UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

# UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

# X.950

(08/97)

SÉRIE X: RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

Traitement réparti ouvert

---

# Technologies de l'information – Traitement réparti ouvert – Fonction de courtage: spécification

Recommandation UIT-T X.950

(Antérieurement Recommandation du CCITT)

---

RECOMMANDATIONS UIT-T DE LA SÉRIE X

**RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS**

| | |
|---|---|
| RÉSEAUX PUBLICS POUR DONNÉES | |
| Services et fonctionnalités | X.1–X.19 |
| Interfaces | X.20–X.49 |
| Transmission, signalisation et commutation | X.50–X.89 |
| Aspects réseau | X.90–X.149 |
| Maintenance | X.150–X.179 |
| Dispositions administratives | X.180–X.199 |
| INTERCONNEXION DES SYSTÈMES OUVERTS | |
| Modèle et notation | X.200–X.209 |
| Définitions des services | X.210–X.219 |
| Spécifications des protocoles en mode connexion | X.220–X.229 |
| Spécifications des protocoles en mode sans connexion | X.230–X.239 |
| Formulaires PICS | X.240–X.259 |
| Identification des protocoles | X.260–X.269 |
| Protocoles de sécurité | X.270–X.279 |
| Objets gérés de couche | X.280–X.289 |
| Tests de conformité | X.290–X.299 |
| INTERFONCTIONNEMENT DES RÉSEAUX | |
| Généralités | X.300–X.349 |
| Systèmes de transmission de données par satellite | X.350–X.399 |
| SYSTÈMES DE MESSAGERIE | X.400–X.499 |
| ANNUAIRE | X.500–X.599 |
| RÉSEAUTAGE OSI ET ASPECTS SYSTÈMES | |
| Réseautage | X.600–X.629 |
| Efficacité | X.630–X.639 |
| Qualité de service | X.640–X.649 |
| Dénomination, adressage et enregistrement | X.650–X.679 |
| Notation de syntaxe abstraite numéro un (ASN.1) | X.680–X.699 |
| GESTION OSI | |
| Cadre général et architecture de la gestion-systèmes | X.700–X.709 |
| Service et protocole de communication de gestion | X.710–X.719 |
| Structure de l'information de gestion | X.720–X.729 |
| Fonctions de gestion et fonctions ODMA | X.730–X.799 |
| SÉCURITÉ | X.800–X.849 |
| APPLICATIONS OSI | |
| Engagement, concomitance et rétablissement | X.850–X.859 |
| Traitement transactionnel | X.860–X.879 |
| Opérations distantes | X.880–X.899 |
| **TRAITEMENT RÉPARTI OUVERT** | **X.900–X.999** |

*Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.*

**NORME INTERNATIONALE 13235-1**

**RECOMMANDATION UIT-T X.950**

# TECHNOLOGIES DE L'INFORMATION – TRAITEMENT RÉPARTI OUVERT – FONCTION DE COURTAGE: SPÉCIFICATION


## Résumé

La présente Recommandation | Norme internationale spécifie la fonction de courtage du traitement réparti ouvert (ODP, *open distributed processing*). Cette fonction fait partie des éléments constituants de l'architecture du modèle de référence ODP définie dans la Rec. UIT-T X.903 | ISO/CEI 10746-3. La fonction de courtage permet d'offrir des services dans un système de traitement ODP et de découvrir les services qui y ont été offerts. Le domaine d'application de la fonction de courtage couvre tout système ODP dans lequel il est requis d'offrir et de découvrir des services de façon progressive, dynamique et ouverte.



## Source

La Recommandation X.950 de l'UIT-T a été approuvée le 9 août 1997. Un texte identique est publié comme Norme internationale ISO/CEI 13235-1.

## AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

La Conférence mondiale de normalisation des télécommunications (CMNT), qui se réunit tous les quatre ans, détermine les thèmes d'études à traiter par les Commissions d'études de l'UIT-T lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution n° 1 de la CMNT.

Dans certains secteurs de la technologie de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT n'avait pas été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

# TABLE DES MATIÈRES

## Introduction

La rapide croissance des applications réparties a fait naître le besoin d'un cadre pour coordonner la normalisation du traitement réparti ouvert (ODP). Le modèle de référence ODP fournit ce cadre. Il établit une architecture qui permet la prise en compte de la répartition, de l'interopérabilité et de la portabilité.

L'une des composantes de l'architecture décrite dans la Rec. UIT-T X.903 | ISO/CEI 10746-3: Traitement réparti ouvert – Modèle de référence: architecture (3e partie du modèle RM-ODP) est la fonction de courtage ODP, qui permet d'offrir un service et de découvrir les services qui ont été offerts. La présente Recommandation | Norme internationale décrit une architecture applicable aux systèmes qui mettent en œuvre la fonction de courtage. Elle spécifie également les interfaces contenues dans cette architecture.

NOTE – Dans la présente Recommandation | Norme internationale, la spécification des interfaces de traitement est techniquement alignée sur le service de courtage du groupe de gestion des objets (OMG).

Les objectifs de la présente Recommandation | Norme internationale sont les suivants:

- offrir une norme qui soit indépendante de toute réalisation;
- garantir que l'on pourra faire interfonctionner (c'est-à-dire fédérer) les mises en œuvre;
- donner suffisamment de détails pour permettre d'évaluer les revendications de conformité.

L'Annexe A (normative) est une spécification en langage IDL (langage de définition d'interface) du traitement ODP pour les signatures d'interface de la fonction de courtage.

L'Annexe B (normative) est une spécification du langage de contrainte pour la fonction de courtage ODP.

L'Annexe C (normative) est une spécification du langage de recette de contrainte pour la fonction de courtage ODP.

L'Annexe D (informative) est une description du conteneur de types de service.

**NORME INTERNATIONALE**

**RECOMMANDATION UIT-T**

# TECHNOLOGIES DE L'INFORMATION – TRAITEMENT RÉPARTI OUVERT – FONCTION DE COURTAGE: SPÉCIFICATION

## 1 Domaine et champ d'application

Le domaine d'application de la présente Recommandation | Norme internationale est le suivant:

- constituer une spécification d'entreprise pour la fonction de courtage;
- constituer une spécification d'information pour la fonction de courtage;
- constituer une spécification de traitement pour les courtiers (c'est-à-dire les objets assurant la fonction de courtage);
- formuler des prescriptions de conformité en termes de points de conformité.

La présente Recommandation | Norme internationale ne vise pas à indiquer la façon dont la fonction de courtage doit être réalisée. Elle n'inclut donc pas de spécification d'ingénierie.

Le champ d'application de la présente Recommandation | Norme internationale est tout système dans lequel il faut introduire et découvrir progressivement, dynamiquement et ouvertement des services.

## 2 Références normatives

Les Recommandations et Normes internationales suivantes contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente Recommandation | Norme internationale. Au moment de la publication, les éditions indiquées étaient en vigueur. Toutes Recommandations et Normes sont sujettes à révision et les parties prenantes aux accords fondés sur la présente Recommandation | Norme internationale sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des Recommandations et Normes indiquées ci-après. Les membres de la CEI et de l'ISO possèdent le registre des Normes internationales en vigueur. Le Bureau de la normalisation des télécommunications de l'UIT tient à jour une liste des Recommandations de l'UIT-T en vigueur.

- Recommandation UIT-T X.901 (1997) | ISO/CEI 10746-1:1997, *Technologies de l'information – Traitement réparti ouvert – Modèle de référence: aperçu général*.
- Recommandation UIT-T X.902 (1995) | ISO/CEI 10746-2:1996, *Technologies de l'information – Traitement réparti ouvert – Modèle de référence: fondements.*
- Recommandation UIT-T X.903 (1995) | ISO/CEI 10746-3:1996, *Technologies de l'information – Traitement réparti ouvert – Modèle de référence: architecture.*
- Recommandation UIT-T X.920 (1997) | ISO/CEI 14750:1998, *Technologies de l'information – Traitement réparti ouvert – Langage de définition d'interface.*
- ISO/CEI 13568[1)]: *Technologies de l'information – Le langage de spécification Z*.

## 3 Notations

La spécification d'information de la fonction de courtage est décrite au moyen du langage SDL-Z. La signature de l'interface de traitement pour la fonction de courtage est décrite à l'article 8 et dans l'Annexe A au moyen du langage de définition d'interface (IDL, *interface definition language*) du traitement ODP.

1) A publier.

# 4 Définitions

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

La présente Spécification est fondée sur le cadre d'abstractions et de concepts mis au point dans le modèle RM-ODP. Elle fait usage des termes suivants, qui sont définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 (Partie 2 du modèle RM-ODP: fondements):

a) action

b) activité

c) comportement

d) compatibilité de comportement

e) rattachement

f) objet client

g) point de conformité

h) contrat

i) domaine

j) comportement d'établissement

k) défaillance

l) identificateur

m) objet initiateur

n) instance

o) interaction

p) interface

q) signature d'interface

r) nom

s) objet

t) obligation

u) système ODP

v) permission

w) politique

x) interdiction

y) qualité de service

z) point de référence

aa) objet répondeur

bb) rôle

cc) objet serveur

dd) sous-type

ee) supertype

ff) gabarit

gg) type-gabarit

hh) courtage

ii) transparence

jj) type

kk) point de vue

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

La présente Spécification est fondée sur le cadre d'abstractions et de concepts mis au point dans le modèle RM-ODP. Elle fait usage des termes suivants, qui sont définis dans la Rec. UIT-T X.903 | ISO/CEI 10746-3 (Partie 3 du modèle RM-ODP: architecture):

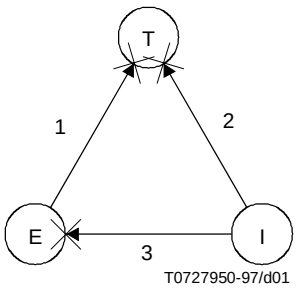
a) communauté
b) gabarit d'interface de traitement
c) point de vue traitement
d) schéma dynamique
e) point de vue ingénierie
f) point de vue entreprise
g) exportateur
h) point de vue information
i) schéma d'invariant
j) schéma
k) exportation de service
l) importation de service
m) offre de service
n) schéma statique
o) point de vue technologie
p) fédération <X>

## 5 Aperçu général de la fonction de courtage ODP

Dans le cadre des objectifs du traitement ODP, qui consiste à offrir une répartition transparente des services sur des plates-formes ou réseaux hétérogènes, le rôle de la fonction de courtage est de permettre aux usagers de trouver des possibilités de service. La découverte dynamique des services est un corollaire de leur répartition.

La fonction de courtage ODP facilite l'offre et la découverte d'instances d'interfaces qui fournissent des services de types particuliers. Un courtier est un objet qui prend en charge la fonction de courtage dans un environnement réparti. On peut le considérer comme un objet par l'intermédiaire duquel d'autres objets peuvent faire connaître leurs capacités et adapter leurs besoins aux capacités signalées. La signalisation d'une capacité ou d'une offre de service est appelée «exportation». L'adaptation aux besoins ou la découverte de services est appelée «importation». L'exportation et l'importation facilitent la découverte dynamique de services et le rattachement retardé à des services.

Pour effectuer une exportation, un objet donne au courtier une description de service accompagnée de la localisation d'une interface à laquelle ce service est disponible. Pour effectuer une importation, un objet demande au courtier un service possédant certaines caractéristiques. Le courtier effectue une vérification sur la base des descriptions de service dont il dispose et répond à l'importateur en lui indiquant la localisation d'interface(s) offrant un service concordant. L'importateur peut ensuite entrer en interaction avec un tel service. Ces interactions sont représentées sur la Figure 1.




Séquence des interactions:
1. Exportation
2. Importation
3. Interaction avec le service

**Figure 1 – Interaction entre le courtier et ses clients**

On peut dissocier, d'une part, l'interaction avec le service et, d'autre part, les interactions avec la fonction de courtage (exportations et importations) en modélisant explicitement un objet *fournisseur de services* et un objet *utilisateur de services*. Ce modèle implique des interactions entre fournisseur et exportateur de services ainsi qu'entre importateur et utilisateur de services, qui négocient des actions comme défini dans la Rec. UIT-T X.902 | ISO/CEI 10746-2. Ces interactions implicites ne sont cependant pas soumises à la présente Spécification.

Etant donné le nombre réel des offres de service dans le monde et les différents besoins qui seront exprimés par les utilisateurs du service de courtage, il est inévitable que ce service soit subdivisé et que les offres de service soient partitionnées.

Chaque partition répondra, en première instance, aux besoins de courtage d'une communauté de clients (exportateurs et importateurs). Lorsqu'un client a besoin, pour ses activités de courtage, d'un domaine de visibilité plus étendu que celui qui correspond à une seule partition, ce client accédera, directement ou indirectement, à d'autres partitions. En accès direct, le client entrera en interaction avec les courtiers traitant ces partitions. En accès indirect, le client n'entrera en interaction qu'avec un seul courtier et celui-ci interagira avec d'autres courtiers, chargés d'autres partitions. Cette dernière possibilité est appelée *fédération de courtiers*. Dans certains cas, des intercepteurs peuvent être nécessaires entre des courtiers fédérés.

L'utilisateur d'un courtier qui interfonctionne avec d'autres courtiers ne peut s'associer qu'à un seul courtier et peut avoir accès transparent aux offres de service des autres courtiers avec lesquels ce courtier peut interfonctionner.

La fonction de courtage permet donc, dans un environnement de traitement ODP:

- aux objets d'exporter (signaler) des services;
- aux objets d'importer des informations sur un ou plusieurs services exportés, selon certains critères;
- aux courtiers de se fédérer.

## 5.1 Diversité et échelonnabilité

Le concept de courtage permettant de découvrir de nouveaux services est applicable à une large gamme de scénarios. Un courtier peut détenir un grand nombre d'offres de service et sa réalisation peut tendre à être fondée sur une base de données. Un courtier peut également ne détenir qu'un petit nombre d'offres de service et être donc réalisable sous la forme d'un courtier résident en mémoire. Ces deux cas possèdent des qualités différentes: la disponibilité et l'intégrité dans le premier cas, la performance dans le second. Les variantes de ces scénarios montrent la nécessité d'une échelonnabilité, aussi bien vers le haut pour les très grands systèmes que vers le bas pour les petits systèmes rapides.

Pour découvrir une certaine offre de service, un courtier a besoin que toutes les offres lui soient (en quelque sorte) visibles. Une partition donnée ne peut pas contenir toutes les offres. Celles-ci seront nécessairement réparties entre plusieurs partitions. Un courtier devra donc posséder, en plus d'un certain nombre d'offres, des renseignements sur d'autres partitions. Il n'est cependant pas nécessaire qu'un courtier soit informé de toutes les autres partitions. Certaines de ces connaissances peuvent être obtenues indirectement, par l'intermédiaire d'autres courtiers.

