? ⦎ *edges*

RemoveEdge prosperará o fracasará según las condiciones previas de los esquemas arriba indicados.

RemoveEdge ✧ *RemoveEdgeOK* ⩒ *RemoveEdgeError*

7.5.6 Modificar borde

El esquema dinámico ModifyEdge modifica las propiedades asociadas con un borde. Las antiguas propiedades asociadas con el borde son sustituidas por las nuevas propiedades. El conjunto de ofertas de servicio y el conjunto de nodos permanecen invariables, como asimismo las propiedades asociadas con todos los demás bordes.

La condición previa para esta acción es que debe existir el borde suministrado.

En Z, el esquema *ModifyEdge* representa el comportamiento correspondiente.

ModifyEdgeOK

Δ*TradingSystem*
edge? : *Node* ⇸ *Node*
new_edge_properties? : ℙ *Property*

---

edge? ⦏ *edges*
edge_properties′ = *edge_properties* ⩥ {*edge?* ➇ *new_edge_properties?*}
edges′ = *edges*
offers′ = *offers*
nodes′ = *nodes*
partition′ = *partition*

Si no se cumplen las condiciones previas de *ModifyEdgeOK*, el sistema de intermediación permanece invariable.

ModifyEdgeError

Ξ*TradingSystem*
edge? : *Node* ⇸ *Node*

---

edge? ⦎ *edges*

ModifyEdge prosperará o fracasará según las condiciones previas de los esquemas Z arriba indicados.

ModifyEdge ✧ *ModifyEdgeOK* ➘ *ModifyEdgeError*

7.5.7 Añadir nodo

El esquema dinámico AddNode describe el comportamiento de añadir un nodo al sistema de intermediación.

La condición previa de esta acción es que el nodo a añadir no exista todavía dentro del sistema de intermediación. Como el nuevo nodo no era un elemento de *nodes*, y por tanto no está en la gama de la función *partition*, no se hacen corresponder ofertas existentes en el nuevo nodo. Esto implica que los nuevos nodos no contienen ofertas.

En Z, el esquema *AddNode* representa el comportamiento correspondiente.

AddNodeOK

TradingSystem
new_node? : *Node*

---

new_node? ⊈ *nodes*
offers′ = *offers*
nodes′ = *nodes* ⇸ { *new_node?* }
edges′ = *edges*
edge_properties′ = *edge_properties*
partition′ = *partition*

Si no se cumplen las condiciones previas de *AddNodeOK*, el sistema de intermediación permanece invariable.

AddNodeError

TradingSystem
new_node? : *Node*

---

new_node? ∉ *nodes*

AddNode prosperará o fracasará según las condiciones previas de los esquemas Z arriba indicados.

AddNode ✧ *AddNodeOK* ➘ *AddNodeError*

7.5.8 Suprimir nodo

El esquema dinámico RemoveNode define el comportamiento de suprimir un nodo del sistema de intermediación. Las condiciones previas son que el nodo a suprimir esté presente en el sistema de intermediación, que no contenga ofertas de servicio y que ya no tenga bordes a otros nodos.

El comportamiento computacional puede combinar el esquema dinámico con otros esquemas dinámicos para suprimir ofertas de servicio y bordes.

En Z, el esquema *RemoveNode* representa el comportamiento correspondiente.

RemoveNodeOK

TradingSystem
old_node? : *Node*

---

old_node? ∈ *nodes*
old_node? ⊈ *ran partition*
old_node? ⊈ *dom edges* ⇸ *ran edges*
offers′ = *offers*
nodes′ = *nodes* \ {*old_node?* }
edge_properties′ = *edge_properties*
edges′ = *edges*
partition′ = *partition*

Si no se cumplen las condiciones previas de *RemoveNodeOK*, el sistema de intermediación permanece invariable.

RemoveNodeError

Ξ*TradingSystem*
old_node? : *Node*

---

old_node? $\notin$ *nodes*
old_node? $\notin$ *ran partition*
old_node? $\notin$ *dom edges* $\lhd$ *ran edges*

RemoveNode prosperará o fracasará según las condiciones previas de los esquemas Z arriba indicados.

RemoveNode $\oplus$ *RemoveNodeOK* $\gg$ *RemoveNodeError*

7.5.9 Buscar

El esquema dinámico Search es el comportamiento que busca el sistema de intermediación, restringido por algunos criterios de determinación de alcance, y retorna un conjunto de ofertas de servicio que satisfacen algunos criterios de concordancia. La acción no cambia el estado del sistema de intermediación. Su salida es un conjunto de ofertas de servicio. Su entrada es una petición de búsqueda.

Las ofertas que satisfacen el resultado de la búsqueda deben satisfacer las condiciones siguientes:

- deben tener el tipo de servicio correcto o un subtipo del mismo;
- deben estar contenidas en un nodo que sea aceptable tanto para los requisitos del importador y para las constricciones del sistema de intermediación, y que sea accesible desde el punto de partida;
- deben cumplir los criterios de concordancia del importador;
- deben satisfacer las constricciones de concordancia del sistema de intermediación.

En Z, el esquema *Search* representa el comportamiento correspondiente. La especificación formal de Search representa el efecto de los criterios de concordancia y los criterios de búsqueda como conjuntos de ofertas de servicio. El resultado de la búsqueda se obtiene por simple intersección de conjuntos. Todas las ofertas que se retornan deben ser accesibles desde el punto de partida original de la búsqueda. Esta propiedad se expresa formalmente declarando que todas las ofertas deben estar contenidas en un nodo que aparezca en el cierre transitivo de la relación *edges*.

SearchOK

Ξ*TradingSystem*
Ξ*TradingSystemConstraints*
SearchRequest?
starting_point? : *Node*
search_result! : $\mathbb{P}$*ServiceOffer*

---

starting_point? $\notin$ *nodes*
let e: EdgeCriteria $\bullet$ *e* $\subseteq$ *edges*
partition $\upharpoonright$ *search_result!* $\cap$ $\mathbb{S}$
  $\{x : Node \mid (starting\_point?, x) \notin e^{+}\}$
search_result! $\mathbb{S}$
  importer_matching? $\triangleleft$ *trader_matching* $\triangleleft$ *importer_scope?* $\triangleleft$ *trader_scope*

Si se hace un intento de buscar a partir de un nodo que no existe, no se retornan ofertas de servicio y el estado del sistema de intermediación no varía.

SearchError

Ξ*TradingSystem*
Ξ*TradingSystemConstraints*
SearchRequest?
starting_point? : *Node*
search_result! : $\mathbb{P}$ *ServiceOffer*

---

starting_point? $\notin$ *nodes*
search_result! = $\emptyset$

El comportamiento de *Search* se describe plenamente combinando los esquemas Z arriba indicados.

Search $\triangleq$ *SearchOK* $\gg$ *SearchError*

7.5.10 Seleccionar

El esquema dinámico Select es el comportamiento que ordena a un conjunto de ofertas de servicio con arreglo a algunos criterios de preferencia. Estos criterios son una combinación de la preferencia del importador y de las constricciones del sistema de intermediación. Estas constricciones incluyen aspectos de la política de arbitraje de la comunidad de intermediación. Las constricciones del sistema de intermediación modifican la preferencia expresada por el importador. La acción no cambia el estado del intermediario.

Selection

Search
selection! : **seq** *ServiceOffer*

---

selection! = trader_preference(*importer_preference*) (*search_result!*)

La especificación Z formal utiliza el esquema *Selection* para describir la aplicación de criterios de preferencia y constricciones al resultado de una búsqueda. *Select* suprime el resultado de búsqueda de la signatura de *Selection*.

Select $\triangleq$ *Selection* \ (*search_result!*)

NOTA – En el punto de vista computacional, los comportamientos *Search* y *Select* se combinan en la operación interrogar (query).

8 Especificación computacional de la función intermediación

El alcance de la especificación computacional se define en el RM-ODP parte 3: arquitectura (véase la Rec. UIT-T X.903 | ISO/CEI 10746-3).

En el punto de vista computacional, las interfaces de un intermedio con su entorno son visibles. La especificación computacional de esta Especificación define plantillas de interfaz para las interfaces computacionales (cliente y servidor) que pueden ser instanciadas por un objeto intermediario.

Para permitir a los implementadores constreñir las clases de conformidad de los intermediarios (véase la cláusula 9) con diferentes combinaciones de interfaces (y por tanto diferente funcionalidad), las interfaces de esta Especificación se agrupan en dos categorías:

- Interfaces funcionales, para la agrupación de operaciones basadas en la provisión de funcionalidad. Las cinco interfaces funcionales para un intercambio son: Lookup (consulta); Register (registro); Proxy (procuración); Link (enlace) y Admin (administración). Además, se especifican también dos interfaces auxiliares, Offer Iterator (iterador de ofertas) y Offer Id Iterator (iterador de Id de oferta).
- Interfaces abstractas, para la agrupación de atributos de lectura solamente basados en la sustentación requerida para una interfaz funcional. Esta Especificación especifica las interfaces abstractas: Support Attributes (atributos de sustentación); Import Attributes (atributos de importación); Link Attributes (atributos de enlace); y Trader Components (componentes de intermediario).

Las signaturas para las operaciones de las interfaces Admin y Link se definen para sustentar la portabilidad de los intermediarios. El comportamiento se especifica para las operaciones de gestión de enlace.

NOTA 1 – Las operaciones administrativas internas, tales como la creación y supresión de interfaces de intermediación se consideran operaciones pertenecientes a la función gestión de objeto genérica, y no son definidas por esta Especificación.

Un intermediario puede ser un cliente de varias funciones genéricas RM-ODP que son objeto de normalización futura. Para operaciones en las interfaces de dichas funciones servidoras, esta Especificación no especifica ni signaturas ni comportamiento.

El IDL ODP (véase la Rec. UIT-T X.920 | ISO/CEI 14750) se utiliza en esta especificación para expresar signaturas de interfaz de operación computacional. El empleo de esta notación no implica el uso de mecanismos soporte específico ni protocolos.

Además de esta especificación computacional, el anexo A incluye una especificación del IDL ODP de las signaturas operacionales en el módulo CosTrading y en el módulo CosTradingDynamic, y el anexo D incluye una especificación del IDL ODP de las signaturas operacionales del módulo CosTradingRepos.

NOTA 2 – Las interfaces operacionales de esta Especificación están técnicamente alineadas con el servicio de objetos de intermediación OMG.

8.1 Correspondencias de puntos de vista

8.1.1 Correspondencia con el punto de vista de la empresa

Las políticas expresadas en el punto de vista de la empresa se expresan en el punto de vista operacional como parámetros de operación o atributos de intermediario. Algunos de estos parámetros de operación se expresan como constricciones.

Cada constricción puede expresarse como una proposición de que:

- existe una propiedad denominada;
- una propiedad denominada tiene una relación especificada con un valor especificado;
- los valores de dos propiedades denominadas tienen una relación especificada.

Cualquier constricción puede ponerse a un valor por defecto de verdadero (true). Una constricción puede ser también una conjunción, una disyunción o una negación de otras constricciones.

Se identifican dos plantillas de acción para políticas de tratamiento:

- Acción consultar – Determina qué constricciones para regir el comportamiento se aplican a fin de respetar la política del intermediario;
- Acción arbitrar – Produce una constricción resultante para efectuar una operación dada combinando la política del cliente (expresada como una constricción o un parámetro de entrada) y la política del intermediario (representada por una constricción y algunos valores de atributo o de propiedad).

8.1.2 Correspondencia con el punto de vista de la información

La especificación de información define un gráfico direccionado en el que las divisiones de ofertas están situadas en uno de los nodos del gráfico. La única constricción discernible desde el punto de vista de la información es que, para un sentido dado, sólo se permite un borde entre dos nodos cualesquiera. Se permiten dos bordes, con sentidos opuestos, entre los mismos dos nodos.

La especificación computacional define objetos intermediarios, es decir, objetos que proporcionan un servicio de intermediación en una interfaz. Un objeto intermediario puede utilizar un servicio de intermediación de otro intermediario, es decir, está enlazado a ese otro intermediario.

La relación entre estas dos especificaciones de puntos de vista se prescribe en esta Especificación. Una división de información corresponderá a un objeto intermediario, y sólo uno. No se permite la posibilidad de muchas divisiones correspondientes a un único objeto intermediario. Cada interfaz de un servicio de intermediación tiene una sola división, desde la cual pueden conseguirse en una búsqueda de importación todas las ofertas asociadas con esa interfaz del servicio de intermediación.

Un borde en el gráfico del punto de vista de la información conecta divisiones que corresponden a diferentes objetos intermediarios, y debe por tanto él mismo corresponder a un enlace entre esos dos intermediarios. La división destinataria está asociada con la interfaz del servicio de intermediación a la que apunta ese enlace. Todas las divisiones correspondientes a un determinado intermediario deben estar asociadas con la interfaz del servicio de intermediación de ese intermediario o ser accesibles, en el gráfico de información, desde dicha división asociada.

8.2 Conceptos y tipos de datos

8.2.1 Exportador e importador

8.2.1.1 Exportador

Un exportador anuncia un servicio mediante un intermediario. Un exportador puede ser el proveedor del servicio o puede anunciar un servicio en nombre de otro.

8.2.1.2 Importador

Un importador utiliza un intermediario para buscar servicios que concuerden con algunos criterios. Un importador puede ser el cliente potencial de un servicio o puede importar un servicio en nombre de otro.

8.2.2 Tipos y valores arquitecturalmente neutros

8.2.2.1 TypeCode (código de tipo)

El IDL ODP no incluye un tipo de primitiva para representar las descripciones de tipo. Las plantillas IDL ODP especificadas en esta Especificación utilizan el término «TypeCode» para representar descripciones de tipo.

Cuando se implementa esta función ODP utilizando una infraestructura ODP determinada, el término «TypeCode» debe definirse utilizando un tipo apropiado par representar las descripciones de tipo en esa infraestructura ODP.

NOTA – Para la infraestructura CORBA, el término «TypeCode» debe definirse como «CORBA::TypeCode».

8.2.2.2 Valor de referencia de interfaz nulo (nil).

En esta Especificación el término «nulo» («nil») se utiliza para designar el valor de referencia de interfaz de una instancia de interfaz inexistente.

8.2.3 Tipos de servicio

8.2.3.1 Información de tipos de servicio

Asociado con cada servicio intermediario hay un tipo de servicio que representa la información necesaria para describir un servicio. Comprende:

- un tipo de interfaz que define la signatura computacional de la interfaz del servicio; y
- cero o más tipos de propiedad denominados. Éstos suelen representar aspectos de comportamiento, no funcionales y no computacionales que no son captados por la signatura computacional.

El tipo de propiedad define el tipo de valor de propiedad, si una propiedad es obligatoria, y si una propiedad es de lectura solamente. Es decir, asociada con un tipo de propiedad está la tripleta <nombre, tipo, modo>, donde los modos son:

```
enum PropertyMode {
    PROP_NORMAL, PROP_READONLY,
    PROP_MANDATORY, PROP_MANDATORY_READONLY
};
```

Un depositario de tipos de servicio se utiliza para contener la información de tipo.

```
typedef Object TypeRepository;
```

Un intermediario tiene un depositario de tipos de servicio asociado. Sin embargo, esta especificación no exige el uso de ninguna interfaz determinada de depositario de tipos de servicio. La interfaz de depositario de tipos puede o no formar parte del propio objeto de intermediación. En el anexo D se especifica una interfaz adecuada.

Cada tipo de servicio en un depositario es identificado por un único ServiceTypeName.

```
typedef Istring ServiceTypeName;
```

NOTA – La typedef Istring indica la intención de sustentar un juego de caracteres internacional en su utilización.

Un exportador especifica el tipo de servicio del servicio que está anunciando. Un importador especifica el tipo de servicio que está buscando.

Los tipos de servicio pueden relacionarse en una jerarquía que refleje la subtipificación de la interfaz (por ejemplo, por herencia) y la agregación de tipos de propiedad. Esta jerarquía proporciona la base para decidir si un servicio de un tipo puede ser sustituido por un servicio de otro tipo. Estas consideraciones se exponen con más detalle en el siguiente modelo de tipo de servicio.

8.2.3.2 Modelo de tipo de servicio

Esta subcláusula corresponde a la especificación del punto de vista de información de las reglas de subtipificación de tipos de servicio de 7.2.3. Estas reglas se expresan en estas subcláusulas utilizando términos del lenguaje computacional.

El modelo de tipo de servicio se ilustra mediante la siguiente BNF:

```
service <ServiceTypeName> [: <BaseServiceTypeName> [, <BaseServiceTypeName>]* ] {
    interface <InterfaceTypeName>;
    [[mandatory] [readonly] property <IDLType> <PropertyName>;]*
};
```

La palabra clave «servicio» introduce un nuevo ServiceTypeName. Como el tipo de servicio resulta visible a los usuarios finales y no simplemente a los programadores, es internacionalizable.

La lista de BaseServiceTypeNames enumera los tipos de servicio de los cuales se obtiene este tipo de servicio, que a su vez indica dónde este tipo de servicio puede sustituir a otro servicio.

La palabra clave «interfaz» introduce la InterfaceTypeName para este servicio. Está relacionado por equivalencia o por derivación con las InterfaceTypeNames en cada uno de los BaseServiceTypeNames.

La cláusula de propiedades es una lista de declaraciones de propiedades. Cada declaración de propiedad está marcada por la palabra clave «propiedad» y puede ir precedida por los atributos de modo «obligatorio» y/o «lectura solamente». Una declaración de propiedad es completada por un IDLType y un PropertyName. Un servicio debe sustentar todas las propiedades de cada uno de sus tipos de servicio de base, debe tener tipos de valor de propiedad idénticos, y debe no perder atributos de modo propiedad.

Los atributos de modo propiedad tienen las siguientes connotaciones:

- Obligatorio – Una instancia de este tipo de servicio debe proporcionar un valor apropiado de esta propiedad cuando exporte su oferta de servicio.
- Lectura solamente – Si una instancia de este tipo de servicio proporciona un valor adecuado de esta propiedad cuando exporte su oferta de servicio, el valor de esta propiedad puede no ser cambiado por una invocación posterior de la operación Register::modify().

El gráfico de preponderancia de propiedad se muestra en la figura 2.

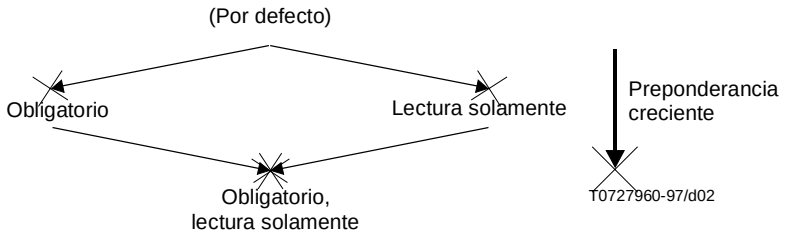


Figura 2 – Gráfico de preponderancia de modos de propiedad del servicio

En resumen, si una propiedad se define sin modificadores algunos, es opcional (es decir, no se requiere una oferta de ese tipo de servicio para proporcionar un valor de ese nombre de propiedad, pero si lo hace debe ser del tipo especificado en el tipo de servicio) y el valor de propiedad puede ser posteriormente modificado. El modificador «obligatorio» indica que debe proporcionarse un valor, pero que puede modificarse posteriormente. El modificador «lectura solamente» indica que la propiedad es opcional, pero que una vez dado un valor, no puede modificarse posteriormente. Especificar ambos modificadores indica que debe proporcionarse un valor y que no puede modificarse posteriormente.

De la discusión anterior, pueden establecerse las reglas de la conformidad de tipo de servicio; un tipo de servicio β es un subtipo del tipo de servicio α si y sólo si:

- el tipo de interfaz asociado con β es el mismo que, o un subtipo de, el tipo de interfaz asociado con α;
- todas las propiedades definidas en α se definen también en β;
- para todas las propiedades definidas en α y β, el modo de la propiedad en β debe ser el mismo que, o más preponderante que, el modo de la propiedad en α;
- todas las propiedades definidas en β que también se definen en α tendrán, en β, el mismo tipo de valor de propiedad que en α, o un subtipo del tipo de valor de propiedad en α.

8.2.4 Propiedades

Las propiedades son pares <nombre, valor>. Un exportador declara valores de propiedades del servicio que está anunciando. Un importador puede obtener estos valores acerca de un servicio y constreñir su búsqueda de ofertas apropiadas basada en los valores de propiedad asociados con tales ofertas.

```
typedef Istring PropertyName;
typedef sequence<PropertyName> PropertyNameSeq;
typedef any PropertyValue;
struct Property {
        PropertyName name;
        PropertyValue value;
};
typedef sequence<Property> PropertySeq;

enum HowManyProps { none, some, all };
union SpecifiedProps switch ( HowManyProps ) {
        case some: PropertyNameSeq prop_names;
};
```

8.2.5 Ofertas de servicio

Una oferta de servicio es una información declarada por un exportador acerca del servicio que está anunciando. Contiene:

- el nombre del tipo de servicio;
- una referencia a la interfaz que proporciona el servicio; y
- cero o más valores de propiedades del servicio.