Le partitionnement de l'espace consacré aux offres de services et les connaissances limitées qui sont détenues dans une partition donnée au sujet des autres partitions constituent la base des prescriptions à observer, tant pour la répartition que pour la mise en contexte de la fonction de courtage.

## 5.2 Mise en liaison de courtiers

Les prescriptions de mise en contexte de l'espace consacré aux offres de services et de répartition de la fonction de courtage sont, dans les deux cas, satisfaites par une mise en liaison de courtiers. Le fait de mettre un courtier en liaison avec des homologues permet implicitement à ce courtier de mettre à la disposition de ses propres clients l'espace réservé à des offres de services par ces homologues.

Chaque courtier a un horizon limité aux homologues auxquels il est explicitement lié. Si ces homologues sont à leur tour mis en liaison avec d'autres courtiers, un grand nombre de courtiers pourront être atteints à partir d'un courtier de départ donné. Les courtiers sont liés de façon à former un graphe orienté, dont les informations descriptives sont réparties entre courtiers. Ce graphe est appelé *graphe de courtage*.

Les liaisons peuvent traverser des frontières de domaine administratif, de domaine technologique, etc. Le courtage peut donc former un système fédéré, c'est-à-dire englobant de nombreux domaines.

## 5.3 Politique

Pour répondre aux diverses prescriptions susceptibles de régir la fonction de courtage, quelques degrés de liberté sont nécessaires lors de la spécification du comportement d'un objet de courtage. A cette fin – tout en restant en conformité

avec les objectifs de la présente Spécification – le concept de *politique* est utilisé en tant que cadre de description du comportement de tout système de courtage ODP conforme.

La présente Spécification identifie un certain nombre de politiques et leur attribue des sémantèmes. Chaque politique détermine partiellement le comportement d'un courtier. Pour faire valoir sa conformité, une réalisation peut avoir besoin d'indiquer quelle est la combinaison de politiques qui garantira un comportement conforme.

Les politiques peuvent être communiquées en cours d'interaction, auquel cas elles se rapportent à une hypothèse quant au comportement attendu.

# 6 Spécification d'entreprise de la fonction de courtage

Le domaine d'application d'une spécification d'entreprise est défini dans la Rec. UIT-T X.903 | ISO/CEI 10746-3 (Partie 3 du modèle RM-ODP: architecture). Cette spécification d'entreprise identifie les objectifs et les déclarations de politique qui régissent les activités d'une fonction de courtage.

L'objectif de la fonction de courtage est de donner la possibilité d'offrir et de découvrir des instances d'un type de service particulier, ayant des caractéristiques particulières.

Une communauté de courtage se compose de membres ayant différents rôles, par exemple courtier, exportateur, importateur. Un objet peut avoir plusieurs rôles dans la même communauté. Par exemple, un objet peut à la fois être importateur et exportateur.

Les activités de courtage de la communauté sont les exportations et les importations de services. Ces activités sont régies par un ensemble de politiques de la communauté de courtage. Une activité d'importation de service peut se propager d'une communauté de courtage à une autre. Dans un tel cas, les domaines associés à ces deux courtiers sont fédérés. Les frontières de ces domaines de courtage peuvent coïncider avec celles d'un autre domaine (par exemple un domaine de type ou un domaine de politique de sécurité).

Une politique est un ensemble de règles visant un objectif particulier. Chaque règle contraint certains aspects d'un comportement de courtier compatible avec l'objectif commun. Les membres de la communauté de courtage sont tenus de respecter les règles des politiques. Ces règles fournissent les directives régissant les décisions visant à répondre aux objectifs de la communauté. Ces règles ne sont pas prescrites dans la présente Spécification. C'est la spécification d'entreprise qui identifie l'ensemble des politiques qui limitent le courtier dans certains types de comportement. Les politiques identifiées constituent un cadre dans lequel le comportement de l'objet-courtier peut être mis en œuvre ou configuré.

## 6.1 Communautés

**6.1.1 communauté de courtage**: communauté d'objets constituée en vue du courtage et régie par une politique de courtage. Les objets remplissent les rôles énumérés au 6.2.

Une communauté de courtage donnée peut (à un niveau d'abstraction donné) être subdivisée en un certain nombre de communautés de courtage interfonctionnant à un deuxième niveau (plus détaillé) d'abstraction. Sous réserve de la politique de la communauté, l'interfonctionnement des communautés de courtage au niveau détaillé est susceptible de donner l'impression qu'il n'existe qu'une seule communauté abstraite, ce qui permet aux objets ayant, dans une sous-communauté donnée, un rôle de courtier, d'importateur ou d'exportateur, d'interagir avec les objets d'une sous-communauté.

## 6.2 Rôles

Les objets peuvent jouer les rôles suivants au sein d'une communauté de courtage.

**6.2.1 courtier**: rôle d'une entité qui enregistre les offres de service issues d'objets exportateurs et qui, sur demande, renvoie à l'importateur des offres de service conformes à certains critères.

**6.2.2 exportateur**: rôle d'une entité qui enregistre des offres de service auprès de l'objet-courtier.

**6.2.3 importateur**: rôle d'une entité qui obtient de l'objet-courtier des offres de service répondant à certains critères.

**6.2.4 administrateur-courtier**: rôle d'une entité qui définit, gère et met en œuvre la politique de courtage de l'objet-courtier. L'administrateur-courtier est l'objet qui contrôle un domaine de courtier (celui-ci et son ensemble d'offres de services).

**6.2.5 offre de service**: rôle d'une entité qui tient à jour la description d'un service.

NOTE – Une telle description peut constituer la base d'un futur contrat. [COMP/PV12]

## 6.3 Activités

Les activités suivantes relèvent d'une communauté de courtage.

**6.3.1 exportation de service**: enchaînement d'actions par un objet-exportateur et par l'objet-courtier, qui établit et termine une liaison au cours de laquelle l'objet-courtier est autorisé à communiquer à un groupe d'objets importateurs l'offre de service de l'objet-exportateur.

**6.3.2 importation de service**: enchaînement d'actions entre un objet-importateur et l'objet-courtier, dans lequel l'objet-importateur obtient un certain nombre d'offres de service répondant à certains critères.

## 6.4 Politiques

Les comportements d'objets d'entreprise à l'intérieur d'une communauté de courtage sont régis par les politiques de celle-ci. Certaines politiques régissent des activités de courtage et certaines politiques imposent des contraintes à d'autres aspects relatifs au comportement d'un courtier et à d'autres rôles joués dans la communauté de courtage, en fonction de l'objectif commun de la communauté. Lorsqu'une activité implique des interactions d'objets, la politique résultante sera un compromis entre les politiques des objets interactifs. Ce compromis sera réalisé au moyen d'un processus de type arbitrage.

NOTE – Par exemple, un objet-courtier peut être régi par des politiques telles qu'il soit obligé de propager une recherche à une profondeur de deux liaisons; mais cet objet peut également être autorisé à terminer une recherche après l'avoir propagée à une profondeur d'une seule liaison. Si l'objet-courtier est autorisé (ou obligé) à répondre aux prescriptions d'un importateur concernant la profondeur des liaisons à traverser pour une recherche, il est nécessaire que certaines règles permettent d'effectuer un arbitrage entre politiques contradictoires.

**6.4.1 politique d'activité d'exportation**: la politique d'activité d'exportation est un ensemble de règles relatives à l'activité d'exportation de services (c'est-à-dire concernant une offre de services telle que ceux-ci puissent être découverts ultérieurement par d'autres objets).

La politique d'activité d'exportation peut comporter, entre autres conditions:

- l'obligation de décrire d'une certaine façon une offre de service;
- l'interdiction d'une découverte de l'offre de service par des activités spécifiques d'importation;
- une obligation d'offre de service fournissant des règles à évaluer dans le cadre d'une activité d'importation de service.

Chaque exportateur peut avoir sa propre politique d'activité d'exportation, qui décrira ce que cet exportateur attend d'une exportation de service. L'activité d'exportation de service est donc régie aussi bien par la politique d'exportation du courtier que par la politique d'exportation de l'exportateur.

**6.4.2 politique d'activité d'importation**: la politique d'activité d'importation est un ensemble de règles relatives à l'activité d'importation de services (c'est-à-dire concernant la tentative de découverte de services offerts, répondant à une prescription spécifiée).

La politique d'activité d'importation peut comporter, entre autres conditions:

- l'obligation de limiter l'utilisation de ressources, y compris la durée d'activité;
- l'autorisation de propager l'importation de service vers une ou plusieurs communautés de courtage en interfonctionnement.

**6.4.3 politique d'arbitrage**: la politique d'arbitrage est un ensemble de règles relatives à l'arbitrage en cas d'apparition de règles contradictoires au cours d'activités de courtage.

La politique d'arbitrage peut comporter, entre autres conditions, l'obligation d'effectuer un arbitrage en faveur des règles de l'objet-courtier au sujet des opérations suivantes:

- l'utilisation de ressources au cours d'une importation de service;
- la propagation d'activités d'importation de services.

**6.4.4 politique d'acceptation d'offre de service**: la politique d'acceptation d'offre de service est un ensemble de règles limitant l'ensemble des offres de service qui seront acceptées par le courtier.

**6.4.5 politique de gestion de type**: la politique de gestion de type est un ensemble de règles relatives à la spécification des types et aux relations entre les types.

NOTE 1 – La politique de gestion de type peut consister à renvoyer vers une fonction de conteneur de types au sujet d'un de ces aspects ou des deux.

NOTE 2 – Exemples: utilisation d'équivalences de noms ou utilisation de sous-types de signature pour faire concorder des types.

**6.4.6 politique de recherche**: la politique de recherche est un ensemble de règles régissant la recherche d'offres de service appropriées dans le système de courtage.

## 6.5 Règles de structuration

### 6.5.1 Règles de communauté

Dans une communauté de courtage, il doit y avoir un objet qui joue le rôle de courtier (objet-courtier). En devenant membre d'une communauté de courtage, un objet obtient la capacité d'interagir avec l'objet-courtier en jouant un rôle d'importateur ou d'exportateur. Un objet peut jouer le rôle d'exportateur, le rôle d'importateur, ou les deux rôles.

Une «entreprise» peut comporter plusieurs communautés de courtage. Un objet peut être membre de plusieurs communautés de courtage. L'objet-courtier d'une communauté donnée peut jouer un rôle d'importateur ou d'exportateur à l'intérieur d'une autre communauté dont il est membre.

La communauté peut couvrir plusieurs domaines en termes de sécurité, de types, de gestion, de rémunération, etc.

Chaque courtier forme, avec son ensemble d'offres de services, un domaine de courtier. Un ensemble de domaines de courtiers qui interfonctionne dans le cadre d'une communauté de courtage constitue donc une fédération de courtiers.

NOTE – Les domaines de courtiers fédérés n'exigent pas toujours que des intercepteurs soient placés à leurs frontières au point de vue ingénierie.

### 6.5.2 Règles de transfert

Les objets-exportateurs peuvent exporter des offres pour des services qu'ils fournissent à leurs propres interfaces; ils peuvent également exporter des offres pour des services fournis par un objet-fournisseur de services distinct.

Les objets-importateurs peuvent importer des offres de services pour leur propre usage ou pour celui d'objets-utilisateurs de services distincts.

### 6.5.3 Démarcation des règles d'autorité

Chaque objet-administrateur de courtier d'une communauté de courtage a le contrôle complet de son propre objet-courtier.

L'objet-exportateur est responsable de l'exactitude de ses offres de services.

Pour faire partie d'une fédération établie de courtiers:

- un courtier donné n'est pas obligé d'exercer l'activité commencée par un autre courtier;
- chaque courtier doit avoir une autonomie totale en ce qui concerne ses propres politiques de courtage.

En particulier, chaque courtier détermine ses propres politiques de recherche d'homologues dans le groupe des courtiers en interfonctionnement (fédérés).

### 6.5.4 Règles de qualité de service

L'objet-courtier n'est ni redevable ni responsable de la qualité des services décrits dans les offres de service.

Un objet-courtier peut être obligé d'assurer la suppression programmée d'offres de service.

NOTE – Les deux cas suivants constituent un exemple:

1) Une politique d'acceptation d'offre de service pour courtage peut imposer une date d'expiration à des offres de service. L'objet-courtier est autorisé à supprimer les offres de service périmées.
2) Une politique d'importation pour courtage peut interdire à l'objet-courtier de renvoyer des offres de service qui ont expiré au moment d'une importation.

### 6.5.5 Règles de concordance

Une importation de service nécessite une vérification du type de signature à l'interface de traitement. Elle peut aussi impliquer d'autres niveaux de vérification concernant les relations avec des sous-types ou des supertypes, la compatibilité de comportement et les contraintes d'environnement. On peut également effectuer une vérification complémentaire des aspects d'entreprise, d'information, d'ingénierie et de technologie.

# 7 Spécification informationnelle de la fonction de courtage

## 7.1 Aperçu général

Le domaine d'application d'une spécification d'information est défini dans la Rec. UIT-T X.903 | ISO/CEI 10746-3 (Partie 3 du modèle RM-ODP: architecture). Cette spécification d'information décrit les types d'information et les relations que ces types doivent avoir entre eux pour définir la fonction de courtage ODP. Elle utilise le langage

d'information défini dans le modèle RM-ODP et, le cas échéant, interprète ce langage en termes de notation formelle SDL-Z. Les alinéas de la notation formelle comportent du texte anglais intercalé dans le style habituel du langage Z.

La spécification d'information décrite dans le présent article définit les éléments suivants:

- les concepts fondamentaux pour les informations utilisées dans la présente Spécification;
- les schémas statiques, les schémas d'invariant et les schémas dynamiques pour la présente Spécification.

## 7.2 Concepts de base

### 7.2.1 Interfaces

Un service est offert à une interface. Il est nécessaire qu'une offre de service indique le type de signature d'interface et l'identificateur de l'interface de ce service.

#### 7.2.1.1 Type de signature d'interface

Le type de signature d'interface identifie la signature des interfaces d'objets.

En langage Z, les types de signature d'interface sont définis formellement par l'introduction d'un ensemble représentant les valeurs que ces types peuvent avoir:

*[InterfaceSignatureType]*

#### 7.2.1.2 Identificateur d'interface

Un identificateur d'interface désigne l'interface à laquelle un service est disponible ou requis.

En langage Z, ces identificateurs sont formellement définis par un ensemble donné.

### 7.2.2 Type de service

Un service est un ensemble de capacités fournies par un objet à une interface de traitement. C'est une instance d'un type de service.

Une définition de type de service se compose d'un type de signature d'interface, d'un ensemble de définitions de propriété de service et d'un ensemble de règles sur les modes des propriétés de service.

Les définitions de propriété de service sont explicitement décrites dans la spécification formelle en termes de noms, de types de valeur et de modes. Les modes valides sont les suivants:

- normal (lecture et écriture mais présence facultative);
- lecture seulement (lecture mais présence facultative);
- obligatoire (présence obligatoire de la lecture et de l'écriture);
- lecture seulement et obligatoire (lecture seulement, présence obligatoire).

Un type de valeur est un ensemble de valeurs.
*[Name, Value]*

*ValueType* == ℙ *Value*

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

Les propriétés de service contiennent des informations sur les aspects de traitement (tels que le comportement et l'environnement d'une interface) ainsi que sur la description des aspects technologie, ingénierie, information et entreprise du service.

La définition formelle du type de service est donnée dans le schéma *ServiceType* en langage Z. Elle regroupe un type de signature d'interface et un ensemble de définitions de propriété de service.

*ServiceType*

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

En langage Z, on fait appel à des fonctions pour extraire l'ensemble des noms des propriétés qui doivent être présentes (c'est-à-dire de type *mandatory* ou *readonly_mandatory*) ainsi que l'ensemble des noms des propriétés qui ne peuvent pas être modifiées (c'est-à-dire de type *readonly* ou *readonly_mandatory*). Ces deux fonctions sont définies formellement comme suit:

***mandatory_props*** *: ServiceType ② ◴ Name*
***readonly_props*** *: ServiceType ② ◴ Name*

---

*✂ s: ServiceType • **mandatory_props** s =*
*{ n : Name | second (s.prop_defs n) ⍀ {mandatory, readonly_mandatory}}*
*✂ s: ServiceType • **readonly_props** s =*
*{n : Name | second (s.prop_defs n) ⍀ {readonly, readonly_mandatory}}*

### 7.2.3 Règles de sous-typage des types de service

Les règles générales pour le sous-typage des services sont les suivantes pour un courtier conforme.

Dans le cas le plus général, un service de type **b** est un sous-type du type de service **a**, si et seulement si:

- le type de signature d'interface de **b** est un sous-type du type de signature d'interface de **a**;
- toutes les propriétés nommées de **a** sont dans **b**;
- toutes les propriétés nommées de **a** ont un type de valeur qui est un supertype de la propriété nommée de manière identique dans **b**;
- toutes les propriétés nommées de **a** ont un mode qui est un supertype du mode de la propriété nommée de manière identique dans **b**.

NOTE – Les règles ci-dessus sont équivalentes aux règles normales de sous-typage d'interface ODP, si les propriétés sont considérées comme des opérations, avec le type et le mode considérés comme des arguments de retour des opérations.

La représentation en langage Z nécessite la définition de trois relations pour représenter le sous-typage de signature d'interface, le supertypage de valeur et le supertypage de mode. Les règles de sous-typage de signature d'interface sont indiquées dans la Rec. UIT-T X.903 | ISO/CEI 10746-3 et ne sont pas définies plus en détail ici. Les définitions formelles sont données pour les relations de supertypage selon les modes et les types de valeur.

*_ **is_sig_subtype_of** _ : InterfaceSignatureType ③ InterfaceSignatureType*
*_ **is_value_supertype_of** _ : ValueType ③ ValueType*
*_ **is_mode_supertype_of** _ : Mode ③ Mode*

---

*✂ a,b:Mode • a **is_mode_supertype_of** b ⊆*
*(a,b) ⍀ {(normal,readonly),(normal,mandatory),(normal, readonly_mandatory),*
*(readonly,readonly_mandatory), (mandatory,readonly_mandatory)}*
*✂ a,b:ValueType • a **is_value_supertype_of** b ⊆ b ⍁ a*

*__ **is_subtype_of** __ : ServiceType ③ ServiceType*

---

*✂ a,b : ServiceType • b **is_subtype_of** a ⊆*
*b.signature **is_sig_subtype_of** a.signature ⩓*
*dom a.prop_defs ⍁ dom b.prop_defs ⩓*
*(✂ n: dom a.prop_defs •*
*first (a.prop_defs n) **is_value_supertype_of** first (b.prop_defs n) ⩓*
*second (a.prop_defs n) **is_mode_supertype_of** second (b.prop_defs n))*

### 7.2.4 Offre de service

Une offre de service signale un service. C'est une assertion faite par un exportateur au sujet d'un service qui est offert pour utilisation par d'autres objets à une interface de traitement. Elle se compose d'une instanciation d'un type de service, d'un identificateur pour l'offre de service et d'un identificateur pour l'interface au moyen de laquelle le service pourra être utilisé. Une offre de service peut également inclure un ensemble de valeurs de propriété d'offre de service.

Pour la définition en langage Z, un ensemble donné est introduit afin de représenter les identificateurs d'offre de service qui sont univoques dans une communauté de courtage.

*[ServiceOfferIdentifier]*

La définition formelle en langage Z d'une offre de service est donnée par le schéma *ServiceOffer*. Chaque offre de service doit correspondre au type de ce service, c'est-à-dire qu'elle doit avoir une valeur pour chacune des propriétés définies par le type de service comme étant obligatoire ou en lecture seulement-obligatoire; et toutes les propriétés qui appartiennent à ce type de service doivent avoir des valeurs issues des ensembles définis dans ce type de service.

*ServiceOffer*

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

### 7.2.5 Critères et contraintes

Les politiques sont représentées dans le point de vue entreprise par des critères et par des contraintes dans le point de vue information.

Il y a trois aspects à considérer pour la spécification de l'ensemble des offres de services qui sont des résultats acceptables d'une action de recherche ou de sélection. Deux de ces aspects sont les relations de filtrage des propriétés de service et d'offre de service. Le premier aspect définit les propriétés essentielles qui ne peuvent pas être violées pour trouver une concordance avec une offre. Le deuxième aspect est une préférence qui agit comme un filtre de sélection lorsqu'il y a de nombreuses offres qui correspondent aux propriétés essentielles. Le troisième aspect est une relation de détection dans un domaine de visibilité qui limite l'ensemble des offres de service qui doivent être comparées aux règles de concordance. Ces spécifications peuvent être fournies aussi bien par l'importateur que par le système de courtage, la spécification finale étant une combinaison des deux sources. Le système de courtage est défini au 7.3.

#### 7.2.5.1 Critères de concordance

Les critères de concordance sont les règles qui sont appliquées à l'ensemble complet des offres de service afin de fournir un ensemble plus petit d'offres de service.

En langage Z, les critères de concordance peuvent être exprimés sous la forme de l'ensemble de toutes les offres de service qui peuvent répondre aux règles de concordance. Le processus d'application de ces règles à un ensemble de règles de service peut alors être représenté par l'intersection de deux ensembles.

$MatchingCriteria == \mathbb{P} ServiceOffer$

#### 7.2.5.2 Critères de préférence

Les critères de préférence sont les règles qui sont appliquées à l'ensemble des offres de service acceptables pour constituer une séquence ordonnée d'offres de service.

En langage Z, les critères de préférence peuvent être exprimés sous la forme d'une fonction totale allant des ensembles d'offres de service à une séquence d'offres de service. La description formelle de l'application de ces règles est donnée dans le 7.5.10.

*PreferenceCriteria* == ⦶*ServiceOffer* ⦾ **seq** *ServiceOffer*

#### 7.2.5.3 Critères de visibilité

Les critères de visibilité sont les règles qui délimitent l'ensemble des offres de service qui doivent être comparées aux règles de concordance.

En langage Z, les critères de visibilité peuvent être exprimés sous la forme d'un ensemble d'offres de service.

*ScopeCriteria* == ⦶*ServiceOffer*

#### 7.2.5.4 Critères de frontière

Les critères de frontière sont les règles qui délimitent l'ensemble des nœuds atteignables à partir d'un nœud donné.

En langage Z, les critères de frontière peuvent être exprimés sous la forme d'associations.

*EdgeCriteria == Node x Node*

#### 7.2.5.5 Contrainte sur la concordance entre courtiers

Cette contrainte s'exerce sur les critères de concordance imposés par la politique du système de courtage.

En langage Z, les contraintes de concordance peuvent être spécifiées de la même façon que les critères de concordance. L'application de ces contraintes peut être représentée par une intersection d'ensembles.

#### 7.2.5.6 Contrainte sur la préférence entre courtiers

Cette contrainte s'exerce sur les critères de préférence imposés par la politique du système de courtage.

En langage Z, les contraintes de préférence sont représentées par une fonction totale entre deux ensembles de critères de préférence. L'application de cette fonction est décrite au 7.5.10.

*PreferenceConstraint == PreferenceCriteria* ⦾ *PreferenceCriteria*

#### 7.2.5.7 Contrainte sur le domaine de visibilité des courtiers

Cette contrainte s'exerce sur les critères de visibilité imposés par la politique de courtage.

En langage Z, les contraintes de visibilité sont, comme les critères de visibilité, représentées par un ensemble d'offres de service.

#### 7.2.5.8 Contraintes sur le système de courtage

Pour la spécification formelle en langage Z, il est pratique de regrouper toutes les contraintes sur le système de courtage dans le schéma intitulé *TradingSystemConstraints*, dont la définition est donnée au 7.5.9.

*TradingSystemConstraints*

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

### 7.2.6 Demande de recherche

Une demande de recherche est une spécification de la politique d'importateur applicable à une action de recherche particulière.

En langage Z, une demande de recherche peut être modélisée sous la forme d'un schéma composé des critères de concordance, de visibilité et de préférence de l'importateur, ainsi que du type du service recherché. Toutes les offres qui répondent aux conditions de concordance de l'importateur doivent avoir des services ayant un sous-type du type de service spécifié. Cette définition est reprise au 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 Schéma d'invariant

L'espace consacré aux offres de service sera partitionné et chaque partition sera associée à une connaissance d'un nombre limité d'autres partitions. Ce schéma est répété pour viser une partition donnée. Dans la spécification d'information, ce schéma est modélisé sous la forme d'un graphe orienté, dont les nœuds représentent les partitions et les frontières la connaissance associée aux partitions. Ce graphe est appelé *graphe de courtage*.

Il peut y avoir un nombre quelconque de critères de partitionnement de l'espace réservé aux offres de service, dans les sites répartis et au-dessus d'eux. Par exemple, le partitionnement peut être fondé sur les propriétés suivantes:

- propriétés de l'emplacement (par exemple architecture machine);
- propriétés des offres de service (par exemple une classification de sécurité);
- propriétés du service (par exemple sa disponibilité).

Une frontière peut avoir des propriétés qui décrivent la perception d'une partition vue à partir d'une autre partition.

Les éléments constituants du système de courtage sont les suivants:

- un ensemble (*offers*) d'offres de service disponibles pour importation;
- un ensemble (*nodes*) de nœuds par lesquels ces offres de service sont partitionnées;
- une relation (*edges*) entre nœuds afin de représenter les frontières du graphe de courtage; cette relation régit la propagation des recherches;
- des ensembles (*edge_properties*) de propriétés associées aux frontières;
- des offres de service, distinguées par leurs identificateurs d'offre de service, qui sont uniques et univoques. Ces éléments sont représentés par les invariants du schéma *TradingSystem*.

Les nœuds individuels sont définis formellement en langage Z par l'introduction d'un ensemble donné, comme suit:

*[Node]*

Chaque offre de service ne doit être attribuée qu'à un et un seul nœud. Aucun chevauchement ni aucune relation de sous-ensembles n'est permis entre les nœuds. Cette restriction est introduite dans la fonction partielle *partition* qui met en concordance chaque offre de service du système de courtage avec un nœud donné. Un exportateur peut exporter le même service vers plusieurs nœuds en créant d'autres offres de service, ayant d'autres identificateurs d'offre.

Le point de vue traitement applique des nœuds à des courtiers. Cette mise en concordance est biunivoque (chaque nœud contenu dans le point de vue information est un objet-courtier unique). Du point de vue traitement, une frontière correspond donc à une interface de traitement entre courtiers.

Le schéma suivant représente en langage Z l'état du système de courtage:

*TradingSystem*

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

---

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

## 7.4 Schéma statique

Le schéma statique applique l'état du système de courtage à un moment précis.

L'action visant à initialiser un système de courtage est incluse ici afin que la spécification formelle en langage Z soit cohérente et complète.

Lorsqu'un système de courtage est créé, il a la valeur nulle pour chaque état élémentaire:

*Initialize*

$TradingSystem'$

---

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

## 7.5 Schémas dynamiques

Ce paragraphe présente les schémas d'informations dynamiques qui décrivent les changements d'état associés aux actions suivantes:

- exportation: adjonction d'une offre de service issue de l'espace des offres de service du système de courtage;
- retrait: retrait d'une offre de service issue de l'espace des offres de service du système de courtage;
- modification d'offre: modification des valeurs de propriété de service et d'offre de service, associées à une offre de service, sans modification de l'identificateur de cette offre de service;
- adjonction de frontière: adjonction d'une frontière à l'ensemble de frontières du système de courtage;
- suppression de frontière: suppression d'une frontière de l'ensemble de frontières du système de courtage;
- modification de frontière: modification des propriétés d'une frontière;
- adjonction de nœud: adjonction d'un nœud à l'ensemble des nœuds du système de courtage;
- suppression de nœud: suppression d'un nœud de l'ensemble des nœuds du système de courtage;
- recherche: recherche du sous-ensemble d'offres de service qui répond à certains critères de concordance et à certains critères de visibilité;
- sélection: utile spécialisation de l'action de recherche, qui renvoie une séquence d'offres de service ordonnées en fonction d'une certaine contrainte de préférence.

### 7.5.1 Exportation

Le schéma dynamique d'exportation (*Export*) décrit le comportement d'une adjonction d'offre de service au système de courtage.

Le déroulement normal d'une exportation d'offre de service est le suivant. La nouvelle offre de service est ajoutée à l'ensemble existant d'offres de service et est associée à un nœud déterminé. La source de la composante identificatrice de l'offre de service n'est pas précisée: cette composante est produite par le courtier, plutôt que par l'exportateur. L'identificateur d'offre de service est renvoyé à l'exportateur. Les relations frontalières et les propriétés associées aux frontières ne sont pas modifiées.

La condition préalable d'une telle action est que le nouvel identificateur d'offre de service ne soit pas déjà utilisé dans le système de courtage. Une autre condition préalable est que le nœud auquel l'offre doit être associée soit présent dans le système de courtage.