Un exportador debe especificar un valor para todas las propiedades obligatorias especificadas en el tipo de servicio asociado. Un exportador puede también nominar valores de propiedades denominadas que no se especifican en el tipo de servicio. En dicho caso, el intermediario no está obligado a hacer la comprobación del tipo de propiedad.

```
struct Offer {
        Object reference;
        PropertySeq properties;
};
typedef sequence<Offer> OfferSeq;

struct OfferInfo {
        Object reference;
        ServiceTypeName type;
        PropertySeq properties;
};
```

8.2.5.1 Propiedades modificables

El valor de una propiedad en una oferta de servicio puede opcionalmente modificarse, si:

- el modo de propiedad no es lectura solamente, si es opcional u obligatorio; y
- el intermediario sustenta la funcionalidad modificar propiedad.

Dichos valores de propiedad pueden actualizarse mediante operaciones modificar explícitas en el intermediario. Un exportador puede controlar una oferta de servicio para que sea no modificable exportando servicios con los tipos de servicio que tienen propiedades de lectura solamente. La operación modificar retornará una excepción NotImplemented si un intermediario no sustenta la funcionalidad modificar propiedad. Un importador puede también especificar si un intermediario debe o no considerar ofertas con propiedades modificables durante la concordación.

8.2.5.2 Propiedades dinámicas

Una oferta de servicio puede opcionalmente contener propiedades dinámicas. El valor de una propiedad dinámica no es mantenido dentro de un intermediario, y se obtiene, por demanda, de la interfaz de un evaluador de propiedades dinámicas designado por el exportador del servicio. Es decir, se requiere un nivel de indirección para obtener el valor de una propiedad dinámica. La estructura de un valor de propiedad dinámica es:

```
exception DPEvalFailure {
        CosTrading::PropertyName name;
        TypeCode returned_type;
        any extra_info;
};
```

```
interface DynamicPropEval {

    any evalDP (
        in CosTrading::PropertyName name,
        in TypeCode returned_type,
        in any extra_info
    ) raises (
        DPEvalFailure
    );
};

struct DynamicProp {
    DynamicPropEval eval_if;
    TypeCode returned_type;
    any extra_info;
};
```

Contiene la interfaz al evaluador de propiedades dinámicas, el tipo de datos de la propiedad dinámica retornada, y cualquier información extraordinaria dependiente de la implementación. El intermediario reconoce esta estructura y, cuando se requiere el valor de la propiedad, invoca la operación evalDP a la interfaz DynamicPropEval apropiada. La interfaz del evaluador de propiedades dinámicas tiene sólo una operación, cuya signatura se define en esta Especificación para su portabilidad, pero no se especifica su comportamiento. Las únicas restricciones impuestas son que la propiedad no debe ser de lectura solamente y que el intermediario debe sustentar la funcionalidad propiedad dinámica.

El uso de dichas propiedades tiene repercusiones en el rendimiento de un intermediario. Un importador puede especificar si un intermediario debe considerar o no ofertas con propiedades dinámicas durante la concordación.

8.2.6 Identificador de oferta

Se retorna un identificador de oferta a un exportador cuando se anuncia en un intermediario una oferta de servicio. Identifica la oferta de servicio exportada y es citado por el exportador cuando retira y modifica la oferta (cuando es sustentada). Sólo tiene significado para el intermediario en el que se ha registrado la oferta de servicio.

```
typedef string OfferId;
typedef sequence<OfferId> OfferIdSeq;
```

8.2.7 Selección de oferta

El espacio total de ofertas de servicio para una selección de oferta es potencialmente muy grande, incluyendo ofertas de todos los intermediarios enlazados. Lógicamente, el intermediario utiliza políticas para identificar el conjunto S1 de ofertas de servicio a examinar. El tipo de servicio y la constricción se aplican entonces a S1 para producir el conjunto S2 que satisface el tipo de servicio y la constricción. Esto se ordena entonces utilizando preferencias antes de retornar las ofertas al importador.

8.2.7.1 Lenguaje de constricciones normalizado

Los importadores seleccionan el conjunto de ofertas de servicio por las que tienen interés utilizando el tipo de servicio y una constricción. La constricción es una expresión bien formada conforme a un lenguaje de constricciones.

Esta Recomendación | Norma Internacional define el lenguaje normalizado obligatorio necesario para el interfuncionamiento entre intermediarios. El anexo B define la sintaxis y el poder de expresión del lenguaje de constricciones. Este lenguaje de constricciones se utiliza para escribir expresiones de constricción normalizadas.

```
typedef Istring Constraint;
```

Sus principales características son:

Tipos de valor de propiedad	la manipulación se limita a los tipos int, float, fixed, boolean, Istring/string, Ichar/char y secuencias de los mismos. Los tipos basados en los caracteres se ordenan utilizando la secuencia de confrontación en efecto para el juego de caracteres considerado. Los tipos fuera de esta gama sólo pueden ser objeto del operador «exists».
Literales	en la constricción son dinámicamente coercidos en la medida necesaria para las propiedades con las que están trabajando. Los literales pueden contener Istring.
Operadores	son de comparación, booleanos conectivos, «in_set», de subcadena, operadores aritméticos, de existencia de propiedad.

Si se utiliza un lenguaje de constricciones propietario (fuera del alcance de esta Especificación), el nombre y la versión del lenguaje de constricciones utilizado se coloca entonces entre << >> al comienzo de la expresión de constricción. El resto de la cadena no es interpretado por un intermediario que no sustenta el lenguaje de constricciones propietario citado.

8.2.7.2 Preferencias

Las preferencias son lógicamente aplicadas al conjunto de ofertas concordadas por aplicación del tipo de servicio, expresión de constricción, y diversas políticas. La aplicación de las preferencias puede considerarse como la determinación del orden de retorno de las ofertas concordadas al importador.

typedef Istring Preference;

La cadena de preferencia puede considerarse compuesta por dos porciones: la primera puede ser cualquiera de las siguientes (se advierte que estas palabras clave son sensibles al caso considerado):

max min with random first

La interpretación de la segunda porción es dependiente de la primera; puede estar vacía. A continuación se describen las preferencias:

Preferencia	Descripción
expresión máx (max expression)	La expresión es numérica. Las ofertas concordadas se retornan en orden descendente de la expresión.
expresión mín (min expression)	La expresión es numérica. Las ofertas concordadas se retornan en orden ascendente de la expresión.
expresión con (with expression)	La expresión es una expresión de constricción. Las ofertas concordadas se ordenan de manera que las que son verdaderas (TRUE) preceden a las que son falsas (FALSE).
aleatoria (random)	El orden de retorno de las ofertas concordadas sigue el siguiente algoritmo: seleccionar una oferta aleatoriamente a partir del conjunto de ofertas concordadas, seleccionar otra oferta aleatoriamente a partir del conjunto restante de ofertas concordadas, ..., seleccionar la única oferta restante.
primera (first)	El orden de las ofertas concordadas retornadas es el orden en el que se descubren las ofertas.

Si no se especifica preferencia, se aplica entonces la preferencia primero (first). No se permiten combinaciones de las preferencias.

La expresión asociada con max, min y with puede referirse a propiedades asociadas con las ofertas concordantes. Cuando se aplica una expresión de preferencia al conjunto de ofertas que concuerdan el tipo de servicio y la expresión de constricción, el conjunto de ofertas se divide en dos:

a) un grupo de ofertas en las que podría evaluarse la expresión de preferencia (ordenada con arreglo a min, max, with); y

b) un grupo de ofertas en las que no podría evaluarse la expresión de preferencia (por ejemplo, la expresión de preferencia se refiere a un nombre de propiedad que es opcional para ese tipo de servicio).

Las ofertas son retornadas al importador en el orden del primer grupo de su orden de preferencia, seguidas por las del segundo grupo.

Si se utiliza un lenguaje de preferencia propietario (fuera del alcance de esta Especificación), el nombre y la versión del lenguaje de preferencia se coloca entonces entre << >> al comienzo de la preferencia. El resto de la cadena no es interpretado por un intermediario que no sustenta el lenguaje propietario citado.

8.2.7.3 Enlaces

Los enlaces representan trayectos para la propagación de interrogaciones desde un intermediario fuente a un intermediario destinatario. Cada enlace corresponde a un borde en un gráfico de intermediación, en el que los vértices son intermediarios. Un enlace describe el conocimiento que un intermediario tiene de otro servicio de intermediación que utiliza. También incluye información sobre cuándo propagar o remitir una operación al intermediario destinatario. Un enlace tiene asociada la siguiente información:

– una interfaz Lookup proporcionada por el intermediario destinatario, que sustenta la operación interrogar;

– una interfaz Register proporcionada por el intermediario destinatario, que sustenta la operación resolver;

– el comportamiento de seguimiento por defecto del enlace, que puede utilizarse y se transmite cuando un importador no especifica una política link_follow_rule;

– el comportamiento de seguimiento limitador del enlace, que contraordena una link_follow_rule del importador si la petición de éste excede el límite impuesto por el enlace.

El IDL OMG para FollowOption se define como:

```
enum FollowOption {
    local_only,
    if_no_local,
    always
};
```

donde:

– «local_only» indica que el enlace nunca es seguido a menos que se nombre explícitamente una operación;

– «if_no_local» indica que el enlace es seguido sólo si no hay ofertas locales que satisfagan la interrogación; y

– «always» indica que el enlace es siempre seguido salvo cuando es contraordenado por alguna política.

Estos valores se ordenan como sigue:

local_only < if_no_local < always

El IDL OMG para LinkInfo se define como:

```
struct LinkInfo {
    Lookup target;
    Register target_reg;
    FollowOption def_pass_on_follow_rule ;
    FollowOption limiting_follow_rule;
};
```

La información anterior se fija para cada enlace cuando éste es creado. Se da un nombre al enlace cuando éste es creado. El nombre identifica unívocamente un enlace en un intermediario.

```
typedef Istring LinkName;
typedef sequence<LinkName> LinkNameSeq;
```

Un enlace es unidireccional. Sólo el intermediario fuente conoce directamente la existencia de un enlace; es el intermediario fuente el que sustenta la interfaz Link.

Puede mantenerse información adicional con un enlace para describir las características del servicio de intermediación deseado percibido por el intermediario fuente.

8.2.7.4 Políticas

Las políticas proporcionan información que influye en el comportamiento del intermediario en el tiempo de funcionamiento. Las políticas se representan como pares de valores de nombre.

```
typedef string PolicyName; // policy names restricted to Latin1
typedef sequence<PolicyName> PolicyNameSeq;
typedef any PolicyValue;
struct Policy {
    PolicyName name;
    PolicyValue value;
};
typedef sequence<Policy> PolicySeq;
```

Algunas políticas no pueden ser contraordenadas mientras se apliquen otras políticas en ausencia de otra información, y pueden ser contraordenadas. Las políticas pueden agruparse en dos categorías:

– las que determinan el alcance de una búsqueda;

– las que determinan la funcionalidad aplicada a una operación.

Diferentes políticas están asociadas con diferentes roles en la prestación de la función de intermediación. Estos roles, que se utilizan en la columna «Dónde» de los siguientes cuadros, son:

T = Trader

L = Link

I = Importer

Estas políticas se tratarán en 8.2.7.5 a 8.2.7.9.

8.2.7.4.1 Políticas de determinación de alcance normalizadas

A continuación se enumeran las políticas de determinación de alcance normalizadas:

Nombre	Dónde	Tipo de IDL	Descripción
def_search_card	T	unsigned long	Límite superior por defecto de las ofertas a buscar; utilizado si no se especifica search_card
max_search_card	T	unsigned long	Límite superior máximo de las ofertas a buscar
search_card	I	unsigned long	Límite superior designado de las ofertas a buscar; será contraordenado por max_search_card
def_match_card	T	unsigned long	Límite superior por defecto de las ofertas concordadas a ordenar; utilizado si no se especifica match_card
max_match_card	T	unsigned long	Límite superior máximo de las ofertas concordadas a ordenar
match_card	I	unsigned long	Límite superior designado de las ofertas a ordenar; será contraordenado por max_match_card
def_return_card	T	unsigned long	Límite superior por defecto de las ofertas ordenadas a retornar; utilizado si no se especifica return_card
max_return_card	T	unsigned long	Límite superior máximo de las ofertas ordenadas a retornar
return_card	I	unsigned long	Límite superior designado de las ofertas ordenadas a retornar; será contraordenado por max_return_card
def_hop_count	T	unsigned long	Límite superior por defecto de profundidad de los enlaces a atravesar si no se especifica hop_count
max_hop_count	T	unsigned long	Límite superior máximo de profundidad de los enlaces a atravesar
hop_count	I	unsigned long	Límite superior designado de profundidad de los enlaces a atravesar; será contraordenado por la max_hop_count del intermediario
def_pass_on_follow_rule	L	FollowOption	Comportamiento de seguimiento por defecto del enlace a transmitir por un determinado enlace si un importador no especifica su link_follow_rule. No debe exceder la limiting_follow_rule
limiting_follow_rule	L	FollowOption	Comportamiento de seguimiento limitador del enlace de un determinado enlace
max_link_follow_policy	T	FollowOption	Límite superior del valor de una regla de seguimiento limitado del enlace en el momento de la creación o la modificación de un enlace
def_follow_policy	T	FollowOption	Comportamiento de seguimiento por defecto del enlace para un intermediario determinado
max_follow_policy	T	FollowOption	Política de seguimiento limitadora del enlace para todos los enlaces del intermediario – contraordena las políticas del enlace y del importador
link_follow_rule	I	FollowOption	Comportamiento de seguimiento designado del enlace; será contraordenado por la max_follow_policy del intermediario y por la limiting_follow_rule del enlace
starting_trader	I	TraderName	Un importador determina el alcance de su búsqueda designando que la operación interrogar empieza en un intermediario distante; un intermediario está obligado a enviar la petición hasta un enlace aun si el comportamiento del enlace es local_only
request_id	I	OctetSeq	Un identificador de una operación interrogar iniciada por un intermediario fuente que actúa como un importador sobre un enlace; un intermediario no está obligado a generar un id, pero está obligado a transmitirlo a un enlace
exact_type_match	I	boolean	Si es TRUE, sólo se consideran ofertas de exactamente el tipo de servicio especificado por el importador; si es FALSE (o sin especificar), se consideran ofertas de cualquier tipo de servicio que se conformen al tipo de servicio del importador

Los tipos de IDL ODP para TraderName y OctetSeq son:

```
typedef LinkNameSeq TraderName;
typedef sequence<octet> OctetSeq;
```

Los resultados recibidos por un importador son afectados por las políticas de determinación de alcance. La hop_count y las políticas de seguimiento fijan el alcance de los intermediarios que visitan. N1 es el espacio de oferta de servicio total de esos intermediarios. Las ofertas que tienen un tipo de servicio conforme son reunidas en el conjunto N2; el tamaño real de N2 puede ser restringido aún más por las políticas de cardinalidad de búsqueda. Las constricciones se aplican a N2 para producir un conjunto N3 de ofertas que satisfagan el tipo de servicio y las constricciones; N3 puede ser restringido aún más por las políticas de cardinalidad de concordación. El conjunto N3 se ordena entonces utilizando las preferencias para producir el conjunto N4. El conjunto final de ofertas retornado al importador, N5, puede todavía ser reducido por las políticas de cardinalidad retornadas.

Se ilustra esto con el diagrama presentado en la figura 3, donde |N1| >= |N2| >= |N3| = |N4| >= |N5|.

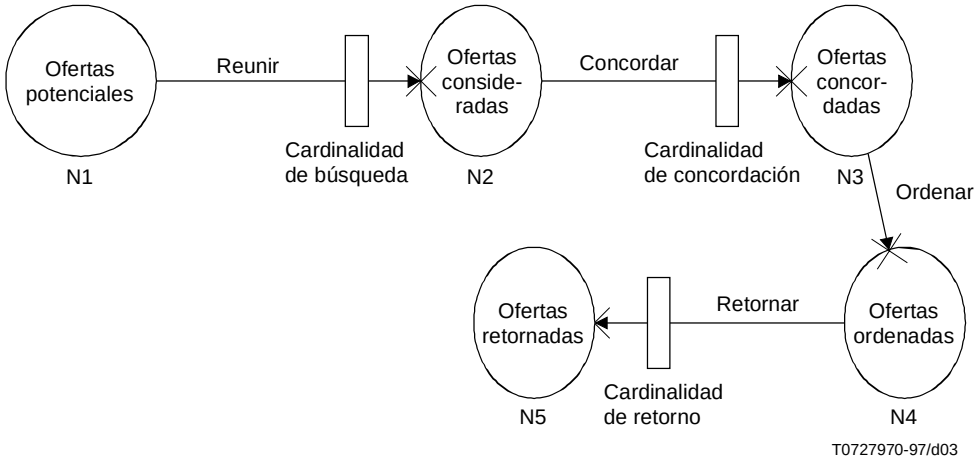


Figura 3 – Flujograma de los pasos de interrogación del intermediario y de la aplicación de constricciones de cardinalidad

8.2.7.4.2 Políticas sustentadas de capacidades normalizadas

Hay tres capacidades: oferta por procuración, propiedades dinámicas y modificar ofertas, que un intermediario puede o no desear sustentar. Si un intermediario no sustenta una capacidad, un importador no puede entonces contraordenar esa capacidad con su propio parámetro política (policy). Sin embargo, si un intermediario sustenta una capacidad y un importador no desea considerar ofertas que exigen dicha funcionalidad, el intermediario debe entonces respetar el deseo del importador.

A continuación se enumeran las políticas normalizadas relacionadas con la funcionalidad sustentada:

Nombre	Dónde	Tipo de IDL	Descripción
supports_modifiable_properties	T	boolean	Si el intermediario sustenta la modificación de propiedad
use_modifiable_properties	I	boolean	Si se consideran ofertas con propiedades modificables en la búsqueda
supports_dynamic_properties	T	boolean	Si el intermediario sustenta propiedades dinámicas
use_dynamic_properties	I	boolean	Si se consideran ofertas con propiedades dinámicas en la búsqueda
supports_proxy_offers	T	boolean	Si el intermediario sustenta ofertas por procuración
use_ proxy_offers	I	boolean	Si se consideran ofertas por procuración en la búsqueda

8.2.7.5 Políticas de intermediario

Las políticas pueden ser fijadas para un intermediario en su conjunto. Las políticas de intermediario se definen como atributos del objeto intermediario. Son inicialmente especificadas cuando se crea el intermediario, y pueden ser modificadas/interrogadas vía la interfaz Admin. Un importador puede interrogar a estas políticas de intermediario vía su interfaz Lookup. Un exportador puede interrogar a políticas sustentadas por una funcionalidad de intermediario vía su interfaz Register.

8.2.7.6 Comportamiento de seguimiento del enlace

Cada enlace de un intermediario tiene sus propias políticas de comportamiento de seguimiento. Un intermediario tiene una política de seguimiento limitadora, max_follow_policy, que contraordena todos los enlaces de ese intermediario para cualquier interrogación determinada. Se especifican políticas de comportamiento de seguimiento para cada enlace cuando se crea un enlace. Estas políticas, def_pass_on_follow_rule y limiting_follow_rule, pueden ser interrogadas/modificadas vía la interfaz Link. Los valores que pueden tener están limitados por otra política de intermediario, max_link_follow_policy, en el momento de la creación o modificación. Un importador puede especificar una link_follow_rule en una interrogación. En ausencia de una link_follow_rule del importador, se utiliza la def_follow_policy del intermediario.

Después de buscar sus ofertas locales en respuesta a una interrogación, un intermediario debe decidir si propaga la interrogación a lo largo de sus enlaces, y de ser así, qué valor de la link_follow_rule transmitir en el argumento políticas.

El IDL ODP para FollowOption se especifica en 8.2.7.3.

La política de seguimiento para un determinado enlace es, por tanto:

```
if the importer specified a link_follow_rule policy
    min(trader.max_follow_policy, link.limiting_follow_rule,
        query.link_follow_rule)

else
    min(trader.max_follow_policy, link.limiting_follow_rule,
        trader.def_follow_policy)
```

es decir, si este valor es «if_no_local» y no había ofertas locales que concuerden la interrogación, se efectúa la interrogación anidada; si este valor es «always» se efectúa la interrogación anidada.

Si la interrogación anidada es permitida por la regla anterior, la lógica siguiente determina el valor de la política «link_follow_rule» que ha de transmitirse al intermediario enlazado.

```
if the importer specified a link_follow_rule policy
    pass on min(query.link_follow_rule, link.limiting_follow_rule,
                trader.max_follow_policy)

else
    pass on min(link.def_pass_on_follow_rule , trader.max_follow_policy)
```

8.2.7.7 Políticas de importador

Un importador puede especificar cero o más políticas de importador en su parámetro política. Si no se especifica la política de importador, el intermediario utiliza su política por defecto. Si una política de importador excede los valores de política limitadora fijados por el intermediario, el intermediario, contraordena las expectativas del importador con su valor de política limitador.