En langage Z, le schéma *Export* représente le comportement correspondant.

*ExportOK*

$\Delta TradingSystem$
$new\_offer? : ServiceOffer$
$node? : Node$
$service\_offer\_identifier! : ServiceOfferIdentifier$

---

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

Si les conditions préalables du schéma *ExportOK* ne sont pas satisfaites, l'état du système de courtage reste inchangé. C'est ce qui est représenté en langage Z dans le schéma d'erreur ci-dessous, dont la condition préalable est la négation de la condition préalable *ExportOK*.

*ExportError*

$\Xi TradingSystem$
$new\_offer? : ServiceOffer$
$node? : Node$

---

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

L'action *Export* réussira ou échouera, selon les conditions préalables des schémas Z ci-dessus.

$Export \triangleq ExportOK \vee ExportError$

### 7.5.2 Retrait

Le schéma dynamique *Withdraw* supprime une offre de service du système de courtage.

L'offre est retirée de l'ensemble des offres. Les frontières et leurs propriétés ne sont pas modifiées. On notera que ce comportement ne spécifie pas la façon dont l'action a été lancée. Le comportement s'applique donc chaque fois qu'une offre est retirée par le courtier, par l'exportateur ou par un quelconque agent autorisé.

La condition préalable de cette action est que l'offre de service existe dans le système de courtage.

En langage Z, le schéma *Withdraw* représente le comportement correspondant.

*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 les conditions préalables du schéma *WithdrawOfferOK* ne sont pas satisfaites, l'état du système de courtage reste sans changement.

*WithdrawOfferError*

*ΞTradingSystem*
*old_offer?* : *ServiceOffer*

---

*old_offer?* ∉ *offers*

L'action *WithdrawOffer* réussira ou échouera, selon les conditions préalables des schémas Z ci-dessus.

*WithdrawOffer* ⩧ *WithdrawOfferOK* ⩔ *WithdrawOfferError*

### 7.5.3 Modification d'offre

Le schéma dynamique *ModifyOffer* définit le comportement d'une modification des propriétés de service, associées ou non à une offre de service. Ce comportement n'est pas simplement la séquence d'une opération *Withdraw* et d'une opération *Export*, car il garantit aussi que l'identificateur de l'offre de service est conservé.

Une offre de service peut être modifiée de trois façons:

- des propriétés peuvent être supprimées, à condition qu'il ne s'agisse pas de propriétés obligatoires;
- de nouvelles propriétés peuvent être ajoutées, à condition qu'une valeur ne leur soit pas déjà attribuée dans l'offre de service;
- les propriétés existantes peuvent être mises à jour, à condition qu'elles ne soient pas en mode lecture seulement.

En langage Z, le schéma *ModifyServiceOffer* représente le comportement correspondant. Sa définition est donnée en plusieurs étapes. Tout d'abord, on donne une définition pour modifier une offre de service en remplaçant les valeurs des propriétés de service, à condition que les contraintes ci-dessus soient respectées. Les autres éléments de l'offre ne sont pas affectés: en particulier, l'identificateur d'offre de service est conservé. On notera que ce comportement ne spécifie pas la façon dont l'action est lancée. Le comportement s'appliquera donc chaque fois qu'une offre sera modifiée par le courtier, par l'exportateur ou par un autre agent autorisé.

*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*

Des conditions d'erreur apparaissent pour ce comportement lorsque l'on essaie:

- de mettre à jour ou de modifier une propriété qui n'existe pas;
- de modifier une propriété qui est soit en lecture seulement soit obligatoirement en lecture seulement;
- ou d'ajouter une nouvelle propriété qui existe déjà dans les propriétés du service.

Si cela se produit, toutes les propriétés de l'offre de service, ainsi que la signature d'interface, restent inchangées.

*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 technique de promotion du langage Z est maintenant employée pour promouvoir le schéma *ModifyServiceOffer* de manière à l'appliquer à une offre spécifique dans le système de courtage. L'offre à modifier est identifiée par la définition d'un schéma de cadrage, *ModifyOffer*, qui indique:

- que l'identificateur de l'offre à modifier est connu du système de courtage;
- qu'après l'action, l'offre choisie a été modifiée selon la nouvelle valeur. Toutes les autres offres restent inchangées et l'offre reste dans le nœud initial;
- les frontières et leurs propriétés restent inchangées.

*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*

Des conditions d'erreur apparaissent pour ce comportement lorsque la condition préalable de l'opération de modification *ModifyOffer* n'est pas vérifiée. Cela se produit lorsque l'on essaie de modifier une offre qui n'est pas présente dans le système de courtage. Celui-ci reste alors dans le même état.

*ModifyOfferError*

*TradingSystem*
*modified_offer?* : *ServiceOffer*

---

*modified_offer? offers*

Finalement, le comportement de type *ModifyOffer* est défini en termes d'opération *ModifyServiceOffer*, de schéma de cadrage ☞*ModifyOffer* (qui identifie une offre particulière à modifier) et de conditions d'erreur (définies dans le schéma *ModifyOfferError*). Les éléments constituants du schéma ✎*ServiceOffer* sont cachés, de façon à ne pas apparaître dans la partie déclarative du schéma *ModifyOffer*, conformément aux conventions usuelles du langage Z.

$$ModifyOffer \triangleq ((ModifyServiceOfferOK \gg \text{☞}ModifyOffer) \gg ModifyOfferError) \setminus (service\_type, prop\_vals, service\_offer\_identifier, interface\_identifier, service\_type\bigcirc, prop\_vals\bigcirc, service\_offer\_identifier\bigcirc, interface\_identifier')$$

### 7.5.4 Adjonction d'une frontière

Le schéma dynamique *AddEdge* définit le comportement associé à l'adjonction d'une frontière au graphe de courtage. Une frontière est ajoutée entre les deux nœuds fournis et les nouvelles propriétés de frontière sont associées à cette frontière. L'ensemble des offres de service et des nœuds reste inchangé dans le système de courtage.

Les conditions préalables de ce schéma dynamique sont que les deux nœuds existent dans le système de courtage et qu'aucune frontière n'existe déjà entre ces deux nœuds, dans le même sens.

En langage Z, le schéma *AddEdge* représente le comportement correspondant.

*AddEdgeOK*

$$\text{✎}TradingSystem$$
$$node1?, node2? : Node$$
$$new\_edge\_properties? : \text{◴}Property$$

---

$$\{node1?, node2?\} \not\in nodes$$
$$(node1?, node2?) \not\in edges$$
$$edges\bigcirc = edges \Rightarrow \{node1? \mapsto node2?\}$$
$$edge\_properties\bigcirc = edge\_properties \Rightarrow \{(node1?, node2?) \mapsto new\_edge\_properties?\}$$
$$offers\bigcirc = offers$$
$$nodes\bigcirc = nodes$$
$$partition\bigcirc = partition$$

Si les conditions préalables du schéma *AddEdgeOK* ne sont pas respectées, l'état du système de courtage reste inchangé.

*AddEdgeError*

$$\text{✠}TradingSystem$$
$$node1?, node2? : Node$$

---

$$node1? \in nodes \gg$$
$$node2? \in nodes \gg$$
$$(node1?, node2?) \in edges$$

L'action *AddEdge* réussira ou échouera, selon les conditions préalables des schémas Z ci-dessus.

$$AddEdge \triangleq AddEdgeOK \gg AddEdgeError$$

### 7.5.5 Suppression d'une frontière

Le schéma dynamique *RemoveEdge* supprime une frontière du graphe de courtage. Après ce schéma dynamique, l'ensemble des offres de service et des nœuds reste inchangé à l'intérieur du système de courtage. Les propriétés associées à chaque frontière sont supprimées de l'ensemble *edge_properties*.

La condition préalable de cette action est que la frontière fournie soit dans l'ensemble des frontières actuelles.

En langage Z, le schéma *RemoveEdge* représente le comportement correspondant.

*RemoveEdgeOK*

*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 les conditions préalables du schéma *RemoveEdgeOK* ne sont pas satisfaites, l'état du système de courtage reste inchangé.

*RemoveEdgeError*

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

---

*old_edge?* ∉ *edges*

L'action *RemoveEdge* réussira ou échouera, selon les conditions préalables des schémas Z ci-dessus.

*RemoveEdge* ⩤ *RemoveEdgeOK* ⩥ *RemoveEdgeError*

### 7.5.6 Modification d'une frontière

Le schéma dynamique *ModifyEdge* modifie les propriétés associées à une frontière. Les anciennes propriétés associées à cette frontière sont remplacées par les nouvelles propriétés. L'ensemble des offres de service et l'ensemble des nœuds restent inchangés, de même que les propriétés associées à toutes les autres frontières.

La condition préalable pour cette action est que la frontière fournie existe.

En langage Z, le schéma *ModifyEdge* représente le comportement correspondant.

*ModifyEdgeOK*

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

---

*edge?* ⩤ *edges*
*edge_properties*′ = *edge_properties* ⩥ {*edge?* ↦ *new_edge_properties?*}
*edges*′ = *edges*
*offers*′ = *offers*
*nodes*′ = *nodes*
*partition*′ = *partition*

Si les conditions préalables du schéma *ModifyEdgeOK* ne sont pas respectées, l'état du système de courtage reste inchangé.

*ModifyEdgeError*

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

---

*edge?* ∉ *edges*

L'action *ModifyEdge* réussira ou échouera, selon les conditions préalables des schémas Z ci-dessus.

*ModifyEdge* ✧ *ModifyEdgeOK* ➘ *ModifyEdgeError*

### 7.5.7 Adjonction d'un nœud

Le schéma dynamique *AddNode* décrit le comportement d'adjonction d'un nœud au système de courtage.

La condition préalable de cette action est que le nœud à ajouter n'existe pas déjà dans le système de courtage. Comme le nouveau nœud n'était pas un élément de l'ensemble *nodes* et qu'il n'était donc pas visible par la fonction *partition*, aucune offre existante n'est introduite dans le nouveau nœud. Cela implique que les nouveaux nœuds ne contiennent pas d'offres de service.

En langage Z, le schéma *AddNode* représente le comportement correspondant.

*AddNodeOK*

$\Delta TradingSystem$
$new\_node? : Node$

---

$new\_node? \notin nodes$
$offers' = offers$
$nodes' = nodes \cup \{ new\_node? \}$
$edges' = edges$
$edge\_properties' = edge\_properties$
$partition' = partition$

Si les conditions préalables du schéma *AddNodeOK* ne sont pas vérifiées, l'état du système de courtage n'est pas modifié.

*AddNodeError*

$\Xi TradingSystem$
$new\_node? : Node$

---

$new\_node? \in nodes$

L'action *AddNode* réussira ou échouera, selon les conditions préalables des schémas Z ci-dessus.

*AddNode* ✧ *AddNodeOK* ➘ *AddNodeError*

### 7.5.8 Suppression d'un nœud

Le schéma dynamique *RemoveNode* définit le comportement de suppression d'un nœud présent dans le système de courtage. Les conditions préalables sont que le nœud à supprimer soit présent dans le système de courtage, qu'il ne contienne pas d'offres de service et qu'il n'ait plus de frontières avec d'autres nœuds.

Le comportement de traitement peut combiner ce schéma dynamique avec d'autres schémas dynamiques pour supprimer des offres de service et des frontières.

En langage Z, le schéma *RemoveNode* représente le comportement correspondant.

*RemoveNodeOK*

$\Delta TradingSystem$
$old\_node? : Node$

---

$old\_node? \in nodes$
$old\_node? \notin ran\ partition$
$old\_node? \notin dom\ edges \lhd ran\ edges$
$offers' = offers$
$nodes' = nodes \setminus \{old\_node?\}$
$edge\_properties' = edge\_properties$
$edges' = edges$
$partition' = partition$

Si les conditions préalables du schéma *RemoveNodeOK* ne sont pas vérifiées, l'état du système de courtage n'est pas modifié.

*RemoveNodeError*

$$
\begin{array}{l}
\Xi TradingSystem \\
old\_node? : Node \\
\hline
old\_node? \notin nodes \\
old\_node? \notin ran\ partition \\
old\_node? \notin dom\ edges \lhd ran\ edges
\end{array}
$$

L'action *RemoveNode* réussira ou échouera, selon les conditions préalables des schémas Z ci-dessus.

$$RemoveNode \gg RemoveNodeOK \gg RemoveNodeError$$

### 7.5.9 Recherche

Le schéma dynamique *Search* est le comportement qui explore le système de courtage, qui est limité par certaines contraintes de visibilité et qui renvoie un ensemble d'offres de service répondant à certains critères de concordance. Cette action ne modifie pas l'état du système de courtage. Sa sortie est un ensemble d'offres de service. Son entrée est une demande de recherche.

Les offres qui répondent au résultat de la recherche doivent satisfaire aux conditions suivantes:

- elles doivent être du type ou du sous-type de service correct;
- elles doivent être contenues dans un nœud qui est acceptable aussi bien selon les besoins de l'importateur que selon les contraintes du système de courtage, et qui doit pouvoir être atteint à partir du point de départ;
- elles doivent correspondre aux critères de concordance de l'importateur;
- elles doivent satisfaire aux contraintes de concordance du système de courtage.

En langage Z, le schéma *Search* représente le comportement correspondant. La spécification formelle du schéma de recherche représente l'effet des critères de concordance et des critères de recherche en tant qu'ensembles d'offres de service. Le résultat de la recherche est obtenu par simple intersection d'ensembles. Toutes les offres qui sont renvoyées doivent être atteignables à partir du point de départ initial de la recherche. On exprime formellement cette propriété en déclarant que toutes les offres de ce type doivent être contenues dans un nœud qui apparaît dans l'enveloppe transitive de la relation *edges*.

*SearchOK*

$$
\begin{array}{l}
\Xi TradingSystem \\
\Xi TradingSystemConstraints \\
SearchRequest? \\
starting\_point? : Node \\
search\_result! : \mathbb{P}\, ServiceOffer \\
\hline
starting\_point? \notin nodes \\
let\ e: EdgeCriteria \bullet e \subseteq edges \\
partition \upharpoonright search\_result! \oplus \{\} \\
\quad \{x : Node \mid (starting\_point?, x) \notin e^{+}\} \\
search\_result! \{\} \\
\quad importer\_matching? \rhd trader\_matching \rhd importer\_scope? \rhd trader\_scope
\end{array}
$$

Si l'on tente de lancer la recherche à partir d'un nœud qui n'existe pas, aucune offre de service n'est renvoyée et l'état du système de courtage n'est pas modifié.

*SearchError*

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

---

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

Le comportement de l'action de recherche est complètement décrit par la combinaison des schémas Z ci-dessus.