Si se utiliza un parámetro de política starting_trader las implementaciones de intermediario colocarán este parámetro política como el primer elemento de la secuencia cuando remitan la petición de interrogación a los intermediarios enlazados.

8.2.7.8 Políticas de exportador

No se especifican en esta Especificación políticas de exportador.

8.2.7.9 Políticas de creación de enlace

En el momento en que se crea un enlace, se especifican las reglas de seguimiento por defecto y limitadoras asociadas con el enlace. Estas reglas pueden constreñirse mediante la max_link_follow_policy del intermediario.

El intermediario comprueba primero si la regla por defecto es menor o igual que la regla limitadora. Si no es así, se produce entonces una excepción. Compara entonces la regla limitadora con la max_link_follow_policy del intermediario, produciéndose una vez más una excepción si la regla limitadora es mayor que la max_link_follow_policy del intermediario.

8.2.8 Mecanismos de interfuncionamiento

8.2.8.1 Control transversal del enlace

La naturaleza flexible de un enlace de intermediario permite producir gráficos direccionados arbitrarios de intermediarios. Éstos pueden introducir dos tipos de problema:

- Un solo intermediario puede ser visitado más de una vez durante una búsqueda debido a que aparece en más de un trayecto (es decir, conjunto distinto de bordes conectados) procedente de un intermediario.
- Pueden producirse bucles – El ejemplo más trivial es cuando dos espacios de intermediario previamente disjuntos deciden unirse mediante el intercambio de enlaces. Esto puede dar lugar a que el primer intermediario propague una interrogación al segundo y a continuación la reciba de vuelta inmediatamente vía el enlace inverso.

Para asegurar que una búsqueda no entra en un bucle infinito, se utiliza una hop_count para limitar la profundidad de los enlaces en propagar una búsqueda. La hop_count se decrementa en uno antes de propagar una interrogación a otros intermediarios. La propagación de la búsqueda termina en el intermediario cuando la hop_count llega a cero.

Para evitar la revisita improductiva de un determinado intermediario mientras se efectúa una interrogación, un intermediario fuente puede generar un RequestId para cada operación interrogar que inicie para la propagación a un intermediario destinatario. El atributo intermediario de request_id_stem se utiliza para formar RequestId.

```
typedef sequence<octet> OctetSeq;
attribute OctetSeq request_id_stem;
```

Un intermediario puede desear recordar el RequestId de todas las operaciones interrogar interfuncionantes recientes que se le ha pedido que realice. Cuando se recibe una operación interrogar, dicho intermediario comprueba este historial y sólo procesa la interrogación si es la primera aparición de la operación.

Para que esto funcione, el administrador de un conjunto de intermediarios federados debe haber inicializado los posibles request_id_stems a valores no superpuestos.

El RequestId se transmite en un parámetro política del importador en la operación interrogar al intermediario destinatario. Si el intermediario destinatario no sustenta el uso de la política RequestId, el intermediario destinatario no necesita procesar el RequestId, pero debe transmitir el RequestId al siguiente intermediario enlazado si continúa propagándose la búsqueda.

8.2.8.2 Ejemplo de interrogación federada

Para propagar una petición de interrogación en un gráfico de intermediación, cada intermediario fuente actúa como cliente en la interfaz Lookup del intermediario destinatario y transmite la operación de interrogación de su cliente a su intermediario destinatario.

La figura 4 utiliza un ejemplo de búsqueda secuencial para ilustrar la modificación del parámetro cuenta de saltos a medida que una petición de interrogación se transmite a través de un conjunto de intermediarios enlazados en un gráfico de intermediación. Suponemos que las políticas de seguimiento del enlace en los intermediarios darán lugar a un comportamiento de seguimiento «always»:

a) Una petición de interrogación es invocada en la interfaz de intermediación de T1 con una política de cuenta de saltos del importador expresada como hop_count = 4. La política de determinación de alcance del intermediario para T1 incluye max_hop_count = 5. La hop_count resultante aplicada para la búsqueda (después de la acción de arbitraje que combina la política del intermediario y la política del importador) es hop_count = 4.

b) Suponemos que no se encuentra concordancia en T1 y que la política de seguimiento resultante es always (siempre). Es decir, T1 ha de transmitir la petición a T3. Se utiliza una política hop_count de importado modificada de hop_count = 3. La política de determinación de alcance del intermediario local para T3 incluye max_ hop_count = 1 y la generación de T3_Request_id para evitar la repetición o búsquedas cíclicas de los mismos intermediarios. La política de determinación de alcance resultante aplicada para la búsqueda en T3 es hop_count = 1 y se almacena el T3_Request_id.

c) Suponiendo que no se encuentra concordancia en T3 y que la política de seguimiento resultante es always, el parámetro de terminación de alcance modificado para la petición de interrogación en T4 es: hop_count = 0 y request_id = T3_Request_id.

d) Suponiendo que no se encuentra concordancia en T4. Aun cuando la max_ hop_count = 4 para T4, la búsqueda no se sigue propagando. Se devolverá un resultado de fracaso de la búsqueda a T3, a T1, y finalmente al usuario en T1.

Naturalmente, si una petición de interrogación concluye con éxito en cualquiera de los intermediarios del trayecto de búsqueda enlazado, se devolverá entonces la lista de ofertas de servicio concordadas al usuario original; si la petición de interrogación se propaga a través del gráfico de intermediación restante es algo que depende de las políticas de seguimiento del enlace; en este caso, en el que se supone que es always, la interrogación visitará todavía a todos los intermediarios compatibles con la política de cuenta de saltos.

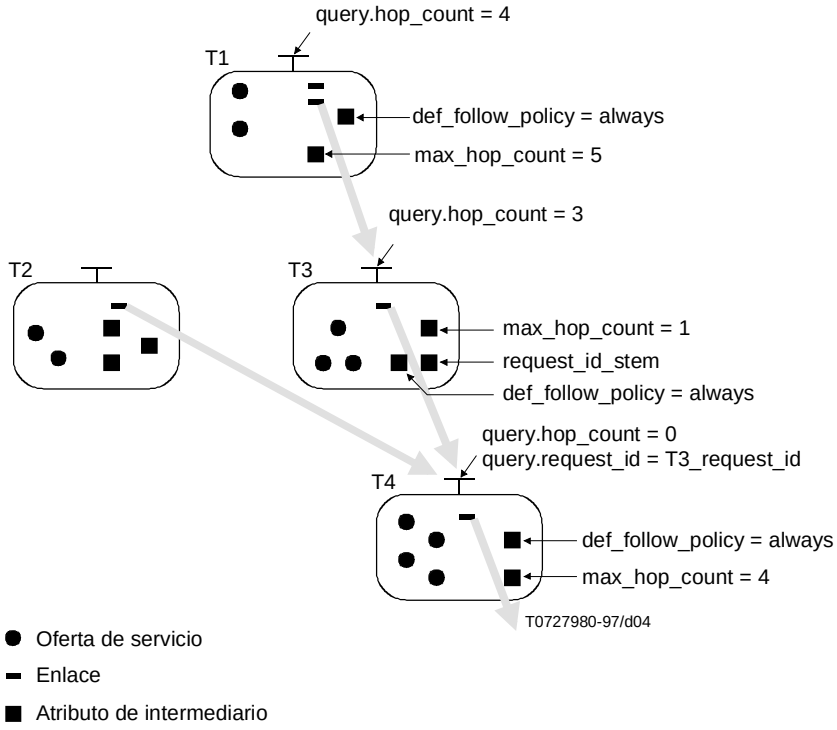


Figura 4 – Flujo de una interrogación a través de un gráfico de intermediario

8.2.8.3 Ofertas por procuración

Una oferta por procuración es un cruce entre una oferta de servicio y una forma de enlace restringido. Incluye el tipo de servicio y propiedades de una oferta de servicio, y como tal, se concuerda de la misma manera. Sin embargo, si la oferta por procuración concuerda con los requisitos del importador, en lugar de retornar detalles de la oferta, la petición de interrogación (modificada) se remite a la interfaz Lookup asociada con la oferta por procuración.

```
typedef Istring ConstraintRecipe;

struct ProxyInfo {
      ServiceTypeName type;
      Lookup target;
      PropertySeq properties;
      boolean if_match_all;
      ConstraintRecipe recipe;
      PolicySeq policies_to_pass_on;
};
```

Si la interrogación de un importador da lugar a una concordancia con una oferta por procuración, el intermediario que mantiene la oferta por procuración efectúa una interrogación anidada en el intermediario ocultándose detrás de la oferta por procuración con los siguientes parámetros:

- El parámetro tipo original se transmite sin variación.
- Se construye un nuevo parámetro constricción a continuación del ConstraintRecipe asociado con la oferta por procuración.
- El parámetro preferencia original se transmite sin variación.
- Un nuevo parámetro políticas se construye agregando las policies_to_pass_on asociadas con la oferta por procuración al parámetro políticas original.
- El parámetro desired_props original se transmite sin variación.
- El intermediario llamante suministra un valor de how_many que tiene sentido dada sus constricciones de recursos.

Las ofertas por procuración son una forma conveniente de embalar el encapsulado de un sistema legado de «objetos» para convertirlo en el sistema de intermediación. Permite a los clientes consultar estos «objetos» concordando la oferta por procuración; la llamada anidada al intermediario por procuración, junto con la expresión de constricción reescrita y las políticas adicionales agregadas al parámetro política original, permite la creación dinámica de una instancia de servicio que encapsula al objeto legado. Otro posible uso de procuraciones es para una fábrica de servicios que hay que anunciar como una oferta por procuración; la llamada anidada a la fábrica causa una nueva instancia del servicio concreto a fabricar.

Una interrogación puede tener concordada una oferta por procuración debido a un determinado valor de una propiedad asociada con la oferta por procuración. Es obligatorio que cualquier oferta retornada por el intermediario por procuración de resultas de la interrogación anidada tenga el mismo valor que la propiedad a fin de no violar las expectativas del cliente en relación con la constricción.

Un intermediario no tiene que sustentar la funcionalidad oferta por procuración. Sin embargo, si un intermediario sustenta dicha funcionalidad, debe proporcionar la interfaz Proxy (procuración) para la exportación, retirada y descripción de ofertas por procuración. Un importador puede especificar y un intermediario debe considerar ofertas por procuración durante la concordación.

8.2.9 Atributos de intermediario

Cada intermediario tiene sus propias características, políticas para las funcionalidades sustentadas, y políticas para la determinación del alcance de la búsqueda. Estas características y políticas se definen como atributos para el intermediario. Estos atributos son:

Nombre	Tipo de IDL ODP	Descripción
def_search_card	unsigned long	Límite superior por defecto de las ofertas a buscar para una operación interrogación
max_search_card	unsigned long	Límite superior mínimo de las ofertas a buscar para una operación interrogación.
def_match_card	unsigned long	Límite superior por defecto de las ofertas concordadas a ordenar al aplicar un criterio de preferencia
max_match_card	unsigned long	Límite superior mínimo de las ofertas concordadas a ordenar al aplicar un criterio de preferencia
def_return_card	unsigned long	Límite superior por defecto de las ofertas ordenadas a retornar a un importador
max_return_card	unsigned long	Límite superior mínimo de las ofertas ordenadas a retornar a un importador
def_hop_count	unsigned long	Límite superior por defecto de la profundidad de los enlaces a atravesar
max_hop_count	unsigned long	Límite superior mínimo de profundidad de los enlaces a atravesar
def_follow_policy	FollowOption	Comportamiento de seguimiento por defecto del enlace para un determinado intermediario
max_follow_policy	FollowOption	Política de seguimiento limitadora del enlace para todos los enlaces del intermediario – contraordena las políticas del enlace y del importador
max_link_follow_policy	FollowOption	Política de seguimiento más permisiva autorizada cuando se crean nuevos enlaces
supports_modifiable_properties	boolean	Si el intermediario sustenta la modificación de propiedad.
supports_dynamic_properties	boolean	Si el intermediario sustenta propiedades dinámicas
supports_proxy_offers	boolean	Si el intermediario sustenta ofertas por procuración
max_list	unsigned long	El atributo max_list determina el límite superior de cualquier lista retornada por el intermediario, a saber: el parámetro ofertas retornadas en la interrogación y la operación next-n en offer-iterator y offer-id-iterator
type_repos	Repository	Interfaz al depositario de servicios del intermediario
request_id_stem	OctetSeq	Identificación del intermediario, a utilizar como raíz para la producción de un id para una petición de interrogación de un intermediario a otro

Estos atributos son inicialmente especificados cuando se crea un intermediario, y pueden ser modificados/interrogados vía la interfaz Admin.

8.3 Excepciones

Esta especificación define las excepciones producidas por operaciones. Las excepciones son parametrizadas para indicar el origen del error. Los segmentos IDL ODP a continuación designan algunas de las typedef definidas más arriba en la cláusula 2.

Cuando surgen múltiples condiciones de excepción, sólo se produce una excepción. La elección de la excepción producida es dependiente de la implementación.

8.3.1 Para el módulo CosTrading

8.3.1.1 Excepciones utilizadas en más de una interfaz

```
exception UnknownMaxLeft {};

exception NotImplemented {};

exception IllegalServiceType {
        ServiceTypeName type;
};

exception UnknownServiceType {
        ServiceTypeName type;
};

exception IllegalPropertyName {
        PropertyName name;
};

exception DuplicatePropertyName {
        PropertyName name;
};

exception PropertyTypeMismatch {
        ServiceTypeName type;
        Property prop;
};

exception MissingMandatoryProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception IllegalConstraint {
        Constraint constr;
};

exception InvalidLookupRef {
        Lookup target;
};

exception IllegalOfferId {
        OfferId id;
};

exception UnknownOfferId {
        OfferId id;
};

exception ReadonlyDynamicProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception DuplicatePolicyName {
        PolicyName name;
};
```

8.3.1.2 Excepciones adicionales para la interfaz Lookup

```
exception IllegalPreference {
        Preference pref;
};
```

```
exception IllegalPolicyName {
        PolicyName name;
};

exception PolicyTypeMismatch {
        Policy the_policy;
};

exception InvalidPolicyValue {
        Policy the_policy;
};
```

8.3.1.3 Excepciones adicionales para la interfaz Register

```
exception InvalidObjectRef {
        Object ref;
};

exception UnknownPropertyName {
        PropertyName name;
};

exception InterfaceTypeMismatch {
        ServiceTypeName type;
        Object reference;
};

exception ProxyOfferId {
        OfferId id;
};

exception MandatoryProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception ReadonlyProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception NoMatchingOffers {
        Constraint constr;
};

exception IllegalTraderName {
        TraderName name;
};

exception UnknownTraderName {
        TraderName name;
};

exception RegisterNotSupported {
        TraderName name;
};
```

8.3.1.4 Excepciones adicionales para la interfaz Link

```
exception IllegalLinkName {
        LinkName name;
};

exception UnknownLinkName {
        LinkName name;
};

exception DuplicateLinkName {
        LinkName name;
};

exception DefaultFollowTooPermissive {
        FollowOption def_pass_on_follow_rule ;
        FollowOption limiting_follow_rule;
};
```

```
exception LimitingFollowTooPermissive {
        FollowOption limiting_follow_rule;
        FollowOption max_link_follow_policy;
};
```

8.3.1.5 Excepciones adicionales para la interfaz Proxy Offer

```
exception IllegalRecipe {
        ConstraintRecipe recipe;
};

exception NotProxyOfferId {
        OfferId id;
};
```

8.3.2 Para el módulo CosTradingDynamic

Hay sólo una interfaz DynamicPropEval en este módulo, y la interfaz sólo tiene una operación que produce la excepción:

```
exception DPEvalFailure {
        CosTrading::PropertyName name;
        TypeCode returned_type;
        any extra_info;
};
```

8.3.3 Para el módulo CosTradingRepos

Hay sólo la interfaz ServiceTypeRepository en este módulo. Pueden producirse las siguientes excepciones específicas de la interfaz:

```
exception ServiceTypeExists {
        CosTrading::ServiceTypeName name;
};

exception InterfaceTypeMismatch {
        CosTrading::ServiceTypeName base_service;
        Identifier base_if;
        CosTrading::ServiceTypeName derived_service;
        Identifier derived_if;
};

exception HasSubTypes {
        CosTrading::ServiceTypeName the_type;
        CosTrading::ServiceTypeName sub_type;
};

exception AlreadyMasked {
        CosTrading::ServiceTypeName name;
};

exception NotMasked {
        CosTrading::ServiceTypeName name;
};

exception ValueTypeRedefinition {
        CosTrading::ServiceTypeName type_1;
        PropStruct definition_1;
        CosTrading::ServiceTypeName type_2;
        PropStruct definition_2;
};

exception DuplicateServiceTypeName {
        CosTrading::ServiceTypeName name;
};
```

8.4 Interfaces abstractas

A fin de permitir la construcción de intermediarios con variada sustentación de las diferentes interfaces de intermediario, esta Especificación define varias interfaces abstractas de las cuales se derivan cada una de las interfaces funcionales de servicio de objeto intermediación (Lookup, Register, Link, Proxy y Admin). A continuación se ilustran cada una de estas interfaces abstractas.

8.4.1 TraderComponents

```
interface TraderComponents {

    readonly attribute Lookup lookup_if;
    readonly attribute Register register_if;
    readonly attribute Link link_if;
    readonly attribute Proxy proxy_if;
    readonly attribute Admin admin_if;
};
```

La funcionalidad de un intermediario puede ser configurada componiendo las interfaces definidas en una combinación de entre un número de combinaciones prescritas. La composición no se modela por herencia sino mediante múltiples interfaces a un objeto. Dada una de esas interfaces, se necesita una forma de encontrar las otras interfaces asociadas. Para facilitarlo, cada interfaz funcional de intermediario se obtiene de la interfaz TraderComponents.

La interfaz TraderComponents contiene cinco atributos de lectura solamente que proporcionan cada uno un medio de obtener una referencia de objeto específica.

El acceso a estos atributos debe retornar un objeto de referencia ninguno (nil) si el servicio de mediación en cuestión no sustenta esa determinada interfaz.

8.4.2 SupportAttributes

```
interface SupportAttributes {

    readonly attribute boolean supports_modifiable_properties;
    readonly attribute boolean supports_dynamic_properties;
    readonly attribute boolean supports_proxy_offers;
    readonly attribute TypeRepository type_repos;
};
```

Además de la posibilidad de una implementación de intermediario para elegir selectivamente qué interfaces funcionales sustentar, una implementación de intermediario puede también decidir no sustentar propiedades modificables, propiedades dinámicas y/u ofertas por procuración. La funcionalidad sustentada por una implementación de intermediario puede determinarse interrogando sobre los atributos de lectura solamente en esta interfaz.

El depositario de tipos que es utilizado por la implementación del intermediario puede también obtenerse de esta interfaz.

8.4.3 ImportAttributes

```
interface ImportAttributes {

    readonly attribute unsigned long def_search_card;
    readonly attribute unsigned long max_search_card;
    readonly attribute unsigned long def_match_card;
    readonly attribute unsigned long max_match_card;
    readonly attribute unsigned long def_return_card;
    readonly attribute unsigned long max_return_card;
    readonly attribute unsigned long max_list;
    readonly attribute unsigned long def_hop_count;
    readonly attribute unsigned long max_hop_count;
    readonly attribute FollowOption def_follow_policy;
    readonly attribute FollowOption max_follow_policy;
};
```

Cada intermediario se configura con valores por defecto y máximos de cierta cardinalidad y constricciones de seguimiento del enlace que se aplican a las interrogaciones. Los valores de estas constricciones pueden obtenerse interrogando los atributos en esta interfaz.

8.4.4 LinkAttributes

```
interface LinkAttributes {

    readonly attribute FollowOption max_link_follow_policy;
};
```

Cuando un intermediario crea un nuevo enlace o modifica uno existente, el atributo max_link_follow_policy determinará el comportamiento más permisivo que se le permitirá al enlace. El valor de esta constricción en la creación y modificación del enlace puede obtenerse de esta interfaz.

8.5 Interfaces funcionales

Esta subcláusula describe las cinco interfaces funcionales a un servicio objeto de intermediación: Lookup, Register, Link, Admin, y Proxy. Las dos interfaces de iterador necesitan también describirse para esta interfaz funcional.