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

### 7.5.10 Sélection

Le schéma dynamique de sélection est le comportement qui ordonne un ensemble d'offres de service conformément à certains critères de préférence. Ces critères sont une combinaison des préférences de l'importateur et des contraintes du système de courtage. Ces contraintes comprennent certains aspects de la politique d'arbitrage de la communauté de courtage. Les contraintes du système de courtage modifient la préférence exprimée par l'importateur. Cette action ne modifie pas l'état du courtier.

*Selection*

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

---

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

La spécification formelle en langage Z utilise le schéma *Selection* pour décrire l'application de critères de préférence et de contraintes au résultat d'une recherche. L'action *Select* supprime, de la signature du schéma *Selection*, le résultat d'une recherche.

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

NOTE – Du point de vue traitement, les comportements de recherche et de sélection sont combinés dans l'opération d'interrogation.

## 8 Spécification de traitement de la fonction de courtage

Le domaine d'application de la spécification de traitement est défini dans la Rec. UIT-T X.903 | ISO/CEI 10746-3 (Partie 3 du modèle RM-ODP: architecture).

Du point de vue traitement, les interfaces d'un courtier avec son environnement sont visibles. La spécification de traitement indiquée dans la présente Spécification définit des gabarits d'interface de traitement (client et serveur) pouvant être instanciés par un objet-courtier.

Afin que les réalisateurs puissent délimiter les classes de conformité des courtiers (voir article 9) selon différentes combinaisons d'interfaces (et donc différentes fonctionnalités), la présente Spécification range les interfaces en deux catégories:

- les interfaces fonctionnelles, utilisées pour le groupement d'opérations fondées sur la fourniture d'une fonctionnalité. Les cinq interfaces fonctionnelles d'un courtier sont les suivantes: consultation, enregistrement, procuration, liaison et administration. Deux interfaces auxiliaires: itérateur d'offre et itérateur d'identificateur d'offre sont également spécifiées;
- les interfaces abstraites, utilisées pour le groupement d'attributs en lecture seulement fondés sur la prescription de prise en charge d'une interface fonctionnelle. La présente Spécification décrit les interfaces abstraites suivantes: attributs de support; attributs d'importation; attributs de liaison et composantes de courtier.

Les signatures pour les opérations des interfaces d'administration et de liaison sont définies de façon à assurer la portabilité des courtiers. Le comportement est spécifié pour les opérations de gestion de liaison.

NOTE 1 – Les opérations administratives internes telles que la création et la suppression d'interfaces de courtage sont considérées comme des opérations relevant de la fonction générique de gestion d'objets et ne sont donc pas définies dans la présente Spécification.

Un courtier peut être client de plusieurs fonctions génériques RM-ODP qui feront l'objet de futures études de normalisation. Aucune signature ni aucun comportement n'est spécifié par la présente Spécification pour les opérations effectuées aux interfaces avec de telles fonctions de serveur.

Le langage IDL du traitement ODP (Rec. UIT-T X.920 | ISO/CEI 14750) est utilisé dans la présente Spécification pour exprimer des signatures d'interface effectuant des opérations de traitement. L'utilisation de cette notation n'implique pas l'emploi d'un quelconque mécanisme ou protocole spécifique de prise en charge.

En plus de la présente spécification de traitement, l'Annexe A contient, dans les modules CosTrading et CosTradingDynamic, une spécification sur le langage IDL du traitement ODP pour les signatures opérationnelles. L'Annexe D contient, dans le module CosTradingRepos, une spécification sur le langage IDL du traitement ODP pour les signatures opérationnelles.

NOTE 2 – Les interfaces de traitement décrites dans la présente Spécification sont techniquement alignées avec le service d'objets de courtage du groupe OMG.

## 8.1 Concordances entre points de vue

### 8.1.1 Concordance avec le point de vue entreprise

Les politiques exprimées dans le point de vue entreprise le sont sous forme de paramètres opérationnels ou d'attributs de courtier dans le point de vue traitement. Certains de ces paramètres opérationnels sont exprimés sous la forme de contraintes.

Chaque contrainte peut être exprimée sous la forme d'une proposition selon laquelle:

- une propriété nommée existe;
- une propriété nommée possède une relation spécifiée avec une valeur déclarée;
- les valeurs de deux propriétés nommées ont une relation spécifiée.

Chaque contrainte peut être mise à la valeur par défaut *true*. Une contrainte peut également être une réunion, une intersection ou une négation d'autres contraintes.

Deux gabarits d'action existent pour les politiques de traitement:

- action de consultation, qui détermine les contraintes de comportement qui s'appliquent pour suivre la politique du courtier;
- action d'arbitrage, dont le résultat est une contrainte applicable à l'exécution d'une opération donnée par combinaison de la politique du client (exprimée sous la forme d'une contrainte ou d'un paramètre d'entrée) et de la politique du courtier (représentée par une contrainte et par certaines valeurs d'attribut ou de propriété).

### 8.1.2 Concordance avec le point de vue information

La spécification d'information définit un graphe orienté dont les nœuds contiennent les partitions d'offres. La seule contrainte visible du point de vue information est que, dans un sens donné, une seule frontière est autorisée entre deux nœuds quelconques. En sens opposés, deux frontières sont autorisées entre deux nœuds.

La spécification de traitement définit les objets-courtiers, c'est-à-dire les objets qui fournissent un service de courtage à une interface. Un objet-courtier peut utiliser un service de courtage offert par un autre courtier, c'est-à-dire qu'il est en liaison avec cet autre courtier.

La relation entre ces deux spécifications de point de vue est prescrite dans la présente Spécification. Une partition d'informations doit correspondre à un objet-courtier et à un seul. La possibilité que de nombreuses partitions correspondent à un seul objet-courtier n'est pas autorisée. Chaque interface du service de courtage possède une seule partition, à partir de laquelle toutes les offres associées à cette interface du service de courtage peuvent être atteintes lors d'une recherche pour importation.

Une frontière du graphe illustrant le point de vue information connecte les partitions qui correspondent à différents objets-courtiers et doit donc correspondre, elle-même, à une liaison entre ces deux courtiers. La partition cible est associée à l'interface du service de courtage qui est visée par cette liaison. Toutes les partitions correspondant à un courtier donné doivent, soit être associées à l'interface du service de courtage, soit être atteignables, dans le graphe d'information, à partir d'une telle partition associée.

## 8.2 Types de concepts et de données

### 8.2.1 Exportateur et importateur

#### 8.2.1.1 Exportateur

Un exportateur signale un service à un courtier. L'exportateur d'un service peut en être le fournisseur ou bien peut le signaler au nom d'une autre entité.

#### 8.2.1.2 Importateur

Un importateur utilise un courtier pour rechercher des services répondant à certains critères. Un importateur peut être le client éventuel d'un service ou peut importer un service au nom d'une autre entité.

### 8.2.2 Types et valeurs neutres du point de vue architectural

#### 8.2.2.1 Code de type

Le langage IDL du traitement ODP n'inclut pas de type de primitive pour représenter des descriptions de type. Les gabarits IDL-ODP indiqués dans la présente Spécification utilisent le terme *TypeCode* pour représenter des descriptions de type.

Lors de la mise en œuvre de cette fonction ODP au moyen d'une infrastructure particulière de traitement ODP, le terme *TypeCode* doit être défini par un type approprié pour représenter des descriptions de type dans cette infrastructure ODP.

NOTE – Pour les infrastructures de l'architecture CORBA, le terme *TypeCode* doit être défini comme suit: «CORBA::TypeCode».

#### 8.2.2.2 Valeur de référence d'interface nulle

Dans la présente Spécification, la valeur «nulle» fait référence à une instance d'interface inexistante.

### 8.2.3 Types de service

#### 8.2.3.1 Informations sur le type de service

Chaque service négocié est associé à un type de service qui représente les informations requises pour décrire un service. Ces informations sont les suivantes:

- type d'interface, qui définit la signature de traitement de l'interface avec le service;
- zéro, un ou plusieurs types de propriété nommés. Normalement, il s'agit des aspects relatifs au comportement, des aspects non fonctionnels et des aspects autres que de traitement, qui ne sont pas intégrés dans la signature de traitement.

Le type de propriété définit le type de valeur de propriété, indique si celle-ci est obligatoire ou si la propriété est en lecture seulement. En d'autres termes, le triplet <nom, type, mode> est associé à un type de propriété où les modes sont les suivants:

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

Un conteneur de types de service est utilisé pour conserver l'information.

```
typedef Object TypeRepository;
```

Un courtier possède un conteneur de types de service associé. Cependant, la présente Spécification n'exige pas l'utilisation d'une interface particulière avec le conteneur de types de service: une telle interface peut éventuellement faire partie de l'objet-courtier proprement dit. Une interface appropriée est spécifiée dans l'Annexe D.

Chaque type de service se trouvant dans un conteneur est identifié par un nom de type de service unique:

```
typedef Istring ServiceTypeName;
```

NOTE – La définition de type de service par chaîne internationale montre qu'il est prévu d'utiliser un jeu de caractères internationaux (non limité au jeu Latin 1).

Un exportateur spécifie le type du service qu'il signale; un importateur spécifie le type du service qu'il recherche.

Les types de service peuvent être mis en relation dans une hiérarchie qui reflète le sous-typage d'interface (par exemple par héritage) et l'agrégation des types de propriété. Cette hiérarchie constitue la base permettant de décider si un service d'un type donné peut être substitué à un service d'un autre type. Ces considérations sont décrites plus complètement dans le modèle ci-après de type de service.

#### 8.2.3.2 Modèle de type de service

Le présent paragraphe correspond à la spécification, du point de vue information, des règles de sous-typage de service exposées dans le 7.2.3. Ces règles sont exprimées dans le présent paragraphe au moyen des termes du langage de traitement.

Le modèle de type de service est illustré par le formalisme BNF suivant:

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

Le mot clé «service» introduit un nouveau nom de type de service. Celui-ci, étant visible aux utilisateurs finals et non seulement aux programmeurs, est internationalisable.

La liste des noms de types de service de base énumère les types de service dont on a déduit le type de service actuel, qui à son tour définit les types de service qui peuvent se substituer à un autre service.

Le mot clé «interface» introduit le nom du type d'interface pour ce service. Ce mot clé est mis en relation, par équivalence ou par dérivation, avec les noms de type d'interface contenus dans chacun des noms de types de service de base.

L'article sur les propriétés est une liste de déclarations de propriété. Chaque déclaration de propriété est marquée du mot clé «property» et peut être précédée des attributs de mode «obligatoire» et/ou «en lecture seulement». Une déclaration de propriété est complétée d'un type de langage IDL et d'un nom de propriété. Un service doit prendre en charge toutes les propriétés de chacun de ses types de service de base. Ces propriétés doivent avoir des types de valeur de propriété identiques et ne doivent perdre aucun attribut de mode de propriété.

Les attributs de mode de propriété ont les connotations suivantes:

- obligatoire: une instance de ce type de service doit fournir une valeur appropriée pour cette propriété lorsqu'elle exporte son offre de service;
- en lecture seulement: si une instance de ce type de service fournit une valeur appropriée pour cette propriété lorsqu'elle exporte son offre de service, la valeur de cette propriété ne doit pas être modifiée par une invocation subséquente de l'opération *Register::modify()*.

Le graphe des puissances des propriétés est représenté sur la Figure 2.

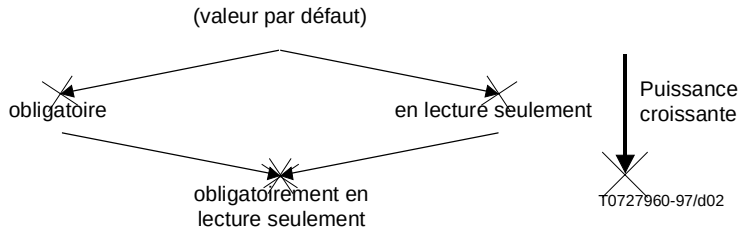



**Figure 2 – Graphe des puissances des propriétés**

En résumé, si une propriété est définie sans modificateurs, elle est facultative (c'est-à-dire qu'une offre du service en cause n'est pas requise pour fournir une valeur de ce nom de propriété; mais si elle est requise, cette offre doit être du type spécifié dans le type de service) et la valeur de la propriété peut être modifiée par la suite. Le modificateur «obligatoire» indique qu'une valeur doit être fournie mais qu'elle pourra être modifiée par la suite. Le modificateur «en lecture seulement» indique que la propriété est facultative mais qu'une fois la valeur donnée, celle-ci ne peut plus être modifiée. La spécification des deux modificateurs indique qu'une valeur doit être fournie et que cette valeur ne pourra plus être modifiée.

Sur la base de l'analyse précédente, on peut déclarer les règles applicables à la conformité des types de service; un type de service $\beta$ est un sous-type du type de service $\alpha$, si, et seulement si:

- le type d'interface associé au type $\beta$ est le type ou le sous-type d'interface associé à $\alpha$;
- toutes les propriétés définies en $\alpha$ sont aussi définies en $\beta$;
- pour toutes les propriétés définies en $\alpha$ comme en $\beta$, le mode de la propriété définie en $\beta$ doit être identique au mode de la propriété définie en $\alpha$ ou lui être supérieur;
- toutes les propriétés définies en $\beta$, qui sont également définies en $\alpha$, doivent avoir, en $\beta$, le même type de valeur de propriété qu'en $\alpha$, ou un sous-type du type de valeur de propriété contenu en $\alpha$.