8.5.1 Lookup

```
interface Lookup : TraderComponents, SupportAttributes, ImportAttributes {

    typedef Istring Preference;

    enum HowManyProps { none, some, all };

    union SpecifiedProps switch ( HowManyProps ) {
        case some: PropertyNameSeq prop_names;
    };

    exception IllegalPreference {
        Preference pref;
    };

    exception IllegalPolicyName {
        PolicyName name;
    };

    exception PolicyTypeMismatch {
        Policy the_policy;
    };

    exception InvalidPolicyValue {
        Policy the_policy;
    };

    void query (
        in ServiceTypeName type,
        in Constraint constr,
        in Preference pref,
        in PolicySeq policies,
        in SpecifiedProps desired_props,
        in unsigned long how_many,
        out OfferSeq offers,
        out OfferIterator offer_itr,
        out PolicyNameSeq limits_applied
    ) raises (
        IllegalServiceType,
        UnknownServiceType,
        IllegalConstraint,
        IllegalPreference,
        IllegalPolicyName,
        PolicyTypeMismatch,
        InvalidPolicyValue,
        IllegalPropertyName,
        DuplicatePropertyName,
        DuplicatePolicyName
    );
};
```

8.5.1.1 Operación interrogar

8.5.1.1.1 Signatura

```
void query (
        in ServiceTypeName type,
        in Constraint constr,
        in Preference pref,
        in PolicySeq policies,
        in SpecifiedProps desired_props,
        in unsigned long how_many,
        out OfferSeq offers,
        out OfferIterator offer_itr,
        out PolicyNameSeq limits_applied
```

```
) raises (
        IllegalServiceType,
        UnknownServiceType,
        IllegalConstraint,
        IllegalPreference,
        IllegalPolicyName,
        PolicyTypeMismatch,
        InvalidPolicyValue,
        IllegalPropertyName,
        DuplicatePropertyName,
        DuplicatePolicyName
);
```

8.5.1.1.2 Función

La operación interrogar es el medio por el cual un objeto puede obtener referencias a otros objetos que proporcionan servicios que cumplen sus requisitos.

El parámetro «type» transmite el tipo de servicio requerido. Es fundamental para el propósito central de la intermediación: efectuar una introducción para futuras interacciones seguras de tipos entre el importador y el exportador. Indicando un tipo de servicio, el importador implica el tipo de interfaz deseado y un campo de disertación para hablar sobre las propiedades del servicio. Si la representación en cadena del «type» no respeta las reglas de los identificadores de tipo de servicio, se produce entonces una excepción IllegalServiceType. Si el «type» es sintácticamente correcto, pero no es reconocido como un tipo de servicio dentro del alcance de intermediación, se produce entonces una excepción UnknownServiceType.

El intermediario puede retornar una oferta de servicio de un subtipo del «type» solicitado. La subtipificación de los tipos de servicio se trata en 8.2.3. Un subtipo de servicio tiene todas las propiedades obligatorias de sus supertipos. Se asegura así que lo que es una interrogación bien formada del «type» es también una interrogación bien formada respecto a cualesquiera subtipos. Sin embargo, si el importador especifica la política de exact_type_match = TRUE, se retornan entonces sólo ofertas con el tipo de servicio exacto (ningún subtipo) solicitado.

La constricción «constr» es el medio por el que el importador indica los requisitos de un servicio que no son captados en la signatura de la interfaz. Estos requisitos tratan del comportamiento computacional del servicio deseado, aspectos no funcionales y aspectos no computacionales (tales como la organización que posee los objetos que proporcionan el servicio). A un importador se le garantiza siempre que cualquier oferta retornada satisface la constricción de concordancia en el momento de importación. Si la «constr» no cumple las reglas de sintaxis de una expresión de constricción legal, que se define en el anexo B, se produce entonces una excepción IllegalConstraint.

El parámetro «pref» se utiliza para ordenar las ofertas que concuerdan con la «constr» a fin de que las ofertas retornadas por el intermediario estén en el orden de máximo interés para el importador. Si «pref» no cumple las reglas de sintaxis de una expresión de preferencia legal, se produce entonces una excepción IllegalPreference.

El parámetro «policies» permite al importador especificar cómo debe efectuarse la búsqueda en oposición a qué clase de servicios deben encontrarse en el curso de la búsqueda, lo cual puede verse parametrizando los algoritmos dentro de la implementación del intermediario. Las «policies» son una secuencia de pares nombre-valor. Los valores disponibles para un importador dependen de la implementación del intermediario. Sin embargo, algunos nombres están normalizados cuando efectúan la interpretación de otros parámetros o cuando pueden repercutir en el enlace y en la federación de intermediarios. Si un nombre de política en este parámetro no cumple las reglas sintácticas de los PolicyName legales, se produce entonces una excepción IllegalPolicyName. Si el tipo del valor asociado con una política difiere del especificado en esta Especificación, se produce entonces una excepción PolicyTypeMismatch. Si el procesamiento posterior de un PolicyValue produce errores (por ejemplo, el valor de política starting_trader está mal formado), se produce entonces una excepción InvalidPolicyValue. Si se incluye el mismo nombre de política dos o más veces en este parámetro, se produce entonces la excepción DuplicatePolicyName.

El parámetro «desired_props» define el conjunto de propiedades que describe las ofertas que son retornadas con la referencia objeto. Hay tres posibilidades: el importador no desea ninguna de las propiedades, todas las propiedades pero sin tener que denominarlas y, en tercer lugar, algunas propiedades cuyos nombres se proporcionan. Si cualquiera de los nombres «desired_props» no cumple las reglas de los identificadores, se produce entonces una excepción IllegalPropertyName. Si se incluye el mismo nombre de propiedad dos o más veces en este parámetro, se produce entonces la excepción DuplicatePropertyName (nombre de propiedad duplicado). El parámetro desired_props puede denominar propiedades que no son obligatorias para el tipo de servicio solicitado. Si la propiedad denominada está

presente en la oferta de servicio concordada, será retornada. El parámetro desired_props no afecta a que la oferta de servicio sea o no retornada. Para evitar que falten propiedades deseadas, el importador debe especificar «prop exists» en la constricción.

Las ofertas retornadas se devuelven de una de dos formas (o una combinación de ambas). El resultado de retorno «offers» transporta una lista de ofertas y la «offer_itr» es una referencia a una interfaz en la que pueden obtenerse ofertas. El parámetro «how_many» indica cuántas ofertas han de ser retornadas vía el resultado «offers»; cualesquiera ofertas restantes están disponibles vía la interfaz iterator. Si «how_many» excede el número de ofertas a retornar, la «offer_itr» será ninguna (nil).

Si se aplicó cualquier cardinalidad u otros límites por uno o más intermediarios al responder a una determinada interrogación, el parámetro «limits_applied» contendrá los nombres de las políticas que limitaron la interrogación. La secuencia de nombres retornados en «limits_applied» de cualesquiera interrogaciones federadas o por procuración, deben estar concatenadas a los nombres de los límites aplicados localmente y retornados.

8.5.1.1.3 Especificaciones de política del importador

```
struct LookupPolicies {
    unsigned long search_card;
    unsigned long match_card;
    unsigned long return_card;
    boolean use_modifiable_properties;
    boolean use_dynamic_properties;
    boolean use_proxy_offers;
    TraderName starting_trader;
    FollowOption link_follow_rule;
    unsigned long hop_count;
    boolean exact_type_match;
    OctetSeq request_id
};
```

La política «search_card» indica al intermediario el número máximo de ofertas que debe considerar cuando busque conformidad de tipo y concordancia de expresión de constricción. El menor de este valor y el atributo max_search_card del intermediario es utilizado por el intermediario. Si no se especifica esta política, se utiliza entonces el valor del atributo def_search_card del intermediario.

La política «match_card» indica al usuario el número máximo de ofertas de concordancia a las que debe aplicarse la especificación de preferencia. El menor de este valor y el atributo max_match_card del intermediario es utilizado por el intermediario. Si no se especifica esta política, se utiliza entonces el valor del atributo def_match_card del intermediario.

La política «return_card» indica al intermediario el número máximo de ofertas concordantes a retornar de resultas de esta interrogación. El menor entre este valor y el del atributo max_return_card del intermediario es utilizado por el intermediario. Si no se especifica esta política, se utiliza entonces el valor del atributo def_return_card del intermediario.

La política «use_modifiable_properties» indica si el intermediario debe considerar ofertas que tengan propiedades modificables cuando construya el conjunto de ofertas al que deben aplicarse la conformidad de tipo y el procesamiento de constricción. Si el valor de esta política es TRUE, se incluirán entonces tales ofertas; si es FALSE, no se incluirán. Si no se especifica esta política, se incluirán dichas ofertas.

La política «use_dynamic_properties» indica si el intermediario debe considerar ofertas que tengan propiedades dinámicas cuando construya el conjunto de ofertas a las que debe aplicarse la conformidad de tipo y el procesamiento de constricción. Si el valor de esta política es TRUE, se incluirán entonces dichas ofertas; si es FALSE, no se incluirán. Si no se especifica esta política, se incluirán dichas ofertas.

La política «use_proxy_offers» indica si el intermediario debe considerar ofertas por procuración cuando construya el conjunto de ofertas a las que debe aplicarse la conformidad de tipo y el procesamiento de constricción. Si el valor de esta política es TRUE, se incluirán entonces tales ofertas; si es FALSE, no se incluirán. Si no se especifica esta política, se incluirán dichas ofertas.

La política «starting_trader» facilita la distribución del propio servicio de intermediación. Permite a un importador determinar el alcance de una búsqueda eligiendo navegar explícitamente por los enlaces del gráfico de intermediación. Si se utiliza la política en una invocación de interrogación, se recomienda que sea el primer par política-valor, lo cual facilita un reenvío óptimo a la operación interrogar. Un parámetro «policies» no necesita incluir un valor para la política «starting_trader». Cuando esta política está presente, el primer componente de nombre es comparado con el nombre mantenido en cada enlace. Si no se encuentra concordancia, se produce la excepción InvalidPolicyValue. En otro caso, el intermediario invoca interrogación() en la interfaz Lookup mantenida por el enlace denominado, pero transmitiendo la política «starting_trader» con la eliminación del primer componente.

La política «link_follow_rule» indica cómo desea el cliente que los enlaces sean seguidos en la resolución de su interrogación. Véanse más detalles en el análisis de 8.2.7.

La política «hop_count» indica al intermediario el número máximo de saltos a través de enlaces de federación que deben tolerarse en la resolución de esta interrogación. La hop_count en el intermediario en curso se determina tomando el mínimo entre el atributo max_hop_count del intermediario y la política hop_count del importador, si es proporcionada, o el atributo def_hop_count del intermediario si no es así. Si el valor resultante es cero, no se permiten entonces interrogaciones federadas. Si es mayor que cero, deben decrementarse antes de transmitirse a un intermediario federado.

La política «exact_type_match» indica al intermediario si el tipo de servicio del importador debe exactamente concordar con un tipo de servicio de la oferta; si no es así (y por defecto), se considera entonces cualquier oferta de un tipo conforme al tipo de servicio del importador.

La política «request_id» indica al intermediario el identificador de una operación interrogar que es iniciada por un intermediario fuente que actúa como importador en un enlace. El id es generado utilizando el atributo request_id_stem. Un intermediario no está obligado a generar dicho id para una operación interrogar para otro intermediario, pero está obligado a transmitirla por un enlace a otro intermediario.

8.5.2 Offer Iterator

8.5.2.1 Signatura

```
interface OfferIterator {

    unsigned long max_left (
    ) raises (
        UnknownMaxLeft
    );

    boolean next_n (
        in unsigned long n,
        out OfferSeq offers
    );

    void destroy ();
};
```

8.5.2.2 Funciones de las operaciones de offer iterator

La interfaz OfferIterator se utiliza para retornar un conjunto de ofertas de servicio a partir de la operación interrogar permitiendo a las ofertas de servicio ser extraídas por operaciones sucesivas en la interfaz OfferIterator.

La operación next_n retorna un conjunto de ofertas de servicio en el parámetro de salida «offers». La operación retorna n ofertas de servicio si hay al menos n ofertas de servicio que permanecen en el iterador. Si hay menos de n ofertas en el iterador, se retornan todas las ofertas de servicio restantes. El número real de ofertas de servicio retornadas puede determinarse a partir de la longitud de la secuencia «offers». La operación next_n retorna TRUE si hay otras ofertas de servicio a extraer del iterador. Retorna FALSE si no hay otras ofertas de servicio que extraer.

La operación max_left retorna el número de ofertas de servicio restantes en el iterador. La excepción UnknownMaxLeft se produce si el iterador no puede determinar el número restante de ofertas de servicio (por ejemplo, si el iterador determina su conjunto de ofertas de servicio mediante evaluación perezosa).

La operación destruir destruye el iterador. No pueden invocarse otras operaciones en un iterador después que ha sido destruido.

8.5.3 Register

```
interface Register : TraderComponents, SupportAttributes {

    struct OfferInfo {
        Object reference;
        ServiceTypeName type;
        PropertySeq properties;
    };

    exception InvalidObjectRef {
        Object ref;
    };
```

```
exception UnknownPropertyName {
        PropertyName name;
};

exception InterfaceTypeMismatch {
        ServiceTypeName type;
        Object reference;
};

exception ProxyOfferId {
        OfferId id;
};

exception MandatoryProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception ReadonlyProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception NoMatchingOffers {
        Constraint constr;
};

exception IllegalTraderName {
        TraderName name;
};

exception UnknownTraderName {
        TraderName name;
};

exception RegisterNotSupported {
        TraderName name;
};

OfferId export (
        in Object reference,
        in ServiceTypeName type,
        in PropertySeq properties
) raises (
        InvalidObjectRef,
        IllegalServiceType,
        UnknownServiceType,
        InterfaceTypeMismatch,
        IllegalPropertyName, // e.g. prop_name = "<foo-bar"
        PropertyTypeMismatch,
        ReadonlyDynamicProperty,
        MissingMandatoryProperty,
        DuplicatePropertyName
);

void withdraw (
        in OfferId id
) raises (
        IllegalOfferId,
        UnknownOfferId,
        ProxyOfferId
);

OfferInfo describe (
        in OfferId id
) raises (
        IllegalOfferId,
        UnknownOfferId,
        ProxyOfferId
);
```

```
        void modify (
            in OfferId id,
            in PropertyNameSeq del_list,
            in PropertySeq modify_list
        ) raises (
            NotImplemented,
            IllegalOfferId,
            UnknownOfferId,
            ProxyOfferId,
            IllegalPropertyName,
            UnknownPropertyName,
            PropertyTypeMismatch,
            ReadonlyDynamicProperty,
            MandatoryProperty,
            ReadonlyProperty,
            DuplicatePropertyName
        );

        void withdraw_using_constraint (
            in ServiceTypeName type,
            in Constraint constr
        ) raises (
            IllegalServiceType,
            UnknownServiceType,
            IllegalConstraint,
            NoMatchingOffers
        );

        Register resolve (
            in TraderName name
        ) raises (
            IllegalTraderName,
            UnknownTraderName,
            RegisterNotSupported
        );
    };
```

8.5.3.1 Operación exportar

8.5.3.1.1 Signatura

```
        OfferId export (
            in Object reference,
            in ServiceTypeName type,
            in PropertySeq properties
        ) raises (
            InvalidObjectRef,
            IllegalServiceType,
            UnknownServiceType,
            InterfaceTypeMismatch,
            IllegalPropertyName, // e.g. prop_name = "<foo-bar"
            PropertyTypeMismatch,
            ReadonlyDynamicProperty,
            MissingMandatoryProperty,
            DuplicatePropertyName
        );
```

8.5.3.1.2 Función

La operación exportar es el medio por el que se anuncia un servicio, vía un intermediario, a una comunidad de importadores potenciales. El OfferId retornado es el asa con el que el exportador puede identificar la oferta exportada cuando intenta acceder a ella por otras operaciones. El OfferId es sólo significativo en el contexto del intermediario que lo generó.

El parámetro «reference» es la información que permite a un cliente interactuar con un servidor distante. Si la implementación de un intermediario decide considerar ciertos tipos de referencias de objeto [por ejemplo, una referencia de objeto nula (nil)] para que sea inexportable, pueda entonces retornar en tales casos la excepción InvalidObjectRef.

El parámetro «type» identifica el tipo de servicio, que contiene el tipo de interfaz de la «reference» y un conjunto de tipos de propiedad denominados que pueden utilizarse para describir aún más esta oferta; es decir, restringe lo que es aceptable en el parámetro properties. Si la representación encadenada del «type» no cumple las reglas de los identificadores, se produce entonces una excepción IllegalServiceType. Si el «type» es sintácticamente correcto pero un intermediario puede determinar inequívocamente que no es un tipo de servicio reconocido, se produce entonces una excepción UnknownServiceType. Si el intermediario puede determinar que el tipo de interfaz del parámetro «reference» no es un subtipo del tipo de interfaz especificado en «type», se produce entonces una excepción InterfaceTypeMismatch.

Si el parámetro «properties» es una lista de valores denominados que son conformes a los tipos de valor de propiedad definidos para esos nombres. Describen el servicio que se ofrece. Esta descripción suele comprender los aspectos de comportamiento, no funcionales y no computacionales del servicio. Si cualquiera de los nombres de propiedad no cumple las reglas de sintaxis para los PropertyNames, se produce entonces una excepción IllegalPropertyName. Si el tipo de cualquiera de los valores de propiedad no es el mismo que el tipo declarado (declarado en el tipo de servicio), se produce entonces una excepción PropertyTypeMismatch. Si se hace un intento de asignar un valor de propiedad dinámica debido a una propiedad de lectura solamente, se produce entonces la excepción ReadonlyDynamicProperty. Si el parámetro «properties» omite cualquier propiedad declarada en el tipo de servicio con un modo obligatorio, se produce entonces la excepción MissingMandatoryProperty. Si se incluyen en este parámetro dos o más propiedades con el mismo nombre de propiedad, se produce entonces la excepción DuplicatePropertyName.

8.5.3.2 Operación retirar

8.5.3.2.1 Signatura

```
void withdraw (
        in OfferId id
) raises (
        IllegalOfferId,
        UnknownOfferId,
        ProxyOfferId
);
```

8.5.3.2.2 Función

La operación retirar suprime del intermediario la oferta de servicio, es decir, una vez retirada, la oferta ya no puede ser retornada de resultas de una interrogación. La oferta es identificada por el parámetro «id» que fue originalmente retornado por exportación. Si la representación en cadena de «id» no cumple las reglas de los identificadores de oferta, se produce entonces una excepción IllegalOfferId. Si el «id» es legal, pero no hay ninguna oferta en el intermediario con ese «id», se produce entonces una excepción UnknownOfferId. Si el «id» identifica una oferta por procuración en lugar de una oferta ordinaria, se produce entonces una excepción ProxyOfferId.

8.5.3.3 Operación describir

8.5.3.3.1 Signatura

```
OfferInfo describe (
        in OfferId id
) raises (
        IllegalOfferId,
        UnknownOfferId,
        ProxyOfferId
);
```

8.5.3.3.2 Función

La operación describir retorna la información acerca del servicio ofrecido que es mantenida por el intermediario. Comprende la «reference» del servicio ofrecido, el «type» de la oferta de servicio y las «properties» que describen esta oferta de servicio. La oferta es identificada por el parámetro «id» que fue originalmente retornado por exportación. Si la representación en cadena de «id» no cumple las reglas de los identificadores de objeto se produce entonces una excepción IllegalOfferId. Si el «id» es legal, pero no hay ninguna oferta dentro del intermediario con ese «id», se produce entonces una excepción UnknownOfferId. Si el «id» identifica una oferta por procuración en lugar de una oferta ordinaria, se produce entonces una excepción ProxyOfferId.

8.5.3.4 Operación modificar

8.5.3.4.1 Signatura

```
void modify (
        in OfferId id,
        in PropertyNameSeq del_list,
        in PropertySeq modify_list
) raises (
        NotImplemented,
        IllegalOfferId,
        UnknownOfferId,
        ProxyOfferId,
        IllegalPropertyName,
        UnknownPropertyName,
        PropertyTypeMismatch,
        ReadonlyDynamicProperty,
        MandatoryProperty,
        ReadonlyProperty,
        DuplicatePropertyName  );
```

8.5.3.4.2 Función

La operación modificar se utiliza para cambiar la descripción de un servicio mantenida dentro de una oferta de servicio. La referencia de objeto y el tipo de servicio asociados con la oferta no pueden cambiarse. Esta operación puede:

a) añadir nuevas propiedades (no obligatorias) para describir una oferta;

b) cambiar los valores de algunas propiedades existentes (no de lectura solamente);

c) suprimir propiedades existentes (ni obligatorias ni de lectura solamente).

La operación modificar prospera por completo o fracasa por completo.

La oferta es identificada por el parámetro «id» que fue originalmente retornado por exportación. Si la representación en cadena de «id» no cumple las reglas de los identificadores de oferta, se produce entonces una excepción IllegalOfferId. Si el «id» es legal, pero no existe oferta dentro del intermediario con ese «id», se produce entonces una excepción UnknownOfferId. Si el «id» identifica una oferta por procuración en lugar de una oferta ordinaria, se produce entonces una excepción ProxyOfferId.

El parámetro «del_list» da los nombres de las propiedades que ya no han de registrarse para la oferta identificada. Las futuras operaciones interrogar y describir no verán estas propiedades. Si cualquiera de los nombres contenidos en la «del_list» no cumple las reglas de los PropertyName, se produce entonces una excepción IllegalPropertyName. Si un «name» es legal, pero no existe propiedad para la oferta con ese «name», se produce entonces una excepción UnknownPropertyName. Si la lista incluye una propiedad que tiene un modo obligatorio, se produce entonces la excepción MandatoryProperty. Si el mismo nombre de propiedad es incluido dos o más veces en este parámetro, se produce entonces la excepción DuplicatePropertyName.