### 8.2.4 Propriétés

Les propriétés sont constituées de paires <nom, valeur>. Un exportateur déclare les valeurs des propriétés du service qu'il signale. Un importateur peut obtenir ces valeurs à propos d'un service et limiter sa recherche aux offres appropriées, sur la base des valeurs de propriété associées à ces offres.

```
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 Offres de service

Une offre de service est l'information déclarée par un exportateur au sujet du service qu'il signale. Elle contient:

- le nom du type de service;
- une référence à l'interface qui fournit le service;
- zéro, une ou plusieurs valeurs de propriété pour le service.

Un exportateur doit spécifier une valeur pour toutes les propriétés obligatoires spécifiées dans le type de service associé. En outre, un exportateur peut également nommer des valeurs pour des propriétés nommées qui ne sont pas spécifiées dans le type de service. Dans ce cas, le courtier n'est pas obligé d'effectuer la vérification du type des propriétés.

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

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

#### 8.2.5.1 Propriétés modifiables

La valeur d'une propriété contenue dans une offre de service peut, facultativement, être modifiée si:

- le mode de propriété n'est pas, obligatoirement ou facultativement, en lecture seulement;
- le courtier prend en compte la fonction de modification de propriété.

De telles valeurs de propriété peuvent être mises à jour par des opérations de modification explicites envoyées au courtier. Un exportateur peut s'assurer qu'une offre de service est non modifiable en exportant des services dont le type possède des propriétés de lecture seulement. L'opération de modification renverra une exception *opération non mise en œuvre* si un courtier ne supporte pas la fonction de modification de propriété. Un importateur peut également spécifier si un courtier doit prendre en compte, au cours de la mise en concordance, les offres contenant des propriétés modifiables.

#### 8.2.5.2 Propriétés dynamiques

Une offre de service peut, facultativement, contenir des propriétés dynamiques. La valeur d'une propriété dynamique n'est pas contenue chez un courtier mais est obtenue sur demande à partir de l'interface d'un évaluateur de propriété dynamique qui est désigné par l'exportateur du service. C'est-à-dire qu'un certain niveau de visibilité indirecte est requis pour obtenir la valeur d'une propriété dynamique, dont la structure est la suivante:

```
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;
};
```

Cette structure contient l'interface avec l'évaluateur de propriété dynamique, le type de données de la propriété dynamique retournée, et toutes informations supplémentaires dépendant de la mise en œuvre. Le courtier reconnaît cette structure et, lorsque la valeur de la propriété est requise, invoque l'opération *evalDP* à partir de l'interface *DynamicPropEval* appropriée, qui ne possède qu'une seule opération, dont la signature est définie dans la présente Spécification en vue de la portabilité mais dont le comportement n'est pas spécifié. Les seules contraintes imposées sont que la propriété ne soit pas en lecture seulement et que le courtier prenne en charge la fonction de propriété dynamique.

L'utilisation de telles propriétés a des répercussions sur la performance d'un courtier. Un importateur peut spécifier si un courtier doit ou non prendre en compte, au cours de la mise en concordance, les offres contenant des propriétés dynamiques.

### 8.2.6 Identificateur d'offre

Un identificateur d'offre est renvoyé à un exportateur lorsqu'une offre de service est signalée chez un courtier. Cet identificateur désigne l'offre de service exportée et est cité par l'exportateur lorsque celui-ci retire ou modifie son offre (selon le cas). Il n'a de signification que pour le courtier auprès duquel l'offre de service est enregistrée.

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

### 8.2.7 Sélection d'offre

L'espace total réservé à la sélection des offres est théoriquement très grand. Il comprend les offres issues de tous les courtiers mis en liaison. Logiquement, le courtier utilise des politiques pour identifier l'ensemble S1 des offres de service à examiner. Le type de service et la contrainte sont ensuite appliqués à l'ensemble S1 pour obtenir l'ensemble S2, qui répond au type de service et à la contrainte. Cet ensemble est ensuite ordonné selon les préférences avant de renvoyer les offres à l'importateur.

#### 8.2.7.1 Langage normalisé de contrainte

Au moyen du type de service et de la contrainte, les importateurs sélectionnent l'ensemble des offres de service qui présentent un intérêt pour eux. La contrainte est une expression structurée qui est conforme à un langage de contrainte.

La présente Recommandation | Norme internationale définit le langage normalisé et obligatoire qui est nécessaire pour assurer l'interfonctionnement des courtiers. L'Annexe B définit la syntaxe et la puissance expressive de ce langage de contrainte. Celui-ci est utilisé pour écrire des expressions normalisées de contrainte.

```
typedef Istring Constraint;
```

Ses principales caractéristiques sont les suivantes:

– types de valeur de propriété: les manipulations sont limitées aux types *int, float, fixed, boolean, Istring/string, Ichar/char* et à leurs séquences. Les types en mode caractère sont ordonnés au moyen de la séquence de collationnement qui est appliquée au jeu de caractères en cause. Les types extérieurs à cette étendue ne peuvent être soumis qu'à l'opérateur «exists»;

– littéraux: dans la contrainte, les littéraux sont dynamiquement forcés selon les besoins des propriétés avec lesquelles ils fonctionnent. Les littéraux peuvent contenir des chaînes de type international;

– opérateurs: comparaison, connexion booléenne, «in_set», sous-chaîne, opérateurs arithmétiques, existence de propriété.

Si l'on utilise un langage de contrainte de propriété (qui est hors du domaine d'application de la présente Spécification), le nom et la version du langage de contrainte utilisé sont placés entre «guillemets» au début de l'expression de la contrainte. Le reste de la chaîne n'est pas interprété par un courtier qui ne prend pas en charge le langage de contrainte non normalisé qui est cité.

#### 8.2.7.2 Préférences

Les préférences sont logiquement appliquées à l'ensemble des offres correspondant à l'application du type de service, de l'expression de contrainte et de diverses politiques. L'application des préférences peut être considérée comme étant la détermination de l'ordre de retour, à l'importateur, des offres mises en concordance.

**typedef Istring Preference;**

La chaîne de préférence peut être considérée comme étant composée de deux parties. La première partie peut contenir un des mots clés suivants (les majuscules sont significatives pour ces mots):

**max min with random first**

L'interprétation de la deuxième partie dépend de la première; elle peut être vide. Les préférences sont décrites ci-après.

| Préférence | Description |
|---|---|
| Expression avec *max* | Cette expression est numérique. Les offres mises en concordance sont retournées dans l'ordre décroissant de l'expression. |
| Expression avec *min* | Cette expression est numérique. Les offres mises en concordance sont retournées dans l'ordre croissant de l'expression. |
| Expression avec *with* | Cette expression est de type contrainte. Les offres mises en concordance sont mises dans un ordre tel que celles qui ont la valeur TRUE précèdent celles qui ont la valeur FALSE. |
| random | L'ordre de retour des offres mises en concordance est conforme à l'algorithme suivant: sélection d'une offre au hasard dans l'ensemble des offres mises en concordance; sélection d'une autre offre au hasard dans l'ensemble restant des offres mises en concordance; ..., sélection de la dernière offre restante. |
| first | L'ordre de retour des offres mises en concordance est celui de la découverte des offres. |

Si aucune préférence n'est spécifiée, la préférence par défaut de type *first* s'applique d'abord. Aucune combinaison de préférences n'est autorisée.

L'expression associée aux valeurs *max, min* et *with* peut se rapporter à des propriétés associées aux offres compatibles. Lors de l'application d'une expression de préférence à l'ensemble des offres qui correspondent au type de service et à l'expression de contrainte, cet ensemble d'offres est partitionné en deux groupes:

a) un groupe d'offres pour lequel l'expression de préférence peut être évaluée (dans l'ordre indiqué par les valeurs *min, max, with*);

b) un groupe d'offres pour lequel l'expression de préférence ne peut pas être évaluée (par exemple, l'expression de préférence se rapporte à un nom de propriété qui est facultatif pour ce type de service).

Les offres sont retournées à l'importateur dans l'ordre des préférences du premier groupe, puis dans l'ordre du second groupe.

Si un langage de préférence non normalisé (hors du domaine d'application de la présente Spécification) est utilisé, le nom et la version de ce langage sont placés entre «guillemets» au début de l'énoncé de la préférence. Le reste de la chaîne n'est pas interprété par un courtier qui ne prend pas en charge le langage non normalisé qui est cité.

#### 8.2.7.3 Liaisons

Les liaisons représentent des voies de propagation pour les interrogations issues d'un courtier-cible. Chaque liaison correspond à une frontière dans un graphe de courtage, dont les nœuds sont des courtiers. Une liaison décrit la connaissance que possède un courtier au sujet d'un autre service de courtage qu'il utilise. Elle donne également l'information relative au moment de la propagation ou du réacheminement d'une opération vers le courtier-cible. Une liaison est assortie des informations suivantes:

– interface de consultation fournie par le courtier-cible, qui prend en charge l'opération d'interrogation;

– interface d'enregistrement fournie par le courtier-cible, qui prend en charge l'opération de résolution;

– comportement par défaut de suivi de liaison, qui peut être utilisé et transmis lorsqu'un importateur ne spécifie pas de politique de suivi de liaison;

– comportement de limitation de suivi de liaison, qui a priorité sur la règle de suivi de liaison d'un importateur si la requête de celui-ci dépasse la limite fixée par la liaison.

Le langage IDL du groupe OMG pour l'option de suivi est défini comme suit:

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

où:

– le terme «local_only» indique que la liaison n'est jamais suivie, sauf si elle est explicitement nommée dans une opération;

– le terme «if_no_local» indique que la liaison n'est suivie que s'il n'existe pas d'offres locales répondant à l'interrogation;

– et le terme «always» indique que la liaison est toujours suivie, sauf si elle est outrepassée par une autre politique.

Ces valeurs sont ordonnées comme suit:

**local_only < if_no_local < always**

Le langage IDL du groupe OMG pour l'information de liaison est défini comme suit:

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

Les informations ci-dessus sont déterminées pour chaque liaison au moment de la création de celle-ci. Un nom est donné à la liaison au moment de sa création. Ce nom identifie la liaison de manière univoque chez un courtier.

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

Une liaison n'a qu'un seul sens. Seul le courtier-source a la visibilité directe d'une liaison: c'est lui qui en supporte l'interface.

Des informations complémentaires peuvent être associées à une liaison afin de décrire les caractéristiques du service de courtage-cible qui sont perçues par le courtier-source.

#### 8.2.7.4 Politiques

Les politiques donnent des informations visant à influencer le comportement du courtier au moment de l'exécution du service. Les politiques sont représentées par des paires nom-valeur.

```
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;
```

Certaines politiques ne peuvent pas être outrepassées, tandis que d'autres politiques s'appliquent en l'absence d'autres informations et peuvent être outrepassées. On peut ranger les politiques dans deux catégories:

– celles dont le domaine d'application englobe l'étendue d'une recherche;

– celles qui déterminent la fonction appliquée à une opération.

Différentes politiques sont associées à différents rôles lors de l'exécution de la fonction de courtage. Ces rôles, qui sont utilisés dans la colonne «Rôle» des tableaux ci-dessous, sont les suivants:

T = courtier (*trader*)

L = liaison

I = importation

Les politiques sont développées du 8.2.7.5 au 8.2.7.9.

#### 8.2.7.4.1 Politiques de détection normalisées

Les politiques normalisées de détection dans un domaine de visibilité sont énumérées ci-dessous:

| Nom | Rôle | Type IDL | Description |
|---|---|---|---|
| def_search_card | T | nombre long non signé | Limite supérieure par défaut des offres à rechercher; politique utilisée si aucune politique search_card n'est spécifiée. |
| max_search_card | T | nombre long non signé | Limite supérieure maximale des offres à rechercher. |
| search_card | I | nombre long non signé | Limite supérieure désignée des offres à rechercher; cette politique sera outrepassée par la politique max_search_card. |
| def_match_card | T | nombre long non signé | Limite supérieure par défaut des offres mises en concordance à ordonner; politique utilisée si aucune politique match_card n'est spécifiée. |
| max_match_card | T | nombre long non signé | Limite supérieure maximale des offres mises en concordance à ordonner. |
| match_card | I | nombre long non signé | Limite supérieure désignée des offres à ordonner; cette politique sera outrepassée par la politique max_match_card. |
| def_return_card | T | nombre long non signé | Limite supérieure par défaut des offres ordonnées à retourner; politique utilisée si aucune politique return_card n'est spécifiée. |
| max_return_card | T | nombre long non signé | Limite supérieure maximale des offres ordonnées à retourner. |
| return_card | I | nombre long non signé | Limite supérieure désignée des offres ordonnées à retourner; cette politique sera outrepassée par la politique max_return_card. |
| def_hop_count | T | nombre long non signé | Limite supérieure par défaut de la profondeur des liaisons à croiser si le décompte des sauts n'est pas spécifié. |
| max_hop_count | T | nombre long non signé | Limite supérieure par défaut de la profondeur des liaisons à croiser. |
| hop_count | I | nombre long non signé | Limite supérieure désignée de la profondeur des liaisons à croiser; cette politique sera outrepassée par la politique max_hop_count du courtier. |
| def_pass_on_follow_rule | L | option de suivi | Comportement par défaut de suivi de liaison à transmettre pour une liaison particulière si un importateur ne spécifie pas sa règle de suivi de liaison. Il ne doit pas dépasser la règle de limitation de suivi de liaison. |
| limiting_follow_rule | L | option de suivi | Comportement de limitation de suivi d'une liaison particulière. |
| max_link_follow_policy | T | option de suivi | Limite supérieure de la valeur d'une règle de limitation du suivi d'une liaison au moment de la création ou de la modification d'une liaison. |
| def_follow_policy | T | option de suivi | Comportement par défaut de suivi de liaison pour un courtier particulier. |
| max_follow_policy | T | option de suivi | Politique de limitation du suivi de liaison pour toutes les liaisons du courtier; cette politique outrepasse les politiques de liaison comme d'importation. |
| link_follow_rule | I | option de suivi | Comportement de suivi de liaison désigné; cette politique sera outrepassée par la politique max_follow_policy du courtier et par la règle de limitation de suivi de liaison. |
| starting_trader | I | nom de courtier | Un importateur délimite la visibilité de sa recherche en désignant un courtier distant comme point de départ de l'interrogation; un courtier est obligé de faire suivre cette requête vers une liaison, même si le comportement de celle-ci n'est que local. |
| request_id | I | séquence d'octets | Identificateur d'opération d'interrogation lancée par un courtier-source jouant le rôle d'importateur dans une liaison; un courtier n'est pas obligé de produire un identificateur mais est obligé de le retransmettre sur une liaison. |
| exact_type_match | I | opérateur booléen | Si la valeur du booléen est TRUE, seules les offres du type de service exact qui a été spécifié par l'importateur sont prises en considération; si la valeur est FALSE (ou non spécifiée), seules sont considérées les offres de n'importe quel type de service, conforme au type de service de l'importateur. |

Les types de langage IDL-ODP pour les noms de courtier d'interrogation et pour les séquences d'octets d'identification sont les suivants:

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

Les résultats reçus par un importateur sont influencés par les politiques de détection. Les politiques *hop_count* et *link_follow* déterminent le domaine de visibilité des courtiers à visiter. N1 est l'espace total réservé aux offres de service. Celles qui ont des services d'un type conforme sont rassemblées dans l'ensemble N2; l'étendue réelle de l'ensemble N2 peut être encore limitée par les politiques de cardinalité des recherches. Des contraintes sont appliquées à l'ensemble N2 afin de produire un ensemble d'offres N3 qui répond aussi bien aux conditions de type de service qu'aux contraintes. L'ensemble N3 peut encore être limité par les politiques de cardinalité des concordances. L'ensemble N3 est ensuite ordonné au moyen des préférences de façon à produire l'ensemble N4. L'ensemble final N5 des offres retournées à l'importateur peut être encore réduit par les politiques de cardinalité des retours.

Cette procédure est illustrée par le schéma de la Figure 3, où |N1| >= |N2| >= |N3| = |N4| >= |N5|.

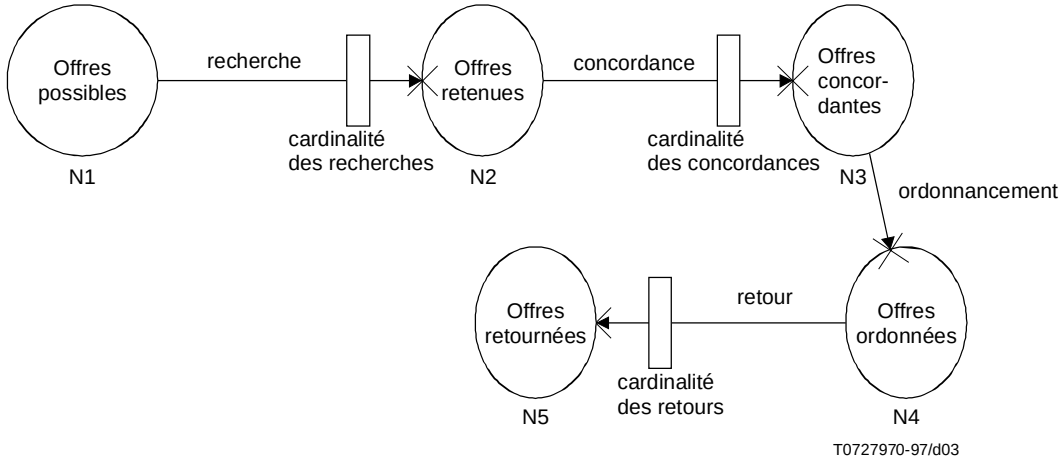



**Figure 3 – Vue en pipeline des étapes d'interrogation d'un courtier et application de contraintes de cardinalité**

#### 8.2.7.4.2 Politiques normalisées relatives aux capacités supportées

Un courtier a la possibilité de prendre en charge trois capacités: l'offre de procuration, les propriétés dynamiques et la modification d'offres. Si un courtier ne supporte pas une capacité, un importateur ne peut pas outrepasser celle-ci avec son paramètre de politique. Si cependant un courtier supporte une capacité et qu'un importateur ne souhaite pas prendre en considération les offres qui nécessitent une telle fonctionnalité, ce courtier doit respecter le choix de l'importateur.

Les politiques normalisées relatives aux capacités supportées sont énumérées ci-dessous:

| Nom | Rôle | Type IDL | Description |
|---|---|---|---|
| supports_modifiable_properties | T | opérateur booléen | Cette politique détermine si le courtier supporte la modification d'une propriété. |
| use_modifiable_properties | I | opérateur booléen | Cette politique détermine si les offres ayant des propriétés modifiables doivent être prises en considération lors de la recherche. |
| supports_dynamic_properties | T | opérateur booléen | Cette politique détermine si le courtier supporte les propriétés dynamiques. |
| use_dynamic_properties | I | opérateur booléen | Cette politique détermine si les offres ayant des propriétés dynamiques doivent être prises en considération lors de la recherche. |
| supports_proxy_offers | T | opérateur booléen | Cette politique détermine si le courtier supporte les offres de procuration. |
| use_ proxy_offers | I | opérateur booléen | Cette politique détermine s'il y a lieu de prendre en considération les offres de procuration lors de la recherche. |

#### 8.2.7.5 Politiques de courtage

Les politiques peuvent être fixées pour un courtier comme un tout. Les politiques de courtage sont définies sous la forme d'attributs de l'objet-courtier. Elles sont initialement spécifiées lors de la création du courtier et peuvent être modifiées/interrogées via l'interface d'administration. Via son interface de consultation, un importateur peut interroger ces politiques de courtage. Via son interface d'enregistrement, un exportateur peut interroger des politiques relatives à une capacité de courtage supportée.

#### 8.2.7.6 Comportement de suivi de liaison

Chaque liaison offerte par un courtier possède ses propres politiques de comportement de suivi. Un courtier possède une politique de limitation du suivi de liaison, *max_follow_policy*, qui a priorité sur toutes les liaisons de ce courtier pour toute interrogation donnée. Les politiques relatives au comportement de suivi de liaison sont spécifiées pour chaque liaison, lors de la création de celle-ci. On peut interroger/modifier ces politiques (*def_pass_on_follow_rule* et *limiting_follow_rule*) via l'interface de liaison. Leurs valeurs possibles sont limitées par une autre politique de courtage, *max_link_follow_policy*, au moment de la création ou de la modification. Dans une interrogation, un importateur peut spécifier une règle de suivi de liaison. En l'absence d'une telle règle, c'est la politique de comportement par défaut pour le suivi de liaison, (*def_follow_policy*) du courtier qui est utilisée.

Après avoir recherché ses offres locales en réponse à une interrogation, un courtier doit décider s'il y a lieu de propager cette interrogation sur ses liaisons et déterminer, si tel est le cas, quelle valeur il doit transmettre dans l'argument *politiques* pour la règle de suivi de liaison.

L'expression en langage IDL-ODP de l'option de suivi de liaison est spécifiée au 8.2.7.3.

La politique de suivi d'une liaison particulière est donc:

```
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)
```

c'est-à-dire que si cette valeur est «if_no_local» et qu'il n'y ait pas d'offres locales pour répondre à l'interrogation, l'opération d'interrogation imbriquée est effectuée. Si la valeur est «always», l'interrogation encastrée est effectuée.

Si une interrogation imbriquée est autorisée par la règle ci-dessus, le module suivant détermine la valeur de la politique «*link_follow_rule*» à transmettre au courtier lié.

```
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 Politiques d'importateur

Un importateur peut spécifier zéro, une ou plusieurs politiques d'importateur dans son paramètre «politiques». Si une politique d'importateur n'est pas spécifiée, le courtier utilise sa politique par défaut. Si une politique d'importateur dépasse les valeurs limites de politique fixées par le courtier, celui-ci remplace les valeurs attendues par l'importateur par la valeur limite de sa politique.

Si un paramètre de politique «starting_trader» est utilisé, les mises en œuvre des courtiers doivent placer ce paramètre de politique comme premier élément de la séquence d'envoi de leur demande d'interrogation aux courtiers liés.

#### 8.2.7.8 Politiques d'exportateur

La présente Spécification ne spécifie pas de politique d'exportateur.

#### 8.2.7.9 Politiques de création de liaison

Au moment de la création d'une liaison, les règles de suivi par défaut et de limitation de suivi de liaison sont spécifiées. Ces règles peuvent être contraintes par la politique *max_link_follow_policy* du courtier.

Le courtier contrôle d'abord que la règle par défaut est moins ou autant restrictive que la règle de limitation. Si ce n'est pas le cas, une exception est propagée. Il compare ensuite la règle de limitation à la politique de suivi du courtier (*max_link_follow_policy*), propageant de nouveau une exception si la règle de limitation est plus restrictive que la politique *max_link_follow_policy* du courtier.

### 8.2.8 Mécanismes d'interfonctionnement

#### 8.2.8.1 Limitation des croisements de liaisons

La nature flexible des liens de courtage permet de produire des graphes orientés arbitraires, ce qui peut apporter deux types de problèmes:

- un courtier donné peut être visité plusieurs fois au cours d'une même recherche, du fait qu'il apparaît dans plusieurs chemins issus d'un même courtier (c'est-à-dire dans des ensembles distincts de frontières connectées);
- des boucles peuvent se produire. L'exemple le plus courant est celui où deux espaces de courtage, préalablement disjoints, décident de se réunir en échangeant des liaisons. Il peut en résulter que le premier courtier propage une interrogation vers le second et que cette interrogation lui revienne immédiatement via la liaison inverse.

Pour s'assurer qu'une recherche n'entre pas dans une boucle sans fin, on fait appel à une politique de compte de sauts (*hop_count*) afin de limiter la profondeur des liaisons susceptibles de propager une recherche. Le décompte de sauts est décrémenté d'une unité avant de propager une interrogation vers d'autres courtiers. La propagation d'une recherche aboutit au courtier lorsque le décompte de sauts atteint zéro.

Pour éviter la revisite improductive d'un courtier donné lors de l'exécution d'une interrogation, un identificateur de requête peut être émis par un courtier-source pour chaque opération d'interrogation qu'il met en propagation vers un courtier-cible. L'attribut de courtier *request_id_stem* est utilisé pour former l'identificateur de requête.

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

Un courtier est autorisé à garder en mémoire les identificateurs de requête pour toutes les récentes opérations d'interrogation pour interfonctionnement qu'il a été prié d'exécuter. Lorsqu'il reçoit une opération d'interrogation, un tel courtier vérifie ces données de référence et ne donne suite à la demande que s'il s'agit de la première instance de cette opération.

Pour que ce processus fonctionne, l'administrateur d'un ensemble de courtiers fédérés doit avoir initialisé les attributs *request_id_stem* avec des valeurs sans chevauchement.

L'identificateur de requête est acheminé vers le courtier-cible dans un paramètre de politique d'importateur au sujet de l'opération d'interrogation. Si le courtier-cible ne supporte pas l'utilisation de la politique d'identification de requête, ce courtier n'a pas besoin de traiter cette identification mais doit la faire suivre au prochain courtier lié si la recherche se propage encore.

#### 8.2.8.2 Exemple d'interrogations fédérées

Pour propager une demande d'interrogation dans un graphe de courtage, chaque courtier-source joue le rôle de client à l'interface de consultation du courtier-cible et transmet l'opération d'interrogation de son client vers son courtier-cible.

La Figure 4 donne un exemple de recherche séquentielle afin d'illustrer la modification du paramètre de décompte de sauts lorsqu'une demande d'interrogation traverse un ensemble de courtiers liés dans un graphe de courtage. L'on part du principe que les politiques de suivi de liaison chez les courtiers produiront un comportement de suivi de type «*always*».

a) Une demande d'interrogation est invoquée à l'interface de courtage de l'interface T1 avec, chez l'importateur, une politique de décompte de sauts exprimée sous la forme *hop_count* = 4. La politique de détection à l'interface T1 pour le courtier donne une valeur maximale de décompte de sauts *max_hop_count* = 5. Le décompte de sauts résultant, appliqué à la recherche (après l'action d'arbitrage qui combine la politique du courtier et la politique de l'importateur) est: *hop_count* = 4.

b) L'on part du principe qu'aucune concordance n'est trouvée à l'interface T1 et que la politique de suivi qui en résulte est de type «*always*». C'est-à-dire que l'interface T1 doit faire suivre la requête à l'interface T3. Une politique modifiée de décompte de sauts chez l'importateur est utilisée avec la valeur *hop_count* = 3. La politique de détection du courtier local à l'interface T3 utilise la valeur *max_hop_count* = 1 et produit l'identificateur *T3_Request_id* pour éviter des recherches répétées ou cycliques des mêmes courtiers. La politique de détection résultante, appliquée à la recherche à l'interface T3, a la valeur de décompte *hop_count* = 1 et l'identificateur de demande à T3 est mis en mémoire.

c) En supposant qu'aucune concordance ne soit trouvée à l'interface T3 et que la politique de suivi résultante soit «*always*», le paramètre de détection modifié pour la demande d'interrogation à l'interface T4 aura les valeurs suivantes: *hop_count* = 0 et *request_id* = *T3_Request_id*.

d) En supposant qu'aucune concordance ne soit trouvée à l'interface T4, même si le décompte maximal de sauts à cette interface a la valeur 4, la recherche n'est pas propagée plus loin. Un résultat d'insuccès sera renvoyé à l'interface T3, puis à l'interface T1 et finalement à l'utilisateur de l'interface T1.

Il est évident que si une demande d'interrogation est traitée avec succès chez un des courtiers appartenant au chemin de recherche lié, la liste des offres de service adaptées sera renvoyée à l'utilisateur d'origine. Les politiques de suivi de liaison détermineront si la demande d'interrogation sera propagée dans le reste du graphe de courtage; dans ce cas, lorsqu'on suppose que la politique est de type «*always*», l'interrogation continuera à passer par les sites de tous les courtiers, dans les limites de la politique de décompte de sauts.