El parámetro «modify_list» da los nombres y valores de las propiedades a cambiar. Si la propiedad no está en la oferta, se añade entonces a ella la operación modificar. Los valores de propiedad modificados (o añadidos) son retornados en futuras operaciones interrogar y describir en lugar de los valores originales. Si cualquiera de los nombres de la «modify_list» no cumple las reglas de los PropertyName, se produce entonces una excepción IllegalPropertyName. Si la lista incluye una propiedad que tiene un modo lectura solamente, se produce entonces la excepción ReadOnlyProperty, a menos que la propiedad de lectura solamente no está en ese momento registrada para la oferta. La excepción ReadOnlyDynamicProperty se produce si se hace un intento de asignar un valor de propiedad dinámica a una propiedad lectura solamente. Si el valor de cualquier propiedad modificada es de un tipo que no es el esperado, se produce entonces la excepción PropertyTypeMismatch. Si se incluyen en este argumento dos o más propiedades con el mismo nombre de propiedad, se produce entonces la excepción DuplicatePropertyName.

La excepción NotImplemented se producirá si y sólo si el atributo supports_modifiable_properties da FALSE.

No es posible cambiar el tipo de servicio de una oferta o la referencia de objeto del servicio. Esto ha de conseguirse retirando y reexportando luego. La finalidad de modificar es cambiar la descripción del servicio ofrecido, pero preservando el OfferId, lo cual podría ser importante cuando se ha propagado el OfferId a toda una comunidad de objetos.

8.5.3.5 Retirar utilizando la operación constricción

8.5.3.5.1 Signatura

```
void withdraw_using_constraint (
        in ServiceTypeName type,
        in Constraint constr
) raises (
        IllegalServiceType,
        UnknownServiceType,
        IllegalConstraint,
        NoMatchingOffers
);
```

8.5.3.5.2 Función

La operación withdraw_using_constraint retira un conjunto de ofertas contenidas en un intermediario único. Este conjunto se identifica de la misma manera que la operación interrogar identifica un conjunto de ofertas a retornar a un importador.

El parámetro «type» transporta el tipo de servicio requerido. Cada oferta del tipo especificado tendrá la expresión constricción aplicada al mismo. Si concuerda con la expresión constricción, se retirará entonces la oferta.

Si «type» no cumple las reglas de los tipos de servicio, se produce entonces una excepción IllegalServiceType. Si el «type» es sintácticamente correcto, pero no es reconocido como un tipo de servicio por el intermediario, se produce entonces una excepción UnknownServiceType.

La constricción «constr» es el medio por el que el cliente restringe el conjunto de ofertas a las que están destinadas a su retirada. Si «constr» no cumple las reglas de sintaxis de una constricción, se produce entonces una excepción IllegalConstraint. Si la constricción no concuerda con ninguna oferta del tipo de servicio especificado, se produce entonces una excepción NoMatchingOffers.

8.5.3.6 Operación resolver

8.5.3.6.1 Signatura

```
Register resolve (
        in TraderName name
) raises (
        IllegalTraderName,
        UnknownTraderName,
        RegisterNotSupported
);
```

8.5.3.6.2 Función

Esta operación se utiliza para resolver un nombre relativo de contexto para otro intermediario. En particular, se utiliza cuando se exporta a un intermediario que es conocido por un nombre distinto de una referencia de interfaz. El cliente proporciona el nombre, que será una secuencia de componentes de nombre. Si el contenido del parámetro no puede dar sintaxis legal para el primer componente, se produce entonces la excepción IllegalTraderName. En otro caso, el primer componente de nombre se compara con el nombre mantenido en cada enlace. Si no se encuentra concordancia, o el intermediario no sustenta enlaces, se produce entonces la excepción UnknownTraderName. En otro caso, el intermediario obtiene el register_if mantenido como parte del enlace concordado. Si la referencia de la interfaz Register no es nula (nil), el intermediario vincula la interfaz Register e invoca resolver, pero transmite el TraderName con el primer componente eliminado; si es nula (nil), se produce entonces la excepción RegisterNotSupported. Cuando un intermediario es capaz de concordar el primer componente de nombre sin dejar ningún nombre residual, el intermediario retorna entonces la referencia de la interfaz Register para ese intermediario enlazado. Al rebobinar la recursión, los intermediarios intermedios retornan a su cliente (otro intermediario) la referencia de interfaz Register.

8.5.4 Offer Id Iterator

8.5.4.1 Signatura

```
interface OfferIdIterator {

        unsigned long max_left (
        ) raises (
                UnknownMaxLeft
        );
```

```
        boolean next_n (
            in unsigned long n,
            out OfferIdSeq ids
        );

        void destroy ();
    };
```

8.5.4.2 Funciones de las operaciones de offer id iterator

La interfaz OfferIdIterator se utiliza para retornar un conjunto de identificadores de oferta a partir de la operación list_offers y la operación list_proxies en la interfaz Admin permitiendo la extracción de los identificadores de oferta por sucesivas operaciones en la interfaz OfferIdIterator.

La operación next_n retorna un conjunto de identificadores de oferta en el parámetro de salida «ids». La operación retorna n identificadores de oferta si hay al menos n identificadores de oferta que permanecen en el iterador. Si hay menos de n identificadores de oferta en el iterador, se retornan todos los identificadores de oferta restantes. El número real de identificadores de oferta retornados puede determinarse a partir de la longitud de la secuencia «ids». La operación next_n retorna TRUE si hay otros identificadores de oferta a extraer del iterador. Retorna FALSE si no hay otros identificadores de oferta que extraer.

La operación max_left retorna el número de identificadores de oferta restantes en el iterador. La excepción UnknownMaxLeft se produce si el iterador no puede determinar el número restante de identificadores de oferta (por ejemplo, si el iterador determina su conjunto de identificadores de oferta mediante evaluación perezosa).

La operación destruir destruye el iterador. No pueden invocarse otras operaciones en un iterador después que ha sido destruido.

8.5.5 Admin

```
interface Admin : TraderComponents, SupportAttributes, ImportAttributes,
                  LinkAttributes {

    typedef sequence<octet> OctetSeq;

    readonly attribute OctetSeq request_id_stem;

    unsigned long set_def_search_card (in unsigned long value);
    unsigned long set_max_search_card (in unsigned long value);

    unsigned long set_def_match_card (in unsigned long value);
    unsigned long set_max_match_card (in unsigned long value);

    unsigned long set_def_return_card (in unsigned long value);
    unsigned long set_max_return_card (in unsigned long value);

    unsigned long set_max_list (in unsigned long value);

    boolean set_supports_modifiable_properties (in boolean value);
    boolean set_supports_dynamic_properties (in boolean value);
    boolean set_supports_proxy_offers (in boolean value);

    unsigned long set_def_hop_count (in unsigned long value);
    unsigned long set_max_hop_count (in unsigned long value);

    FollowOption set_max_follow_policy (in FollowOption policy);
    FollowOption set_def_follow_policy (in FollowOption policy);

    FollowOption set_max_link_follow_policy (in FollowOption policy);

    TypeRepository set_type_repos (in TypeRepository repository);

    OctetSeq set_request_id_stem (in OctetSeq stem);

    void list_offers (
        in unsigned long how_many,
        out OfferIdSeq ids,
        out OfferIdIterator id_itr
    ) raises (
        NotImplemented
    );
```

```
        void list_proxies (
            in unsigned long how_many,
            out OfferIdSeq ids,
            out OfferIdIterator id_itr
        ) raises (
            NotImplemented
        );
};
```

8.5.5.1 Atributos y operaciones fijar (set)

La interfaz Admin permite que los valores de los atributos de intermediario sean leídos y escritos. Todos los atributos se definen como de lectura solamente en SupportAttributes, ImportAttributes, LinkAttributes, o Admin. Para fijar el «attribute» de intermediario a un nuevo valor, las operaciones set_<attribute_name> se definen en Admin. Cada una de estas operaciones fijar retorna el valor previo del atributo como su valor de función.

Si la operación de la interfaz Admin set_support_proxy_offers es invocada con el valor fijado a FALSE, para un intermediario que soporta la proxy_interface, el valor set_support_proxy_offer no afecta a la función de operaciones en la interfaz Proxy. Sin embargo, en este caso, el efecto del valor support_proxy_offers que se fija a FALSE tiene el efecto de hacer cualesquiera ofertas por procuración exportadas vía la interfaz Proxy para ese intermediario no disponibles para satisfacer interrogaciones en esa interfaz Lookup de intermediario.

8.5.5.2 Operación listar ofertas

8.5.5.2.1 Signatura

```
        void list_offers (
            in unsigned long how_many,
            out OfferIdSeq ids,
            out OfferIdIterator id_itr
        ) raises (
            NotImplemented
        );
```

8.5.5.2.2 Función

La operación list_offers permite al administrador de un intermediario efectuar trabajos auxiliares (housekeeping) obteniendo un asa en cada una de las ofertas contenidas en un intermediario por ejemplo, para la recogida de residuos, etc. Sólo se retornan los identificadores de ofertas ordinarias, los identificadores de ofertas por procuración no se retornan mediante esta operación. Si el intermediario no sustenta la interfaz Register, se produce la excepción NotImplemented.

Los identificadores retornados se devuelven de una o dos maneras (o una combinación de ambas). El resultado de retorno «ids» transporta una lista de identificadores de oferta y la «id_itr» es una referencia a una interfaz en la que pueden obtenerse identidades de oferta adicionales. El parámetro «how_many» indica cuántos identificadores han de retornarse mediante el resultado «ids»; los restantes están disponibles mediante la interfaz Iterator. Si «how_many» excede el número de ofertas conservadas en el intermediario, la «id_itr» es nula (nil).

8.5.5.3 Operación listar procuraciones

8.5.5.3.1 Signatura

```
        void list_proxies (
            in unsigned long how_many,
            out OfferIdSeq ids,
            out OfferIdIterator id_itr
        ) raises (
            NotImplemented
        );
```

8.5.5.3.2 Función

La operación list_proxies retorna el conjunto de identificadores de oferta para ofertas por procuración mantenidas por un intermediario. A lo sumo, se retornan identificadores de oferta «how_many» mediante «ids». Si hay más que identificadores de oferta «how_many», los restantes se retornan mediante el iterador «id_itr». Si hay sólo identificadores «how_many» o de ofertas menores, la id_itr es nula (nil). Si el intermediario no sustenta la interfaz Proxy, se produce la excepción NotImplemented.

8.5.6 Link

```
interface Link : TraderComponents, SupportAttributes,
                                 LinkAttributes {

    struct LinkInfo {
        Lookup target;
        Register target_reg;
        FollowOption def_pass_on_follow_rule ;
        FollowOption limiting_follow_rule;
    };

    exception IllegalLinkName {
        LinkName name;
    };

    exception UnknownLinkName {
        LinkName name;
    };

    exception DuplicateLinkName {
        LinkName name;
    };

    exception DefaultFollowTooPermissive {
        FollowOption def_pass_on_follow_rule ;
        FollowOption limiting_follow_rule;
    };

    exception LimitingFollowTooPermissive {
        FollowOption limiting_follow_rule;
        FollowOption max_link_follow_policy;
    };

    void add_link (
        in LinkName name,
        in Lookup target,
        in FollowOption def_pass_on_follow_rule ,
        in FollowOption limiting_follow_rule
    ) raises (
        IllegalLinkName,
        DuplicateLinkName,
        InvalidLookupRef, // e.g. nil
        DefaultFollowTooPermissive,
        LimitingFollowTooPermissive
    );

    void remove_link (
        in LinkName name
    ) raises (
        IllegalLinkName,
        UnknownLinkName
    );

    LinkInfo describe_link (
        in LinkName name
    ) raises (
        IllegalLinkName,
        UnknownLinkName
    );

    LinkNameSeq list_links ();

    void modify_link (
        in LinkName name,
        in FollowOption def_pass_on_follow_rule ,
        in FollowOption limiting_follow_rule
    ) raises (
        IllegalLinkName,
        UnknownLinkName,
        DefaultFollowTooPermissive,
        LimitingFollowTooPermissive
    );
};
```

8.5.6.1 Operación Add_Link

8.5.6.1.1 Signatura

```
void add_link (
        in LinkName name,
        in Lookup target,
        in FollowOption def_pass_on_follow_rule ,
        in FollowOption limiting_follow_rule
) raises (
        IllegalLinkName,
        DuplicateLinkName,
        InvalidLookupRef, // e.g. nil
        DefaultFollowTooPermissive,
        LimitingFollowTooPermissive
);
```

8.5.6.1.2 Función

La operación add_link permite a un intermediario utilizar posteriormente el servicio de otro intermediario en la ejecución de sus propias operaciones del servicio de intermediación.

El parámetro «name» se utiliza en operaciones de gestión de enlaces posteriores para identificar el enlace deseado. Si el parámetro no está legalmente formado, se produce entonces la excepción IllegalLinkName. Se produce una excepción de DuplicateLinkName si ya existe el nombre del enlace. El nombre del enlace se utiliza también como un componente en una secuencia de componentes de nombre al denominar un intermediario para resolver o enviar operaciones. La secuencia de nombres de enlace relativos al contexto proporciona un trayecto a un intermediario.

El parámetro «target» identifica la interfaz Lookup en la que puede accederse al servicio de intermediación proporcionado por el intermediario destinatario. En caso de que el parámetro de la interfaz Lookup sea nulo (nil), se produce una excepción de InvalidLookupRef. La interfaz destinataria se utiliza para obtener la interfaz Register asociada, que posteriormente se retornará como parte de una operación describe_link y se invocará como parte de una operación resolver.

El parámetro «def_pass_on_follow_rule» especifica el comportamiento de enlace por defecto para el enlace si no se especifica ningún comportamiento de enlace en una petición de interrogación del importador. Si la «def_pass_on_follow_rule» excede la «limiting_follow_rule» especificada en el parámetro siguiente, se produce entonces una excepción DefaultFollowTooPermissive.

El parámetro «limiting_follow_rule» especifica el comportamiento de seguimiento del enlace más permisivo que el enlace desea tolerar. La excepción LimitingFollowTooPermissive se produce si este parámetro excede el atributo de intermediario de la «max_link_follow_policy» en el momento de la creación del enlace.

NOTA – Es posible para una «limiting_follow_rule» de un enlace exceder la «max_link_follow_policy» del intermediario posteriormente en la vida de un enlace, ya que es posible que el intermediario pudiera fijar su «max_link_follow_policy» a un valor más restrictivo después de la creación del enlace.

8.5.6.2 Operación suprimir enlace

8.5.6.2.1 Signatura

```
void remove_link (
        in LinkName name
) raises (
        IllegalLinkName,
        UnknownLinkName
);
```

8.5.6.2.2 Función

La operación remove_link elimina todo conocimiento del intermediario destinatario. El intermediario destinatario no puede posteriormente ser utilizado para resolver, enviar, o propagar operaciones de intermediación, desde este intermediario.

El parámetro «name» identifica el enlace a suprimir. La excepción IllegalLinkName se produce si el enlace está deficientemente formado y se produce la excepción UnknownLinkName y el enlace denominado no está en el intermediario.

8.5.6.3 Operación describir enlace

8.5.6.3.1 Signatura

```
LinkInfo describe_link (
        in LinkName name
) raises (
        IllegalLinkName,
        UnknownLinkName
);
```

8.5.6.3.2 Función

La operación describe_link retorna información sobre un enlace mantenida en el intermediario.

El parámetro «name» identifica el enlace cuya descripción se requiere. Para un nombre de enlace mal formado, se produce la excepción IllegalLinkName. Se produce una excepción UnknownLinkName si el enlace denominado no se encuentra en el intermediario.

La operación retorna una estructura LinkInfo que comprende: la interfaz Lookup en el servicio de intermediación destinatario, la interfaz Register en el servicio de intermediación destinatario, y el comportamiento por defecto, así como el comportamiento de seguimiento limitador del enlace denominado. Si el servicio destinatario no sustenta la interfaz Register, el campo de la estructura LinkInfo es entonces nulo (nil).

NOTA – Dada la descripción de la operación Register::resolve() en 8.5.3.6, la mayoría de las implementaciones puede optar por determinar la interfaz Register cuando se llama add_link y almacenando esa información estáticamente con el resto del estado del enlace.

8.5.6.4 Operación listar enlaces

8.5.6.4.1 Signatura

```
LinkNameSeq list_links ();
```

8.5.6.4.2 Función

La operación list_links retorna una lista de nombres de todos los enlaces de intermediación contenidos en el intermediario. Los nombres pueden posteriormente utilizarse para otras operaciones de gestión, tales como describe_link_o remove_link.

8.5.6.5 Operación modificar enlace

8.5.6.5.1 Signatura

```
void modify_link (
        in LinkName name,
        in FollowOption def_pass_on_follow_rule ,
        in FollowOption limiting_follow_rule
) raises (
        IllegalLinkName,
        UnknownLinkName,
        DefaultFollowTooPermissive,
        LimitingFollowTooPermissive
);
```

8.5.6.5.2 Función

La operación modify_link se utiliza para cambiar los comportamientos de seguimiento de enlaces existentes de un enlace identificado. La referencia de la interfaz Lookup del intermediario destinatario y el nombre del enlace no pueden cambiarse.

El parámetro «name» identifica el enlace cuyos comportamientos de seguimiento han de cambiarse. Un «name» deficientemente formado produce la excepción IllegalLinkName. Una excepción UnknownLinkName se produce si el nombre del enlace no resulta conocido al intermediario.

El parámetro «def_pass_on_follow_rule» especifica el nuevo comportamiento de enlace por defecto de este enlace. Si la «def_pass_on_follow_rule» excede la «limiting_follow_rule» especificada en el parámetro siguiente, se produce entonces una excepción DefaultFollowTooPermissive.

El parámetro «limiting_follow_rule» especifica el nuevo límite para el comportamiento de seguimiento de este enlace. La excepción LimitingFollowTooPermissive se produce si el valor excede la «max_link_follow_policy» del intermediario.

8.5.7 Proxy

```
interface Proxy : TraderComponents, SupportAttributes {

    typedef Istring ConstraintRecipe;

    struct ProxyInfo {
        ServiceTypeName type;
        Lookup target;
        PropertySeq properties;
        boolean if_match_all;
        ConstraintRecipe recipe;
        PolicySeq policies_to_pass_on;
    };

    exception IllegalRecipe {
        ConstraintRecipe recipe;
    };

    exception NotProxyOfferId {
        OfferId id;
    };

    OfferId export_proxy (
        in Lookup target,
        in ServiceTypeName type,
        in PropertySeq properties,
        in boolean if_match_all,
        in ConstraintRecipe recipe,
        in PolicySeq policies_to_pass_on
    ) raises (
        IllegalServiceType,
        UnknownServiceType,
        InvalidLookupRef, // e.g. nil
        IllegalPropertyName,
        PropertyTypeMismatch,
        ReadonlyDynamicProperty,
        MissingMandatoryProperty,
        IllegalRecipe,
        DuplicatePropertyName,
        DuplicatePolicyName
    );

    void withdraw_proxy (
        in OfferId id
    ) raises (
        IllegalOfferId,
        UnknownOfferId,
        NotProxyOfferId
    );

    ProxyInfo describe_proxy (
        in OfferId id
    ) raises (
        IllegalOfferId,
        UnknownOfferId,
        NotProxyOfferId
    );
};
```

8.5.7.1 Operación exportar procuración

8.5.7.1.1 Signatura

```
OfferId export_proxy (
    in Lookup target,
    in ServiceTypeName type,
    in PropertySeq properties,
    in boolean if_match_all,
    in ConstraintRecipe recipe,
    in PolicySeq policies_to_pass_on
```

```
) raises (
        IllegalServiceType,
        UnknownServiceType,
        InvalidLookupRef, // e.g. nil
        IllegalPropertyName,
        PropertyTypeMismatch,
        ReadonlyDynamicProperty,
        MissingMandatoryProperty,
        IllegalRecipe,
        DuplicatePropertyName,
        DuplicatePolicyName
);
```

8.5.7.1.2 Función

La interfaz Proxy permite la exportación y posterior manipulación de las ofertas por procuración. Las ofertas por procuración permiten la determinación del tiempo de funcionamiento de la interfaz en la que se proporciona un servicio. La operación export_proxy añade una oferta por procuración al conjunto de ofertas de servicio del intermediario.

Como las ofertas de servicio normales, las ofertas por procuración tienen un tipo de servicio «type» y valores de propiedad denominados «properties». Sin embargo, una oferta por procuración no incluye una referencia de objeto en la cual se proporciona el servicio ofrecido. En su lugar, esta referencia de objeto se obtiene cuando se necesita para una operación interrogar; se obtiene invocando otra operación interrogar sobre la interfaz Lookup «destinataria» mantenida en la oferta por procuración.

El parámetro «if_match_all», si es TRUE, indica que el intermediario debe considerar esta oferta por procuración como una concordación con una interrogación de un importador basada solamente en la conformidad de tipo, es decir, no concuerda la expresión de constricción del importador con las propiedades asociadas con la oferta por procuración. Muy a menudo esto es útil cuando la expresión de constricción suministrada por el importador se transfiere simplemente en la operación interrogar secundaria.

El parámetro «recipe» (receta) dice al intermediario cómo construir la expresión de constricción para la operación interrogar secundaria al «destinatario». El lenguaje «recipe» se describe en el anexo C; permite a la expresión de constricción secundaria estar compuesta por literales, valores de propiedades de la oferta por procuración, y la expresión de constricción primaria.

El parámetro «policies_to_pass_on» proporciona un conjunto estático de pares <nombre, valor> para retransmitir al intermediario «destinatario». El cuadro que sigue describe cómo se genera el parámetro policy secundario a partir del parámetro policy primario y las «policies_to_pass_on».

Si una operación interrogar concuerda la oferta por procuración (utilizando la concordancia de tipo de servicio normal y la concordancia de propiedad y algoritmos de preferencia), esta operación interrogar primaria invoca una operación interrogar secundaria en la interfaz Lookup designada en la oferta por procuración. Aunque la oferta por procuración designa una interfaz Lookup, esta interfaz sólo necesita conformarse sintácticamente a la interfaz Lookup; no necesita conformarse al comportamiento de la interfaz Lookup arriba especificado.

La operación interrogar secundaria se invoca como sigue:

en ServiceTypeName type	El tipo se copia de la interrogación primaria.
en Constraint constr	La receta en la oferta por procuración se «evalúa» para proporcionar el parámetro constr.
en Preference pref	La preferencia se copia de la interrogación primaria.
en PolicySeq policies	Las «policies» (nombres y valores) contenidas en el campo policies_to_pass_on de la oferta por procuración son agregadas a las políticas de la interrogación primaria.
en SpecifiedProps desired_props	Las desired_props se copian de la interrogación primaria.
en unsigned long how_many	El parámetro how_many es fijado por el intermediario para reflejar la preferencia de la implementación de intermediario por recibir la oferta resultante como una lista o mediante un iterador.

fuera de OfferSeq offers	A lo sumo se retornan ofertas how_many de la operación interrogar secundaria vía ofertas.
fuera de OfferIterator offer_itr	Si la interrogación secundaria necesita retornar más que ofertas how_many, puede entonces accederse a las restantes ofertas mediante el iterador offer_itr. Si hay solamente ofertas *how_many* o menores, la offer_itr es nula (nil).
fuera de PolicyNameSeq limits_applied	Los nombres de cualesquiera límites de política que fueron aplicados por el intermediario por procuración.

La excepción IllegalServiceType se produce si es el nombre de tipo de servicio (type) no está bien formado. La excepción UnknownServiceType se produce si el nombre de tipo de servicio (type) no es conocido al intermediario. La excepción InvalidLookupRef se produce si el destinatario no es una referencia de interfaz Lookup válida [por ejemplo, si el destinatario es una referencia de objeto nula (nil)]. La excepción IllegalPropertyName se produce si el nombre de propiedad en «properties» no está bien formado. La excepción PropertyTypeMismatch se produce si un valor de propiedad no es de un tipo apropiado determinado por el tipo de servicio. La excepción ReadonlyDynamicProperty se produce si un valor de propiedad dinámica fue suministrado para una propiedad que fue consignada como de lectura solamente. La excepción MissingMandatoryProperty se produce si «properties» no contiene una de las propiedades obligatorias definidas por este tipo de servicio. La excepción IllegalRecipe se produce si la receta no está bien formada. La excepción DuplicatePropertyName se produce si se incluyen dos o más propiedades con el mismo nombre de propiedad en el parámetro «properties». La excepción DuplicatePolicyName se produce si se incluyen dos o más políticas con el mismo nombre de política en el parámetro «policies_to_pass_on».

Las ofertas por procuración no pueden ser modificadas; deben retirarse y reexportarse.

8.5.7.2 Operación retirar procuración

8.5.7.2.1 Signatura

```
void withdraw_proxy (
        in OfferId id
) raises (
        IllegalOfferId,
        UnknownOfferId,
        NotProxyOfferId
);
```

8.5.7.2.2 Función

La operación withdraw_proxy elimina la oferta por procuración identificada por «id» procedente del intermediario.

La excepción IllegalOfferId se produce si «id» no está bien formada. La excepción UnknownOfferId se produce si «id» no identifica ninguna oferta mantenida por el intermediario. La excepción NotProxyOfferId se produce si «id» identifica una oferta de servicio normal en lugar de una oferta por procuración.

8.5.7.3 Operación describir procuración

8.5.7.3.1 Signatura

```
ProxyInfo describe_proxy (
        in OfferId id
) raises (
        IllegalOfferId,
        UnknownOfferId,
        NotProxyOfferId
);
```

8.5.7.3.2 Función

La operación describe_proxy retorna la información contenida en la oferta por procuración identificada por «id» en el intermediario.

La excepción IllegalOfferId se produce si «id» no está bien formado. La excepción UnknownOfferId se produce si «id» no identifica ninguna oferta mantenida por el intermediario. La excepción NotProxyOfferId se produce si «id» identifica una oferta de servicio normal en lugar de una oferta por procuración.

8.6 Interfaz de evaluación de propiedad dinámica

8.6.1 Signatura

```
module CosTradingDynamic {

        exception DPEvalFailure {
            CosTrading::PropertyName name;
            TypeCode returned_type;
            any extra_info;
        };

        interface DynamicPropEval {

            any evalDP (
                in CosTrading::PropertyName name,
                in TypeCode returned_type,
                in any extra_info)
            raises (DPEvalFailure);
        };

        struct DynamicProp {
            DynamicPropEval eval_if;
            TypeCode returned_type;
            any extra_info;
        };
};
```

8.6.2 Funciones de las operaciones de la interfaz DynamicProperty Eval

La interfaz DynamicPropEval es proporcionada por un exportador que desea proporcionar un valor de propiedad dinámica en una oferta de servicio mantenida por el intermediario.

Cuando exporte una oferta de servicio (o una oferta por procuración), la propiedad con el valor dinámico tiene un valor «any» que contiene una estructura DynamicProp más que un valor de propiedad normal. Un intermediario que sustenta propiedades dinámicas acepta este valor DynamicProp que contiene la información que permite a un valor de propiedad correctamente tipificado ser obtenido durante la evaluación de una interrogación. La operación exportar (o export_proxy) produce la PropertyTypeMismatch si el returned_type no es apropiado para el nombre de propiedad definido por el tipo de servicio.

Las propiedades de lectura solamente pueden no tener valores dinámicos. Las operaciones exportar y modificar en la interfaz Register y la operación export_proxy en la interfaz Proxy producen la excepción ReadonlyDynamicProperty si se asignan valores dinámicos a propiedades de lectura solamente.

Cuando una interrogación requiere un valor de propiedad dinámica, se invoca la operación evalDP en la interfaz eval_if de la estructura DynamicProp. El parámetro de nombre de propiedad es el nombre de la propiedad cuyo valor se está obteniendo. Los parámetros returned_type y extra_info se copian de la estructura DynamicProp. La operación evalDP retorna cualquier valor que deba contener un valor para esa propiedad. El valor debe ser de un tipo indicado por returned_type.

La excepción DPEvalFailure se produce si no puede determinarse el valor de la propiedad. Si el valor se requiere para la evaluación de una constricción o preferencia, esa evaluación se considera entonces que ha fracasado en esa oferta de servicio (u oferta por procuración).

Además de las reglas precedentes, el comportamiento adicional de la operación evalDP no es especificado por esta Especificación. En particular, la finalidad de los datos extra_info al determinar la propiedad dinámica es específico de la implementación.

Si el intermediario no sustenta propiedades dinámicas (indicadas por el atributo supports_dynamic_properties), las operaciones exportar y export_proxy no deben ser parametrizadas por propiedades dinámicas. El comportamiento de dichos intermediarios en tales circunstancias no es especificado por esta Especificación.

Si el intermediario no sustenta propiedades dinámicas o el importador ha solicitado que no se utilicen propiedades dinámicas (mediante el parámetro policies de la operación interrogar), no se efectúa la evaluación de la propiedad dinámica. Si el valor de una propiedad dinámica es requerido por la evaluación de una constricción o preferencia, se considera entonces que esa evaluación ha fracasado en esa oferta de servicio (u oferta por procuración).

La operación describir de la interfaz Register y la operación describe_proxy_de la interfaz Proxy no efectúa la evaluación de propiedad dinámica, pero retorna la estructura DynamicProp como valor de la propiedad. Como estas interfaces se utilizan para crear propiedades dinámicas vía las operaciones exportar y export_proxy, las otras operaciones en estas interfaces deben asegurar que la naturaleza dinámica de las propiedades permanezca visible a los exportadores.

La operación modificar en la interfaz Register de un intermediario que sustenta propiedades dinámicas debe aceptar el establecimiento y la modificación de propiedades dinámicas, consecuente con la operación exportar. No hay ninguna restricción sobre un valor de propiedad que cambie de un valor estático almacenado por el intermediario a un valor dinámico, y viceversa; sin embargo, las propiedades estáticas de lectura solamente no pueden modificarse para que sean dinámicas.

8.7 Plantilla de objeto de intermediario

Una plantilla de objeto computacional básica para el intermediario contiene:

- plantillas de interfaz computacional para las interfaces que puede instanciar;
- una especificación de comportamiento;
- una especificación de contrato de entorno.

8.7.1 Plantillas de interfaz

Esta Especificación proporciona las siguientes plantillas de interfaz:

- plantilla de interfaz Trader Components (componentes de intermediario);
- plantilla de interfaz Support Attributes (atributos de sustentación);
- plantilla de interfaz Import Attributes (atributos de importación);
- plantilla de interfaz Link Attributes (atributos de enlace);
- plantilla de interfaz Lookup (consulta);
- plantilla de interfaz Register (registro);
- plantilla de interfaz Admin (administración);
- plantilla de interfaz Link (enlace);
- plantilla de interfaz Proxy (procuración);
- plantilla de interfaz Offer Iterator (iterador de oferta);
- plantilla de interfaz Offer Id Iterator (iterador de id de oferta);
- plantilla de interfaz Dynamic Property Evaluator (evaluador de propiedad dinámica).

8.7.2 Especificación de comportamiento

La instanciación de una plantilla de interfaz exige un objeto que cumpla las especificaciones de comportamiento asociadas con las operaciones contenidas en la interfaz.

La instanciación de una plantilla de interfaz en el rol cliente exige un objeto que sea capaz de recibir todas las terminaciones posibles para las operaciones que invoca por esa interfaz.

Cuando un objeto intermediario instancia una interfaz lookup en el rol cliente a fin de sustentar servicios invocados en una interfaz lookup en su rol servidor, transmitirá las respuestas recibidas de esa interfaz de rol cliente a la interfaz de rol servidor, de acuerdo con la especificación de comportamiento de la plantilla de interfaz lookup.

Cuando un objeto intermediario instancia una interfaz register en el rol cliente a fin de sustentar servicios invocados en una interfaz lookup en su rol servidor, transmitirá las respuestas recibidas de esa interfaz de rol cliente a la interfaz de rol servidor, de acuerdo con la especificación de comportamiento de la plantilla de interfaz lookup.

Cuando un objeto intermedio instancia una interfaz register en el rol cliente a fin de sustentar servicios invocados en una interfaz register en su rol servidor, transmitirá las respuestas recibidas de esa interfaz de rol cliente a la interfaz de rol servidor, de acuerdo con la especificación de comportamiento de la plantilla de interfaz register.

Además, hay otras acciones que el intermediario puede realizar.

8.7.2.1 Instanciación de plantillas de interfaz

Pueden instanciarse las siguientes plantillas:

- plantilla de interfaz Lookup (rol servidor);
- plantilla de interfaz Lookup (rol cliente);
- plantilla de interfaz Register (rol servidor);
- plantilla de interfaz Admin (rol servidor);
- plantilla de interfaz Link (rol servidor);
- plantilla de interfaz Proxy (rol servidor);
- plantilla de interfaz Offer Iterator (rol servidor);
- plantilla de interfaz Offer Id Iterator (rol servidor).

NOTA – Puede haber muchas instancias de cada una de las interfaces lookup (rol cliente), offer iterator, offer id iterator.

Las implementaciones pueden instanciar otras interfaces de rol cliente, por ejemplo, interfaz depositario de tipos (type repository) o interfaz de almacenamiento (storage). Sin embargo, éstas son funciones ODP separadas y su normalización cae fuera del alcance de esta Especificación.

8.7.2.2 Acciones relativas al estado

El intermediario puede acceder a su estado y modificarlo. Se identifican los siguientes componentes del estado:

- el conjunto de ofertas de servicio;
- el conjunto de ofertas por procuración;
- el conjunto de enlaces;
- el conjunto de atributos de intermediario.

La interfaz Register actualiza las ofertas de servicio.

La interfaz Proxy actualiza las ofertas por procuración.

La interfaz Link actualiza los enlaces.

La interfaz Admin actualiza los atributos de intermediario.

8.7.2.3 Comportamiento de objeto combinado de consulta-registro (lookup-register)

Todas las ofertas de servicio que están en ese momento en el intermediario debido al uso de operaciones en la interfaz Register están disponibles para se retornadas mediante la operación interrogación de la interfaz Lookup.

8.7.2.4 Comportamiento de objeto combinado de consulta-procuración (lookup-proxy)

Todas las ofertas por procuración que están en ese momento en el intermediario debido al uso de operaciones en la interfaz Proxy están disponibles para se retornadas mediante la operación interrogación de la interfaz Lookup.

8.7.2.5 Comportamiento de objeto combinado de administración-consulta (admin-lookup)

El conjunto de ofertas de servicio que pueden ser retornadas es afectado por el valor del atributo support_dynamic_properties fijado mediante las operaciones de la interfaz Admin. Si el atributo support_dynamic_properties se fija al valor FALSE, las operaciones de interrogación posteriores no retornarán ofertas que incluyan propiedades dinámicas.

El conjunto de ofertas por procuración que pueden retornarse es afectado por el valor del atributo support_proxy_offer fijado mediante las operaciones de la interfaz Admin. Si la support_proxy_offer se fija al valor FALSE, las operaciones interrogar posteriores no retornarán ofertas mediante una oferta por procuración, aun si el parámetro de política de importación operación interrogar que tiene el valor use_proxy_offers se fija a TRUE.

El conjunto de ofertas de servicio que pueden retornarse es afectado por el valor del atributo support_modifiable_properties, fijado mediante las operaciones de la interfaz Admin. Si el atributo support_modifiable_properties se fija al valor FALSE, las operaciones de interrogación posteriores no retornarán ofertas que incluyan propiedades modificables.

8.7.2.6 Comportamiento de objeto combinado de administración-registro (admin-register)

La disponibilidad de la operación modificar en la interfaz Register es afectada por el valor del atributo supports_modifiable_properties del intermediario.

La aptitud para exportar ofertas de servicio con propiedades dinámicas es afectada por el valor del atributo supports_dynamic_properties del intermediario. Si el valor del atributo supports_dynamic_properties se fija a FALSE, pueden rechazarse entonces las peticiones de exportación que incluyan propiedades dinámicas.

8.7.2.7 Comportamiento de objeto combinado de administración-procuración (admin-proxy)

Las funciones asociadas con la interfaz Proxy no son afectadas por el valor del atributo supports_proxy_offer. Si el valor del atributo supports_proxy_offer se fija a FALSE, cualesquiera ofertas por procuración dentro del intermediario no se harán disponibles a las operaciones interrogar en la interfaz Lookup.

La aptitud de exportar ofertas por procuración con propiedades dinámicas es afectada por el valor del atributo supports_dynamic_properties del intermediario. Si el valor del atributo supports_dynamic_properties se fija a FALSE, pueden rechazarse entonces las peticiones de exportación que incluyan propiedades dinámicas.

8.7.2.8 Comportamiento de objeto combinado de enlace-consulta (link-lookup)

Todas las ofertas de servicio que están disponibles a través de las interfaces Lookup de los exportadores enlazados serán accesibles recurrentemente mediante una operación interrogar sobre una interfaz Lookup inicial.

8.7.2.9 Comportamiento de objeto combinado de enlace-registro (link-register)

La operación resolver en el registro está sólo disponible si el intermediario está enlazado con otros intermediarios.

8.7.2.10 Acción arbitraje

Una plantilla de acción arbitraje es una plantilla para acciones que combina un argumento criterios (proporcionado en una interfaz) con criterios de intermediario y valores de propiedad (disponibles a partir del propio estado del intermediario). La acción produce un criterio resultante que corresponde a la política (en términos de empresa) para realizar una operación dada.

La acción arbitraje representa algún algoritmo condicional dentro del objeto intermediario. Corresponde a la política de arbitraje de especificación de la empresa.

8.7.2.11 Constricciones sobre la ocurrencia de acciones

El comportamiento de la interfaz Export modifica el espacio de oferta de servicio y las identidades de oferta. El comportamiento de la interfaz Link modifica el espacio de enlace. El comportamiento de la interfaz Admin modifica los atributos del intermediario. Por tanto, no son necesarias constricciones en el entrelazado de acciones a partir de operaciones en una interfaz con acciones a partir de operaciones en la otra interfaz para asegurar la coherencia de los datos. El comportamiento de la interfaz Proxy modifica el espacio de oferta de procuración y los identificadores de oferta.

8.7.3 Constricción de entorno

Puede haber una constricción de entorno de que la primera interfaz del servicio de intermediación que es instanciada esté en una determinada ubicación. Se permitiría así a otros objetos ser instanciados con conocimiento de un intermediario.

No se identifican otras constricciones de entorno.

9 Declaraciones de conformidad y puntos de referencia

Las siguientes interfaces operacionales son puntos de referencia programáticos para probar la conformidad:

- la interfaz Lookup (consulta) (como servidor) proporcionada por la implementación de intermediario sometida a prueba;
- la interfaz Register (registro) (como servidor) proporcionada por la implementación de intermediario sometida a prueba;
- la interfaz Link (enlace) (como servidor) proporcionada por la implementación de intermediario sometida a prueba;
- la interfaz Admin (administración) (como servidor) proporcionada por la implementación de intermediario sometida a prueba;
- la interfaz Proxy (procuración) (como servidor) proporcionada por la implementación de intermediario sometida a prueba;
- una interfaz Lookup (consulta) (como cliente) proporcionada por la implementación de intermediario sometida a prueba;
- una interfaz Register (registro) (como cliente) proporcionada por la implementación de intermediario sometida a prueba;
- una interfaz DynamicPropEval (evaluación de la propiedad dinámica) (como cliente) de un objeto, utilizada por la implementación proporcionada por la implementación de intermediario sometida a prueba durante la evaluación de una propiedad dinámica.

El comportamiento definido para cada una de las operaciones normativas en la especificación computacional se exhibirá en los puntos de conformidad asociados con ese comportamiento.

A fin de alegar conformidad, los implementadores especificarán qué interfaz de ingeniería corresponde a cada una de las interfaces computacionales identificadas como puntos de conformidad.

Además, el implementador especificará qué mecanismo de comunicaciones se utiliza y qué transparencias son proporcionadas sobre qué interfaces.

La siguiente taxonomía se define para clases específicas de conformidad de implementación de las implementaciones de objeto de función de intermediación:

- Intermediario de interrogación (query trader): Puede servir la interfaz Lookup.
- Intermediario simple (simple trader): Puede servir las interfaces Lookup y Register.
- Intermediario independiente (stand-alone trader): Puede servir las interfaces Lookup, Register y Admin.
- Intermediario enlazado (linked trader): Puede servir las interfaces Lookup, Register, Admin y Link; y puede ser cliente de las interfaces Lookup y Register.
- Intermediario de procuración (proxy trader): Puede servir las interfaces Lookup, Register, Admin y Proxy; y puede ser cliente de la interfaz Lookup.
- Intermediario de servicio completo (full-service trader): Puede servir las interfaces Lookup, Register, Admin, Link y Proxy; y puede ser cliente de las interfaces Lookup y Register.

Cualquiera de estas clases de conformidad de objeto de intermediación específicos puede ser también cliente de la interfaz DynamicPropEval si sustenta propiedades dinámicas.

La citada taxonomía da lugar a cinco tipos de objeto específicos de la función de intermediación para alegaciones de conformidad.

9.1 Requisito de conformidad para interfaces de función de intermediación como servidor

Dado que las interfaces a un objeto de intermediación son separables, y la sustentación de estas interfaces es seleccionable, a reserva de las clases de conformidad más arriba definidas, esta subcláusula especifica los requisitos de conformidad para cada interfaz.

9.1.1 Interfaz Lookup

Una implementación que alegue conformidad a la interfaz Lookup como servidor implementará el comportamiento completo asociado con todas las operaciones y atributos de lectura solamente definidos dentro del alcance del tipo de interfaz Lookup.

Una implementación que alegue conformidad con la interfaz Lookup como servidor sustentará también el tipo de interfaz OfferIterator como servidor.

9.1.2 Interfaz Register

Una implementación que alegue conformidad con la interfaz Register como servidor implementará el comportamiento completo asociado con todas las operaciones y atributos de lectura solamente definidos dentro del alcance del tipo de interfaz Register, con las siguientes excepciones permitidas:

- Una implementación que sólo permite el valor FALSE para el atributo supports_modifiable_properties es conforme, en cuyo caso puede rechazar una oferta de servicio que incluya propiedades modificables transmitidas en una operación exportar, y puede también responder a peticiones de operación modificar con una excepción.
- Una implementación que sólo permite el valor FALSE para el atributo supports_dynamic_properties es conforme, en cuyo caso puede rechazar una oferta de servicio que incluya propiedades modificables transmitidas en una operación exportar.
- Una implementación que alegue conformidad con la interfaz Register como servidor, con el valor de supoports_dynamic_properties fijado a TRUE, podrá también asumir el rol cliente para el tipo de interfaz DynamicPropEval.
- Una implementación que alegue conformidad con la interfaz Register como servidor, con el valor del atributo de lectura solamente supports_proxy_offers puesto a TRUE, sustentará también la interfaz Proxy.

9.1.3 Interfaz Link

Una implementación que alegue conformidad con la interfaz Link como servidor implementará el comportamiento completo asociado con todas las operaciones y atributos de lectura solamente definidos dentro del alcance del tipo de interfaz Link.

9.1.4 Interfaz Admin

Una implementación que alegue conformidad con la interfaz Admin como servidor implementará el comportamiento completo asociado con todas las operaciones y atributos de lectura solamente definidos dentro del alcance del tipo de interfaz Admin.

Una implementación que alegue conformidad con la interfaz Admin como servidor sustentará también la interfaz OfferIdIterator como servidor.

9.1.5 Interfaz Proxy

Una implementación que alegue conformidad con la interfaz Proxy como servidor implementará el comportamiento completo asociado con todas las operaciones y definidos dentro del alcance del tipo de interfaz Proxy.

Una implementación que alegue conformidad con la interfaz Proxy como servidor sustentará también la interfaz OfferIdIterator como servidor.

9.1.6 Interfaz OfferIterator

Una implementación que alegue conformidad con la interfaz OfferIterator como servidor implementará el comportamiento completo asociado con todas las operaciones definidas dentro del alcance del tipo de interfaz OfferIterator.

9.1.7 Interfaz OfferIdIterator

Una implementación que alegue conformidad con la interfaz OfferIdIterator como servidor implementará el comportamiento completo asociado con todas las operaciones definidas dentro del alcance del tipo de interfaz OfferIdIterator.

9.1.8 Interfaz DynamicPropEval

Una implementación que alegue conformidad con la interfaz DynamicPropEval como servidor implementará el comportamiento requerido asociado con todas las operaciones definidas dentro del alcance del tipo de interfaz DynamicPropEval.

NOTA – El comportamiento de esta interfaz se limita a la sustentación de la signatura de interfaz.

9.1.9 Conformidad de la regla de subtipificación del servicio

El IDL ODP se utiliza en esta Especificación para expresar signaturas de interfaz de operación computacional. El uso de esta notación no implica el uso de mecanismos y protocolos de sustentación específicos.

Para la conformidad, las reglas de subtipificación de servicio neutras arquitecturales de 8.2.3.2 son la más amplia interpretación de la subtipificación para un intermediario conforme. Una implementación de estas reglas de subtipificación de servicio neutras arquitecturales en un mecanismo y en un protocolo específicos no pueden ser más permisibles que estas reglas de subtipificación de servicio. Un mecanismo y un protocolo específico no pueden sustentar una definición de subtipificación que no es permitida por la función de intermediación ODP.

Las reglas más restrictivas (límite inferior) para la subtipificación de una implementación de intermediario conforme construida en un mecanismo y protocolo de sustentación determinados, autorizadas para que haya conformidad, son:

- Un servicio tipo b, es un subtipo del tipo de servicio a, si y sólo si:
  - el tipo de interfaz de ST b puede añadir operaciones al tipo de interfaz de ST a; y
  - todas las propiedades denominadas de ST a están en ST b; y
  - todas las propiedades denominadas de ST a tienen un tipo que es idéntico al tipo de la propiedad idénticamente denominada en ST b; y
  - todas las propiedades denominadas de ST a tienen un modo que es idéntico al modo de la propiedad idénticamente denominada en ST b.

9.2 Requisitos de conformidad para la clase de conformidad intermediario de interrogación

Una implementación de un sistema de intermediación que alegue conformidad con la clase de conformidad intermediario de interrogación se conformará como servidor a los requisitos de conformidad de la interfaz Lookup como servidor.

Las implementaciones conformes con esta clase de conformidad no deben alegar concordancia con la interfaz Register, y utilizaría medios no normalizados para introducir ofertas de servicio en el intermediario.

9.3 Requisitos de conformidad para la clase de conformidad intermediario simple

Una implementación de un sistema de intermediación que alegue conformidad con la clase de conformidad intermediario simple se conformará a los requisitos de conformidad de los tipos de interfaz Lookup y Register (consulta y registro) como servidor.

Un intermediario simple se conformará al comportamiento de objeto combinado de consulta-registro especificado en 8.7.2.

9.4 Requisitos de conformidad para la clase de conformidad intermediario independiente

Una implementación de un sistema de intermediación que alegue conformidad con la clase de conformidad intermediario independiente, se conformará a los requisitos de los tipos de interfaz Lookup, Register y Admin como servidor.

Un intermediario independiente se conformará al comportamiento de objeto combinado de consulta-registro especificado en 8.7.2.

Un intermediario independiente se conformará al comportamiento de objeto combinado de administración-registro especificado en 8.7.2.

Un intermediario independiente se conformará al comportamiento de objeto combinado de administración-consulta especificado en 8.7.2.

9.5 Requisitos de conformidad para la clase de conformidad intermediario enlazado

Una implementación de un sistema de intermediación que alegue conformidad con la clase de conformidad intermediario enlazado, se conformará a los requisitos de los tipos de interfaz Lookup, Register, Admin y Link; y podrá asumir el rol de cliente para invocar operaciones en el tipo de interfaz Lookup.

Un intermediario enlazado se conformará al comportamiento de objeto combinado de consulta-registro especificado en 8.7.2.

Un intermediario enlazado se conformará al comportamiento de objeto combinado de administración-registro especificado en 8.7.2.

Un intermediario enlazado se conformará al comportamiento de objeto combinado de administración-consulta especificado en 8.7.2.

Un intermediario enlazado se conformará al comportamiento de objeto combinado de consulta-enlace especificado en 8.7.2.

9.6 Requisitos de conformidad para la clase de conformidad intermediario de procuración

Una implementación de un sistema de intermediación que alegue conformidad con la clase de conformidad intermediario de procuración, se conformará como servidor a los requisitos de los tipos de interfaz Lookup, Register, Admin y Proxy; y podrá asumir el rol de cliente para invocar operaciones en el tipo de interfaz Lookup.

Un intermediario de procuración se conformará al comportamiento de objeto combinado de consulta-registro especificado en 8.7.2.

Un intermediario de procuración se conformará al comportamiento de objeto combinado de administración-registro especificado en 8.7.2.

Un intermediario de procuración se conformará al comportamiento de objeto combinado de administración-consulta especificado en 8.7.2.

Un intermediario de procuración se conformará al comportamiento de objeto combinado de consulta-procuración especificado en 8.7.2.

Un intermediario de procuración se conformará al comportamiento de objeto combinado de administración-procuración especificado en 8.7.2.

9.7 Requisitos de conformidad para la clase de conformidad intermediario de servicio completo

Una implementación de un sistema de intermediación que alegue conformidad con la clase de conformidad intermediaria de servicio completo, se conformará como servidor a los requisitos de conformidad de los tipos de interfaz Lookup, Register, Admin, Link y Proxy; y podrá asumir el rol de cliente para invocar operaciones por el tipo de interfaz Lookup.

Un intermediario de servicio completo se conformará al comportamiento de objeto combinado de consulta-registro especificado en 8.7.2.

Un intermediario de servicio completo se conformará al comportamiento de objeto combinado de administración-registro especificado en 8.7.2.

Un intermediario de servicio completo se conformará al comportamiento de objeto combinado de administración-consulta especificado en 8.7.2.

Un intermediario de servicio completo se conformará al comportamiento de objeto combinado de consulta-enlace especificado en 8.7.2.

Un intermediario de servicio completo se conformará al comportamiento de objeto combinado de consulta-procuración especificado en 8.7.2.

Un intermediario de servicio completo se conformará al comportamiento de objeto combinado de administración-procuración especificado en 8.7.2.

9.8 Pruebas de conformidad

Para cada prueba de conformidad, la implementación identificará los puntos de referencia programáticos para los que se alega conformidad.

Para cada prueba de conformidad, el implementador especificará:

- qué política está en vigor mientras dura la prueba;
- qué estado de objeto se supone, con respecto a la especificación de información, como esquema estático, por ejemplo, ofertas de servicio mantenidas.

Una implementación que alegue conformidad ha de proporcionar el conjunto de políticas o circunstancias en las que una oferta exportada pueda ser posteriormente recuperada por una operación interrogar.

Las implementaciones que aleguen conformidad con las clases de conformidad intermediario enlazado o intermediario de servicio completo deberán poder demostrar la aptitud para propagar una operación interrogar a una interfaz Lookup de intermediario distante.

Las implementaciones que aleguen conformidad con las clases de conformidad intermediario de procuración o intermediario de servicio completo podrán demostrar la aptitud para remitir una operación interrogar a un objeto distante mediante una oferta de procuración.

Anexo A

Interfaz de definición de lenguaje para ODP basada en la especificación de la función intermediación

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

A.1 Introduction

This annex provides the ODP-IDL (see ITU-T Rec. X.920 | ISO/IEC 14750) specification of the interface signature for the trading function's computational specification. It specifies the signature for each computational operation in ODP-IDL, according to the functional description (signature and semantics) provided in clause 8.

If there are any discrepancies between the ODP-IDL specifications in this annex and those in Clause 8, the specifications in this annex take precedence.

ODP-IDL is used in this Specification to express computational operation interface signatures. Use of this notation does not imply use of specific supporting mechanisms and protocols.

NOTE – The templates in this annex are technically aligned with the templates of the OMG Trading Object Service.

A.2 ODP Trading Function module

```
module CosTrading {

        // forward references to our interfaces

        interface Lookup;
        interface Register;
        interface Link;
        interface Proxy;
        interface Admin;
        interface OfferIterator;
        interface OfferIdIterator;

        // type definitions used in more than one interface

        typedef string Istring;
        typedef Object TypeRepository;

        typedef Istring PropertyName;
        typedef sequence<PropertyName> PropertyNameSeq;
        typedef any PropertyValue;
        struct Property {
              PropertyName name;
              PropertyValue value;
        };
        typedef sequence<Property> PropertySeq;

        struct Offer {
              Object reference;
              PropertySeq properties;
        };
        typedef sequence<Offer> OfferSeq;
        typedef string OfferId;
        typedef sequence<OfferId> OfferIdSeq;

        typedef Istring ServiceTypeName;  // similar structure to IR::Identifier

        typedef Istring Constraint;

        enum FollowOption {
              local_only,
              if_no_local,
              always
};
```

```
typedef Istring LinkName;
typedef sequence<LinkName> LinkNameSeq;
typedef LinkNameSeq TraderName;

typedef string PolicyName;  // policy names restricted to Latin1
typedef sequence<PolicyName> PolicyNameSeq;
typedef any PolicyValue;
struct Policy {
        PolicyName name;
        PolicyValue value;
};
typedef sequence<Policy> PolicySeq;

// exceptions used in more than one interface

exception UnknownMaxLeft {};

exception NotImplemented {};

exception IllegalServiceType {
        ServiceTypeName type;
};

exception UnknownServiceType {
        ServiceTypeName type;
};

exception IllegalPropertyName {
        PropertyName name;
};

exception DuplicatePropertyName {
        PropertyName name;
};

exception PropertyTypeMismatch {
        ServiceTypeName type;
        Property prop;
};

exception MissingMandatoryProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception ReadonlyDynamicProperty {
        ServiceTypeName type;
        PropertyName name;
};

exception IllegalConstraint {
        Constraint constr;
};

exception InvalidLookupRef {
        Lookup target;
};

exception IllegalOfferId {
        OfferId id;
};

exception UnknownOfferId {
        OfferId id;
};

exception DuplicatePolicyName {
        PolicyName name;
};

// the interfaces
```

```
interface TraderComponents {

    readonly attribute Lookup lookup_if;
    readonly attribute Register register_if;
    readonly attribute Link link_if;
    readonly attribute Proxy proxy_if;
    readonly attribute Admin admin_if;
};

interface SupportAttributes {

    readonly attribute boolean supports_modifiable_properties;
    readonly attribute boolean supports_dynamic_properties;
    readonly attribute boolean supports_proxy_offers;
    readonly attribute TypeRepository type_repos;
};

interface ImportAttributes {

    readonly attribute unsigned long def_search_card;
    readonly attribute unsigned long max_search_card;
    readonly attribute unsigned long def_match_card;
    readonly attribute unsigned long max_match_card;
    readonly attribute unsigned long def_return_card;
    readonly attribute unsigned long max_return_card;
    readonly attribute unsigned long max_list;
    readonly attribute unsigned long def_hop_count;
    readonly attribute unsigned long max_hop_count;
    readonly attribute FollowOption def_follow_policy;
    readonly attribute FollowOption max_follow_policy;
};

interface LinkAttributes {

    readonly attribute FollowOption max_link_follow_policy;
};

interface Lookup : TraderComponents, SupportAttributes, ImportAttributes {

    typedef Istring Preference;

    enum HowManyProps { none, some, all };

    union SpecifiedProps switch ( HowManyProps ) {
        case some: PropertyNameSeq prop_names;
    };

    exception IllegalPreference {
        Preference pref;
    };

    exception IllegalPolicyName {
        PolicyName name;
    };

    exception PolicyTypeMismatch {
        Policy the_policy;
    };

    exception InvalidPolicyValue {
        Policy the_policy;
    };

    void query (
        in ServiceTypeName type,
        in Constraint constr,
        in Preference pref,
        in PolicySeq policies,
        in SpecifiedProps desired_props,
        in unsigned long how_many,
        out OfferSeq offers,
        out OfferIterator offer_itr,
        out PolicyNameSeq limits_applied
```

```
        ) raises (
                IllegalServiceType,
                UnknownServiceType,
                IllegalConstraint,
                IllegalPreference,
                IllegalPolicyName,
                PolicyTypeMismatch,
                InvalidPolicyValue,
                IllegalPropertyName,
                DuplicatePropertyName,
                DuplicatePolicyName
        );
};

interface Register : TraderComponents, SupportAttributes {

        struct OfferInfo {
                Object reference;
                ServiceTypeName type;
                PropertySeq properties;
        };

        exception InvalidObjectRef {
                Object ref;
        };

        exception UnknownPropertyName {
                PropertyName name;
        };

        exception InterfaceTypeMismatch {
                ServiceTypeName type;
                Object reference;
        };

        exception ProxyOfferId {
                OfferId id;
        };

        exception MandatoryProperty {
                ServiceTypeName type;
                PropertyName name;
        };

        exception ReadonlyProperty {
                ServiceTypeName type;
                PropertyName name;
        };

        exception NoMatchingOffers {
                Constraint constr;
        };

        exception IllegalTraderName {
                TraderName name;
        };

        exception UnknownTraderName {
                TraderName name;
        };

        exception RegisterNotSupported {
                TraderName name;
        };

        OfferId export (
                in Object reference,
                in ServiceTypeName type,
                in PropertySeq properties
        ) raises (
                InvalidObjectRef,
                IllegalServiceType,
                UnknownServiceType,
                InterfaceTypeMismatch,
```

```
                IllegalPropertyName, // e.g. prop_name = "<foo-bar"
                PropertyTypeMismatch,
                ReadonlyDynamicProperty,
                MissingMandatoryProperty,
                DuplicatePropertyName
        );

        void withdraw (
                in OfferId id
        ) raises (
                IllegalOfferId,
                UnknownOfferId,
                ProxyOfferId
        );

        OfferInfo describe (
                in OfferId id
        ) raises (
                IllegalOfferId,
                UnknownOfferId,
                ProxyOfferId
        );

        void modify (
                in OfferId id,
                in PropertyNameSeq del_list,
                in PropertySeq modify_list
        ) raises (
                NotImplemented,
                IllegalOfferId,
                UnknownOfferId,
                ProxyOfferId,
                IllegalPropertyName,
                UnknownPropertyName,
                PropertyTypeMismatch,
                ReadonlyDynamicProperty,
                MandatoryProperty,
                ReadonlyProperty,
                DuplicatePropertyName
        );

        void withdraw_using_constraint (
                in ServiceTypeName type,
                in Constraint constr
        ) raises (
                IllegalServiceType,
                UnknownServiceType,
                IllegalConstraint,
                NoMatchingOffers
        );

        Register resolve (
                in TraderName name
        ) raises (
                IllegalTraderName,
                UnknownTraderName,
                RegisterNotSupported
        );
};

interface Link : TraderComponents, SupportAttributes, LinkAttributes {

        struct LinkInfo {
                Lookup target;
                Register target_reg;
                FollowOption def_pass_on_follow_rule;
                FollowOption limiting_follow_rule;
        };

        exception IllegalLinkName {
                LinkName name;
        };
```

```
        exception UnknownLinkName {
                LinkName name;
        };

        exception DuplicateLinkName {
                LinkName name;
        };

        exception DefaultFollowTooPermissive {
                FollowOption def_pass_on_follow_rule;
                FollowOption limiting_follow_rule;
        };

        exception LimitingFollowTooPermissive {
                FollowOption limiting_follow_rule;
                FollowOption max_link_follow_policy;
        };

        void add_link (
                in LinkName name,
                in Lookup target,
                in FollowOption def_pass_on_follow_rule,
                in FollowOption limiting_follow_rule
        ) raises (
                IllegalLinkName,
                DuplicateLinkName,
                InvalidLookupRef, // e.g. nil
                DefaultFollowTooPermissive,
                LimitingFollowTooPermissive
        );

        void remove_link (
                in LinkName name
        ) raises (
                IllegalLinkName,
                UnknownLinkName
        );

        LinkInfo describe_link (
                in LinkName name
        ) raises (
                IllegalLinkName,
                UnknownLinkName
        );

        LinkNameSeq list_links ( );

        void modify_link (
                in LinkName name,
                in FollowOption def_pass_on_follow_rule,
                in FollowOption limiting_follow_rule
        ) raises (
                IllegalLinkName,
                UnknownLinkName,
                DefaultFollowTooPermissive,
                LimitingFollowTooPermissive
        );
};

interface Proxy : TraderComponents, SupportAttributes {

        typedef Istring ConstraintRecipe;

        struct ProxyInfo {
                ServiceTypeName type;
                Lookup target;
                PropertySeq properties;
                boolean if_match_all;
                ConstraintRecipe recipe;
                PolicySeq policies_to_pass_on;
        };
```

```
        exception IllegalRecipe {
            ConstraintRecipe recipe;
        };

        exception NotProxyOfferId {
            OfferId id;
        };

        OfferId export_proxy (
            in Lookup target,
            in ServiceTypeName type,
            in PropertySeq properties,
            in boolean if_match_all,
            in ConstraintRecipe recipe,
            in PolicySeq policies_to_pass_on
        ) raises (
            IllegalServiceType,
            UnknownServiceType,
            InvalidLookupRef, // e.g. nil
            IllegalPropertyName,
            PropertyTypeMismatch,
            ReadonlyDynamicProperty,
            MissingMandatoryProperty,
            IllegalRecipe,
            DuplicatePropertyName,
            DuplicatePolicyName
        );

        void withdraw_proxy (
            in OfferId id
        ) raises (
            IllegalOfferId,
            UnknownOfferId,
            NotProxyOfferId
        );

        ProxyInfo describe_proxy (
            in OfferId id
        ) raises (
            IllegalOfferId,
            UnknownOfferId,
            NotProxyOfferId
        );
    };

    interface Admin : TraderComponents, SupportAttributes, ImportAttributes,
                                LinkAttributes {
        typedef sequence<octet> OctetSeq;

        readonly attribute OctetSeq request_id_stem;

        unsigned long set_def_search_card (in unsigned long value);
        unsigned long set_max_search_card (in unsigned long value);

        unsigned long set_def_match_card (in unsigned long value);
        unsigned long set_max_match_card (in unsigned long value);

        unsigned long set_def_return_card (in unsigned long value);
        unsigned long set_max_return_card (in unsigned long value);

        unsigned long set_max_list (in unsigned long value);

        boolean set_supports_modifiable_properties (in boolean value);
        boolean set_supports_dynamic_properties (in boolean value);
        boolean set_supports_proxy_offers (in boolean value);

        unsigned long set_def_hop_count (in unsigned long value);
        unsigned long set_max_hop_count (in unsigned long value);

        FollowOption set_def_follow_policy (in FollowOption policy);
        FollowOption set_max_follow_policy (in FollowOption policy);
```

```
        FollowOption set_max_link_follow_policy (in FollowOption policy);

        TypeRepository set_type_repos (in TypeRepository repository);

        OctetSeq set_request_id_stem (in OctetSeq stem);

        void list_offers (
            in unsigned long how_many,
            out OfferIdSeq ids,
            out OfferIdIterator id_itr
        ) raises (
            NotImplemented
        );

        void list_proxies (
            in unsigned long how_many,
            out OfferIdSeq ids,
            out OfferIdIterator id_itr
        ) raises (
            NotImplemented
        );
    };

    interface OfferIterator {

        unsigned long max_left (
        ) raises (
            UnknownMaxLeft
        );

        boolean next_n (
            in unsigned long n,
            out OfferSeq offers
        );

        void destroy ();
    };

    interface OfferIdIterator {

        unsigned long max_left (
        ) raises (
            UnknownMaxLeft
        );
        boolean next_n (
            in unsigned long n,
            out OfferIdSeq ids
        );

        void destroy ();
    };

}; /* end module CosTrading */
```

A.3 Dynamic Property module

When implementing this ODP Function using a particular ODP infrastructure, the term "TypeCode" must be defined using an appropriate type for representing type descriptions in that ODP infrastructure.

NOTE – For CORBA infrastructure, the term "TypeCode" should be defined as "CORBA::TypeCode".

```
module CosTradingDynamic {

    exception DPEvalFailure {
        CosTrading::PropertyName name;
        TypeCode returned_type;
        any extra_info;
    };
```

```
        interface DynamicPropEval {

            any evalDP (
                in CosTrading::PropertyName name,
                in TypeCode returned_type,
                in any extra_info
            ) raises (
                DPEvalFailure
            );
        };

        struct DynamicProp {
            DynamicPropEval eval_if;
            TypeCode returned_type;
            any extra_info;
        };
}; /* end module CosTradingDynamic */
```

Anexo B

BNF de lenguaje de constricciones de la función intermediación ODP

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

B.1 Introduction

This annex provides the BNF specification of the ODP Trading Function standard constraint language. It is used for specifying both the constraint and preference expression parameters to various operations in the trader interfaces.

NOTE – This annex is technically aligned with the OMG Trading Object Service.

A statement in this language is an Istring. Other constraint languages may be supported by a particular trader implementation; the constraint language used by a client of the trader is indicated by embedding "<<Identifier major.minor>>" at the beginning of the string. For this purpose, the name of the constraint language specified in this annex is "OMG 1.0". If such an escape is not used, the constraint language in this annex is the default (i.e, it is equivalent to embedding "<<OMG 1.0>>" at the beginning of the string).

B.2 Language basics

B.2.1 Basic elements

Both the constraint and preference expressions in a query can be constructed from property names of conformant offers and literals. The constraint language in which these expressions are written consists of the following items (examples of these expressions are shown in square brackets below each bulleted item):

– comparative functions: == (equality), != (inequality), >, >=, <, <=, ~ (substring match), in (element in sequence); the result of applying a comparative function is a boolean value:

  ["Cost < 5" implies only consider offers with a Cost property value less than 5; "'Visa' in CreditCards" implies only consider offers in which the CreditCards property, consisting of a set of strings, contains the string 'Visa'];

– boolean connectives: and, or, not:

  ["Cost >= 2 and Cost <= 5" implies only consider offers where the value of the Cost property is in the range 2 <= Cost <= 5];

– property existence: exist;

– property names;

– numeric and string constants;

– mathematical operators: +, –, *, /:

  ["10 < 12.3 * MemSize + 4.6 * FileSize" implies only consider offers for which the arithmetic function in terms of the value of the MemSize and FileSize properties exceeds 10];

– grouping operators: (,).

Note that the keywords in the language are case sensitive.

B.2.2 Precedence relations

The following precedence relations hold in the absence of parentheses, in the order of highest to lowest:

( ) exist unary-minus

not

*** /**

+ –

~

in

== != < <= > >=

and

or

B.2.3 Legal property value types

While one can define properties of service types with arbitrarily complex ODP-IDL value types, only the following property value types can be manipulated using the constraint language:

– boolean, short, unsigned short, long, unsigned long, float, double, char, Ichar, string, Istring;

– sequences of the above types.

The "exist" operator can be applied to any property name, regardless of the property's value type.

B.2.4 Operator restrictions

exist can be applied to any property;

~ can only be applied if left operand and right operand are both strings or both Istrings;

in can only be applied if the left operand is one of the simple types described above and the right operand is a sequence of the same simple type;

== can only be applied if the left and right operands are of the same simple type;

!= can only be applied if the left and right operands are of the same simple type;

< can only be applied if the left and right operands are of the same simple type;

<= can only be applied if the left and right operands are of the same simple type;

> can only be applied if the left and right operands are of the same simple type;

>= can only be applied if the left and right operands are of the same simple type;

+ can only be applied to simple numeric operands;

– can only be applied to simple numeric operands;

* can only be applied to simple numeric operands;

/ can only be applied to simple numeric operands.

<, <=, >, >= comparisons imply use of the appropriate collating sequence for characters and strings; TRUE is greater than FALSE for booleans.

B.2.5 Representation of literals

– boolean TRUE or FALSE;

– integers sequences of digits, with a possible leading + or –;

– floats digits with decimal point, with optional exponential notation;

– characters char and Ichar are of the form '<char>', string and Istring are of the form '<char><char>+'; to embed an apostrophe in a string, place a backslash (\) in front of it; to embed a backslash in a string, use \\.

B.3 The Constraint Language BNF

The Constraint Language Proper in Terms of Lexical Tokens

```
<constraint>   :=  /* empty */
               |   <bool>

<preference>   :=  /* <empty> */
               |   min <bool>
               |   max <bool>
               |   with <bool>
               |   random
               |   first
<bool>         :=  <bool_or>

<bool_or>      :=  <bool_or> or <bool_and>
               |   <bool_and>

<bool_and>     :=  <bool_and> and <bool_compare>
               |   <bool_compare>
```

```
<bool_compare>      :=  <expr_in> == <expr_in>
                    |   <expr_in> != <expr_in>
                    |   <expr_in> <  <expr_in>
                    |   <expr_in> <= <expr_in>
                    |   <expr_in> >  <expr_in>
                    |   <expr_in> >= <expr_in>
                    |   <expr_in>

        <expr_in>       :=  <expr_twiddle> in <Ident>
                    |   <expr_twiddle>

<expr_twiddle>      :=  <expr> ~ <expr>
                    |   <expr>

            <expr>  :=  <expr> + <term>
                    |   <expr> – <term>
                    |   <term>

            <term>  :=  <term> * <factor_not>
                    |   <term> / <factor_not>
                    |   <factor_not>

        <factor_not>:=  not <factor>
                    |   <factor>

        <factor>    :=  ( <bool_or> )
                    |   exist <Ident>
                    |   <Ident>
                    |   <Number>
                    |   – <Number>
                    |   <String>
                    |   TRUE
                    |   FALSE
```

B.3.2 “BNF” for Lexical Tokens up to Character Set Issues

```
        <Ident>     :=  <Leader> <FollowSeq>

        <FollowSeq> :=  /* <empty> */
                    |   <FollowSeq> <Follow>

        <Number>    :=  <Mantissa>
                    |   <Mantissa> <Exponent>

        <Mantissa>  :=  <Digits>
                    |   <Digits> .
                    |   . <Digits>
                    |   <Digits> . <Digits>

        <Exponent>  :=  <Exp> <Sign> <Digits>

        <Sign>      :=  +
                    |   –

        <Exp>       :=  E
                    |   e

        <Digits>    :=  <Digits> <Digit>
                    |   <Digit>
        <String>    :=  ' <TextChars> '

        <TextChars> :=  /* <empty> */
                    |   <TextChars> <TextChar>

        <TextChar>  :=  <Alpha>
                    |   <Digit>
                    |   <Other>
                    |   <Special>

        <Special>   :=  \\
                    |   \'
```

B.3.3 Character Set Issues

The previous BNF has been complete up to the non-terminals <Leader>, <Follow>, <Alpha>, <Digit>, and <Other>. For a particular character set, one must define the characters which make up these character classes.

Each character set which the trading service is to support must define these character classes.

The character classes for the ISO Latin-1 character set are:

<Leader> := <Alpha>

<Follow> := <Alpha>
| <Digit>
| _

<Alpha> is the set of alphabetic characters(letters), as defined in ISO/IEC 14750
<Digit> is the set of digits [0-9]
<Other> is the set of ASCII characters that are not <Alpha>, <Digit>, or <Special>

Anexo C

Lenguaje de recetas de constricciones de la función intermediación ODP

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

C.1 Introduction

This annex describes the recipe language used to construct the secondary constraint expression when resolving proxy offers; the secondary constraint expression is constructed from the primary constraint expression and the properties associated with the proxy offer.

A statement in this language is an Istring. Other recipe languages may be supported by a particular trader implementation; the recipe language used by a client of the trader is indicated by embedding "<<Identifier major.minor>>" at the beginning of the string. For this purpose, the name of the constraint recipe language specified in this annex is "OMG 1.0". If such an escape is not used, the constraint recipe language in this annex is the default (i.e, it is equivalent to embedding "<<OMG 1.0>>" at the beginning of the string).

While the nested invocation of the Trader behind the proxy assumes support for the Lookup interface, the secondary constraint expression does not necessarily need to conform to the language described in Annex B.

C.2 The recipe syntax

The rewriting from primary to secondary works similarly to formatted output in a variety of programming languages and systems.

NOTE – This syntax is patterned after the variable replacement syntax of the Bourne and Korn shells on most UNIX systems.

When it is time to construct the secondary constraint expression from the recipe, the algorithm is as follows:

```
while not end of recipe
    fetch the next character from the recipe
    if not a '$' character
        append the character to the secondary constraint
    else
        fetch next character from the recipe
        if a '*' character
            append the entire primary constraint to the secondary constraint
        else if not a '(' character
            append the character to the secondary constraint
        else
            collect characters up to a ')' character, discarding ')'
            lookup property with that name
            append formatted value of that property to secondary constraint
```

C.3 Example

Assume a proxy offer has been exported to a trader with the following properties:

<Name, 'MyName'>, <Cost, 42>, <Host, 'x.y.co.uk'>

and with the following recipe:

"Name == $(Name) and Cost == $$$(Cost)"

The above algorithm will generate the following secondary constraint for the nested call to the trader behind the proxy:

"Name == 'MyName' and Cost== $42"

Anexo D

Depositario de tipos de servicio

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

D.1 Introduction

This annex contains a computational specification of a service type repository interface that is suitable for the needs of the ODP Trading Function. Other interfaces for service type repository may be used by trading system implementations.

NOTE 1 – This annex is technically aligned with the Service Type Repository interface of the OMG Trading Object Service. Upon its publication, the ODP Type Repository Function (work in progress at the time of publication of this Specification) would be the preferred service type repository for use by the ODP Trading Function.

When implementing this ODP Function using a particular ODP infrastructure, the term "TypeCode" must be defined using an appropriate type for representing type descriptions in that ODP infrastructure.

NOTE 2 – For CORBA infrastructure, the term "TypeCode" should be defined as "CORBA::TypeCode".

D.2 Service type repository

```
module CosTradingRepos {

    interface ServiceTypeRepository {

    // local types
        typedef sequence<CosTrading::ServiceTypeName> ServiceTypeNameSeq;
        enum PropertyMode {
            PROP_NORMAL, PROP_READONLY,
            PROP_MANDATORY, PROP_MANDATORY_READONLY
        };
        struct PropStruct {
            CosTrading::PropertyName name;
            TypeCode value_type;
            PropertyMode mode;
        };
        typedef sequence<PropStruct> PropStructSeq;

        typedef CosTrading::Istring Identifier; // IR::Identifier
        struct IncarnationNumber {
            unsigned long high;
            unsigned long low;
        };
        struct TypeStruct {
            Identifier if_name;
            PropStructSeq props;
            ServiceTypeNameSeq super_types;
            boolean masked;
            IncarnationNumber incarnation;
        };

        enum ListOption { all, since };
        union SpecifiedServiceTypes switch ( ListOption ) {
            case since: IncarnationNumber incarnation;
        };

    // local exceptions
        exception ServiceTypeExists {
            CosTrading::ServiceTypeName name;
        };
        exception InterfaceTypeMismatch {
            CosTrading::ServiceTypeName base_service;
            Identifier base_if;
            CosTrading::ServiceTypeName derived_service;
            Identifier derived_if;
        };
        exception HasSubTypes {
            CosTrading::ServiceTypeName the_type;
            CosTrading::ServiceTypeName sub_type;
        };
```

```
        exception AlreadyMasked {
                CosTrading::ServiceTypeName name;
        };
        exception NotMasked {
                CosTrading::ServiceTypeName name;
        };
        exception ValueTypeRedefinition {
                CosTrading::ServiceTypeName type_1;
                PropStruct definition_1;
                CosTrading::ServiceTypeName type_2;
                PropStruct definition_2;
        };
        exception DuplicateServiceTypeName {
                CosTrading::ServiceTypeName name;
        };

// attributes
        readonly attribute IncarnationNumber incarnation;

// operation signatures
        IncarnationNumber add_type (
                in CosTrading::ServiceTypeName name,
                in Identifier if_name,
                in PropStructSeq props,
                in ServiceTypeNameSeq super_types
        ) raises (
                CosTrading::IllegalServiceType,
                ServiceTypeExists,
                InterfaceTypeMismatch,
                CosTrading::IllegalPropertyName,
                CosTrading::DuplicatePropertyName,
                ValueTypeRedefinition,
                CosTrading::UnknownServiceType,
                DuplicateServiceTypeName
        );

        void remove_type (
                in CosTrading::ServiceTypeName name
        ) raises (
                CosTrading::IllegalServiceType,
                CosTrading::UnknownServiceType,
                HasSubTypes
        );

        ServiceTypeNameSeq list_types (
                in SpecifiedServiceTypes which_types
        );

        TypeStruct describe_type (
                in CosTrading::ServiceTypeName name
        ) raises (
                CosTrading::IllegalServiceType,
                CosTrading::UnknownServiceType
        );

        TypeStruct fully_describe_type (
                in CosTrading::ServiceTypeName name
        ) raises (
                CosTrading::IllegalServiceType,
                CosTrading::UnknownServiceType
        );

        void mask_type (
                in CosTrading::ServiceTypeName name
        ) raises (
                CosTrading::IllegalServiceType,
                CosTrading::UnknownServiceType,
                AlreadyMasked
        );
```

```
            void unmask_type (
                in CosTrading::ServiceTypeName name
            ) raises (
                CosTrading::IllegalServiceType,
                CosTrading::UnknownServiceType,
                NotMasked
            );

        };
}; /* end module CosTradingRepos */
```

D.2.1 Add Type Operation

D.2.1.1 Signature

```
            IncarnationNumber add_type (
                in CosTrading::ServiceTypeName name,
                in Identifier if_name,
                in PropStructSeq props,
                in ServiceTypeNameSeq super_types
            ) raises (
                CosTrading::IllegalServiceType,
                ServiceTypeExists,
                InterfaceTypeMismatch,
                CosTrading::IllegalPropertyName,
                CosTrading::DuplicatePropertyName,
                ValueTypeRedefinition,
                CosTrading::UnknownServiceType,
                DuplicateServiceTypeName
            );
```

D.2.1.2 Function

The add_type operation enables the creation of new service types in the service type repository. The caller supplies the "name" for the new type, the identifier for the interface associated with instances of this service type, the properties definitions for this service type, and the service type names of the immediate super-types to this service type.

If the type creation is successful, an incarnation number is returned as the value of the operation. Incarnation numbers are opaque values that are assigned to each modification to the repository's state. An incarnation number can be quoted when invoking the list_types operation to retrieve all changes to the service repository since a particular logical time.

NOTE – IncarnationNumber is currently declared as a struct consisting of two unsigned longs; what we really want here is an unsigned hyper (64-bit integer). A future revision could modify this when CORBA systems support ODP-IDL 64-bit integers.

If the "name" parameter is malformed, then the CosTrading::IllegalServiceType exception is raised. If the type already exists, then the ServiceTypeExists exception is raised.

If the "if_name" parameter is not a subtype of the interface associated with a service type from which this service type is derived, such that substitutability would be violated, then the InterfaceTypeMismatch exception is raised.

If a property name supplied in the "props" parameter is malformed, the CosTrading::IllegalPropertyName exception is raised. If the same property name appears two or more times in the "props" parameter, the CosTrading::DuplicatePropertyName exception is raised.

If a property value type associated with this service type illegally modifies the value type of a supertype's property, or if two supertypes incompatibly declare value types for the same property name, then the ValueTypeRedefinition exception is raised.

If one of the ServiceTypeNames in "super_types" is malformed, then the CosTrading::IllegalServiceType exception is raised. If one of the ServiceTypeNames in "super_types" does not exist, then the CosTrading::UnknownServiceType exception is raised. If the same service type name is included two or more times in this parameter the DuplicateServiceTypeName exception is raised.

D.2.2 Remove Type operation

D.2.2.1 Signature

```
            void remove_type (
                in CosTrading::ServiceTypeName name
```

```
) raises (
        CosTrading::IllegalServiceType,
        CosTrading::UnknownServiceType,
        HasSubTypes
);
```

D.2.2.2 Function

The remove_type operation removes the named type from the service type repository.

If “name” is malformed, then the CosTrading::IllegalServiceType exception is raised. If “name” does not exist within the repository, then the CosTrading::UnknownServiceType exception is raised. If “name” has a service type which has been derived from it, then the HasSubTypes exception is raised.

D.2.3 List Types operation

D.2.3.1 Signature

```
ServiceTypeNameSeq list_types (
        in SpecifiedServiceTypes which_types
);
```

D.2.3.2 Function

The list_types operation permits a client to obtain the names of service types which are in the repository. The “which_types” parameter permits the client to specify one of two possible values:

- all types known to the repository;
- all types added/modified since a particular incarnation number.

The names of the requested types are returned by the operation for subsequent querying via the describe_type or the fully_describe_type operation.

D.2.4 Describe Type operation

D.2.4.1 Signature

```
TypeStruct describe_type (
        in CosTrading::ServiceTypeName name
) raises (
        CosTrading::IllegalServiceType,
        CosTrading::UnknownServiceType
);
```

D.2.4.2 Function

The describe_type operation permits a client to obtain the details for a particular service type.

If “name” is malformed, then the CosTrading::IllegalServiceType exception is raised. If “name” does not exist within the repository, then the CosTrading::UnknownServiceType exception is raised.

D.2.5 Fully Describe Type operation

D.2.5.1 Signature

```
TypeStruct fully_describe_type (
        in CosTrading::ServiceTypeName name
) raises (
        CosTrading::IllegalServiceType,
        CosTrading::UnknownServiceType
);
```

D.2.5.2 Function

The fully_describe_type operation permits a client to obtain the details for a particular service type; the property sequence returned in the TypeStruct includes all properties inherited from the transitive closure of its supertypes; the sequence of supertypes in the TypeStruct contains the names of the types in the transitive closure of the supertype relation.

If “name” is malformed, then the CosTrading::IllegalServiceType exception is raised. If “name” does not exist within the repository, then the CosTrading::UnknownServiceType exception is raised.

D.2.6 Mask Type operation

D.2.6.1 Signature

```
void mask_type (
        in CosTrading::ServiceTypeName name
) raises (
        CosTrading::IllegalServiceType,
        CosTrading::UnknownServiceType,
        AlreadyMasked
);
```

D.2.6.2 Function

The mask_type operation permits the deprecation of a particular type, i.e. after being masked, exporters will no longer be able to advertise offers of that particular type. The type continues to exist in the service repository due to other service types being derived from it.

If “name” is malformed, then the CosTrading::IllegalServiceType exception is raised. If “name” does not exist within the repository, then the CosTrading::UnknownServiceType exception is raised. If the type is currently in the masked state, then the AlreadyMasked exception is raised.

D.2.7 Unmask Type operation

D.2.7.1 Signature

```
void unmask_type (
        in CosTrading::ServiceTypeName name
) raises (
        CosTrading::IllegalServiceType,
        CosTrading::UnknownServiceType,
        NotMasked
);
```

D.2.7.2 Function

The unmask_type undeprecates a type, i.e. after being unmasked, exporters will be able to resume advertisement of offers of that particular type.

If “name” is malformed, then the CosTrading::IllegalServiceType exception is raised. If “name” does not exist within the repository, then the CosTrading::UnknownServiceType exception is raised. If the type is not currently in the masked state, then the NotMasked exception is raised.

SERIES DE RECOMENDACIONES DEL UIT-T

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