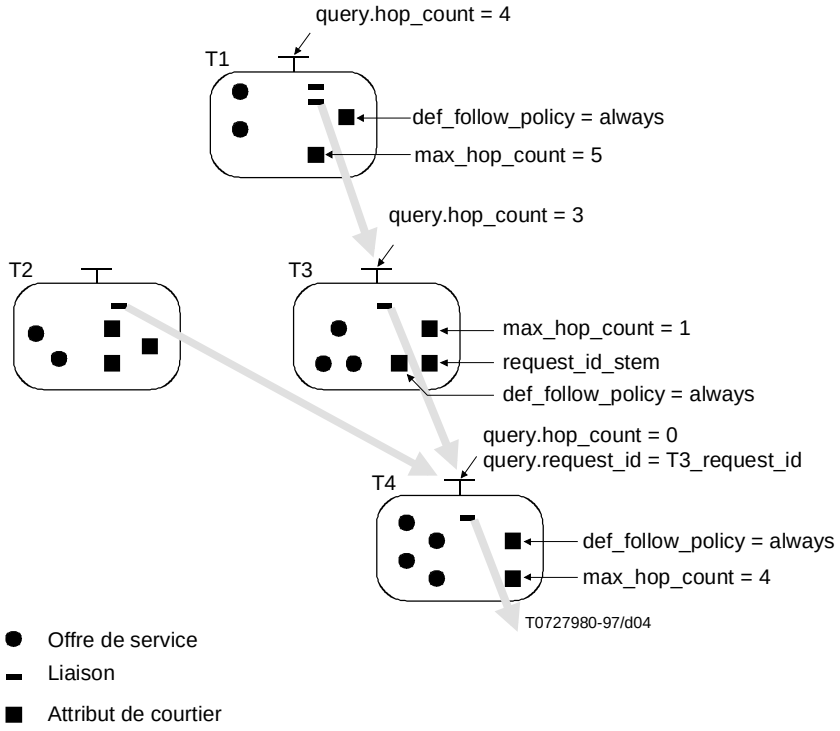



**Figure 4 – Flux d'une interrogation dans un graphe de courtier**

#### 8.2.8.3 Offres de procuration

Une offre de procuration est au croisement d'une offre de service et d'une sorte de liaison restreinte. Comme elle comporte le type de service et les propriétés d'une offre de service, elle est mise en concordance de la même façon. Si toutefois l'offre de procuration correspond aux besoins de l'importateur, la demande d'interrogation (modifiée) est renvoyée à l'interface de consultation associée à cette offre de procuration, plutôt que de renvoyer les détails de l'offre.

```
typedef Istring ConstraintRecipe;

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

Si l'interrogation d'un importateur se traduit par une offre de procuration, le courtier possédant cette offre de procuration effectue une interrogation imbriquée auprès du courtier impliqué par l'offre de procuration, avec les paramètres suivants:

- le paramètre *Type initial* est transmis sans changement;
- un nouveau paramètre *Contrainte* est construit selon la recette de contrainte associée à l'offre de procuration;
- le paramètre *Préférence initiale* est transmis sans changement;
- un nouveau paramètre *Politiques* est construit par adjonction – au paramètre *Politiques* initial – des politiques à transmettre qui sont associées à l'offre de procuration;
- le paramètre *Propriétés souhaitées* initial est transmis sans changement;
- le courtier appelant fournit une valeur du paramètre «how_many» correspondant au nombre d'offres à renvoyer, telle qu'elle soit compatible avec ses propres contraintes de ressources.

Les offres de procuration sont une façon commode d'insérer dans le système de courtage un système de legs d'objets. Il permet aux clients de consulter ces «objets» en s'adaptant à l'offre de procuration; l'appel imbriqué adressé au courtier-application permet, au moyen de l'expression de contrainte récrite et des politiques additionnelles jointes au paramètre de politique initial, d'effectuer une création dynamique d'une instance de service encapsulant l'objet à léguer. Une autre façon d'utiliser les offres de procuration consiste à signaler un atelier de services comme une offre de procuration: l'appel imbriqué adressé à l'atelier provoque la fabrication d'une nouvelle instance du service demandé.

Une interrogation peut avoir été associée à une offre de procuration en raison d'une valeur particulière prise par une propriété déjà associée à cette offre de procuration. Il est impératif que toute offre renvoyée par le courtier-application en réponse à une interrogation imbriquée ait la même valeur pour cette propriété, de façon à ne pas violer les attentes du client concernant la contrainte.

Un courtier n'est pas tenu de gérer la fonction d'offre de procuration. Mais si un courtier peut la prendre en charge, il doit fournir l'interface de procuration pour l'exportation, pour le retrait et pour la description des offres de procuration. Un importateur peut spécifier si un courtier doit ou non prendre en compte les offres de procuration au cours de la mise en concordance.

### 8.2.9 Attributs de courtier

Chaque courtier a ses propres caractéristiques, ses propres politiques relatives aux fonctions prises en charge, et ses propres politiques pour détecter le domaine de visibilité d'une recherche. Ces caractéristiques et politiques sont définies comme des attributs de courtier. Ces attributs sont énumérés ci-dessous.

| Nom | Type IDL-ODP | Description |
|---|---|---|
| def_search_card | nombre long non signé | Limite supérieure par défaut des offres à rechercher pour une opération d'interrogation. |
| max_search_card | nombre long non signé | Limite supérieure maximale des offres à rechercher pour une opération d'interrogation. |
| def_match_card | nombre long non signé | Limite supérieure par défaut des offres mises en concordance à ordonner lors de l'application d'un critère de préférence. |
| max_match_card | nombre long non signé | Limite supérieure maximale des offres mises en concordance à ordonner lors de l'application d'un critère de préférence. |
| def_return_card | nombre long non signé | Limite supérieure par défaut des offres ordonnées à renvoyer à un importateur. |
| max_return_card | nombre long non signé | Limite supérieure maximale des offres ordonnées à renvoyer à un importateur. |
| def_hop_count | nombre long non signé | Limite supérieure par défaut de la profondeur des liaisons à croiser. |
| max_hop_count | nombre long non signé | Limite supérieure maximale de la profondeur des liaisons à croiser. |
| def_follow_policy | option de suivi | Comportement par défaut de suivi de liaison pour un courtier donné. |
| max_follow_policy | option de suivi | Politique de limitation de suivi de liaison pour toutes les liaisons du courtier – outrepasse aussi bien les politiques de liaison que les politiques d'importateur. |
| max_link_follow_policy | option de suivi | Politique de suivi de liaison la moins contraignante qui soit autorisée lors de la création de nouvelles liaisons. |
| supports_modifiable_properties | opérateur booléen | Détermine si le courtier prend en charge la modification d'une propriété. |
| supports_dynamic_properties | opérateur booléen | Détermine si le courtier prend en charge les propriétés dynamiques. |
| supports_proxy_offers | opérateur booléen | Détermine si le courtier prend en charge les offres de procuration. |
| max_list | nombre long non signé | Cet attribut détermine la limite supérieure de toute liste renvoyée par le courtier, à savoir: le paramètre *Offres renvoyées* contenu dans l'interrogation et l'opération *next-n* contenue dans l'itérateur d'offre et dans l'itérateur d'identificateur d'offre. |
| type_repos | conteneur | Interface avec le conteneur de types de service du courtier. |
| request_id_stem | séquence d'octets | Identification du courtier, à utiliser comme radical pour la production d'un identificateur de demande d'interrogation d'un courtier à un autre. |

Ces attributs sont initialement spécifiés lors de la création d'un courtier; ils peuvent être modifiés ou interrogés via l'interface d'administration.

## 8.3 Exceptions

La présente Spécification définit les exceptions propagées par les opérations. Les exceptions sont paramétrées de façon à indiquer la source de l'erreur. Les segments en langage IDL-ODP ci-dessous se rapportent à certaines des définitions de type indiquées dans l'article 2.

Lorsque de multiples conditions d'exception apparaissent, une seule exception est propagée. Le choix de l'exception à propager est fonction de la mise en œuvre.

### 8.3.1 Exceptions pour le module CosTrading

#### 8.3.1.1 Exceptions utilisées dans plusieurs interfaces

```
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 Exceptions additionnelles pour l'interface de consultation

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

```
exception IllegalPolicyName {
        PolicyName name;
};

exception PolicyTypeMismatch {
        Policy the_policy;
};

exception InvalidPolicyValue {
        Policy the_policy;
};
```

#### 8.3.1.3 Exceptions additionnelles pour l'interface d'enregistrement

```
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 Exceptions additionnelles pour l'interface de liaison

```
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 Exceptions additionnelles pour l'interface d'offre de procuration

```
exception IllegalRecipe {
        ConstraintRecipe recipe;
};

exception NotProxyOfferId {
        OfferId id;
};
```

### 8.3.2 Exceptions pour le module CosTradingDynamic

Il n'existe qu'une seule interface d'évaluation de propriété dynamique dans ce module. Cette interface n'a qu'une seule opération, qui propage l'exception suivante:

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

### 8.3.3 Exceptions pour le module CosTradingRepos

Il n'existe qu'une seule interface de conteneur de type de service dans ce module. Les exceptions ci-après, spécifiques de cette interface, peuvent être propagées:

```
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 abstraites

De façon à permettre la construction de courtiers possédant divers moyens de prise en charge des différentes interfaces de courtier, la présente Spécification définit plusieurs interfaces abstraites à partir desquelles on peut déduire chacune des interfaces fonctionnelles du service d'objets de courtage (consultation, enregistrement, liaison, procuration et administration). Chacune de ces interfaces abstraites est décrite ci-dessous.

### 8.4.1 Composantes de courtier

```
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 fonctionnalité d'un courtier peut être configurée par composition des interfaces définies de façon à obtenir un certain nombre de combinaisons prescrites. L'opération de composition n'est pas modélisée par héritage mais par interfaces multiples avec un objet. Si l'on considère l'une de ces interfaces, il faut trouver le moyen de trouver les autres interfaces associées. C'est pour faciliter cette recherche que chaque interface fonctionnelle de courtier est déduite de l'interface relative aux composantes d'objet-courtier.

L'interface relative aux composantes d'objet-courtier contient cinq attributs en lecture seulement, qui offrent chacun le moyen d'obtenir une référence d'objet spécifique.

L'accès à ces attributs doit renvoyer une référence d'objet nulle si le service de courtage en question ne prend pas en charge cette interface particulière.

### 8.4.2 Attributs de support

```
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;
};
```

En plus de la capacité de choix sélectif, par une réalisation de courtier, des interfaces fonctionnelles à prendre en charge, une réalisation de courtier peut aussi choisir de ne pas prendre en charge des propriétés modifiables, des propriétés dynamiques, et/ou des offres de procuration. On peut déterminer la capacité prise en charge par une réalisation de courtier en recherchant les attributs en lecture seulement de cette interface.

Le conteneur de types de service utilisé par la réalisation de courtier peut également être obtenu à partir de cette interface.

### 8.4.3 Attributs d'importation

```
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;
};
```

Chaque courtier est configuré avec des valeurs par défaut et avec des valeurs maximales pour certaines contraintes de cardinalité et de suivi de liaison qui s'appliquent aux interrogations. Les valeurs de ces contraintes peuvent être obtenues par recherche des attributs de cette interface.

### 8.4.4 Attributs de liaison

```
interface LinkAttributes {

    readonly attribute FollowOption max_link_follow_policy;
};
```

Lorsqu'un courtier crée une nouvelle liaison ou modifie une liaison existante, l'attribut de la politique *max_link_follow* déterminera le comportement le moins contraignant qui sera autorisé pour la liaison. La valeur de cette contrainte sur la création et la modification d'une liaison peut être obtenue à partir de cette interface.

## 8.5 Interfaces fonctionnelles

Ce paragraphe décrit les cinq interfaces fonctionnelles pour un service d'objet-courtier: consultation, enregistrement, liaison, administration et procuration. Les deux interfaces d'itérateur, nécessaires pour ces interfaces fonctionnelles, sont également décrites.

### 8.5.1 Consultation

```
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 Opération d'interrogation

##### 8.5.1.1.1 Signature

```
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 Fonction

L'opération d'interrogation est le moyen qui permet à un objet d'obtenir des références à d'autres objets fournissant des services répondant à ses besoins.

Le paramètre «type» achemine le type de service requis. C'est un élément clé de l'objectif principal du courtage, qui est de constituer une introduction pour de futures interactions entre importateurs et exportateurs, quel que soit le type de service. En indiquant un type de service, l'importateur sous-entend le type d'interface souhaité ainsi qu'un domaine de discours pour parler des propriétés du service. Si la chaîne représentant le paramètre «type» ne suit pas les règles applicables aux identificateurs de type de service, une exception pour type de service incorrect est propagée. Si le «type» est syntaxiquement correct mais n'est pas reconnu en tant que type de service à l'intérieur du domaine de courtage, une exception pour type de service inconnu est propagée.

Le courtier peut renvoyer une offre pour un service appartenant à un sous-type du «type» requis. Le sous-typage des services est traité au 8.2.3 ci-dessus. Un sous-type de service possède toutes les propriétés obligatoires de ses supertypes. Cela garantit qu'une interrogation bien formée pour le «type» sera également une interrogation bien formée pour n'importe lequel de ses sous-types. Si toutefois l'importateur spécifie la politique de vérification du fait que la concordance exacte des types a la valeur TRUE, seules seront renvoyées les offres ayant le type (et non le sous-type) de service exactement demandé.

La contrainte «constr» est le moyen qui permet à l'importateur de déclarer les caractéristiques d'un service qui ne sont pas associées à la signature de l'interface. Ces caractéristiques traitent du comportement de traitement du service désiré, des aspects non fonctionnels, et des aspects autres que le traitement (comme l'organisation détenant les objets qui fournissent le service). Un importateur a toujours la garantie que toute offre renvoyée répondra à la contrainte de concordance au moment de l'importation. Si la contrainte «constr» n'obéit pas aux règles syntaxiques applicables à l'expression d'une contrainte correcte (telles que définies dans l'Annexe B), une exception pour contrainte incorrecte est propagée.

Le paramètre «pref» est utilisé pour ordonner les offres qui suivent la contrainte «constr», de manière que les offres renvoyées par le courtier soient mises dans l'ordre d'intérêt maximal pour l'importateur. Si le paramètre «pref» ne suit pas les règles syntaxiques applicables à l'expression d'une préférence correcte, une exception pour préférence incorrecte est propagée.

Le paramètre «policies» (politiques) permet à l'importateur de spécifier la façon dont la recherche doit être exécutée et non pas la sorte de services qu'il y a lieu de trouver au cours de la recherche. On peut considérer cela comme un paramétrage des algorithmes situés à l'intérieur de la réalisation du courtier. Les paramètres de politique se présentent sous la forme de séquences de paires nom-valeur. Les noms mis à la disposition d'un importateur dépendent de la réalisation du courtier. Certains noms sont cependant normalisés, lorsqu'ils ont une incidence sur l'interprétation d'autres paramètres ou lorsqu'ils peuvent influencer la mise en liaison et en fédération de courtiers. Si un nom de politique contenu dans ce paramètre ne suit pas les règles syntaxiques applicables aux noms de politique légaux, une exception pour nom de politique incorrect est propagée. Si le type de la valeur associée à une politique diffère de celui qui est spécifié dans la présente Spécification, une exception pour discordance de type de politique est propagée. Si le traitement subséquent d'une valeur de politique donne des erreurs (par exemple une malformation de la valeur de politique d'un courtier de départ), une exception pour valeur de politique invalide est propagée. Si le même nom de politique est inclus au moins deux fois dans ce paramètre, l'exception pour duplicité de nom de politique est propagée.

Le paramètre «desired_props» (propriétés souhaitées) définit l'ensemble des propriétés qui décrivent les offres qui doivent être renvoyées avec la référence d'objet. Trois possibilités se présentent: l'importateur ne souhaite aucune des propriétés; il les souhaite toutes mais sans avoir à les nommer; et, troisièmement, il souhaite certaines propriétés, dont les noms sont fournis. Si l'un des noms du paramètre «desired_props» ne suit pas les règles d'identification, une exception pour nom de propriété incorrect est propagée. Si le même nom de propriété est inclus au moins deux fois dans ce paramètre, l'exception pour nom de propriété dupliqué est propagée. Le paramètre «desired_props» peut désigner des propriétés qui ne sont pas obligatoires pour le type de service demandé. Si la propriété désignée est présente dans l'offre

de service mise en concordance, cette propriété doit être renvoyée. Le paramètre «desired_props» n'a pas d'influence sur la possibilité de retour ou de non-retour d'une offre de service. Pour éviter les propriétés souhaitées «manquantes», l'importateur doit spécifier l'élément «prop_exists» dans la contrainte.

Les offres renvoyées sont réacheminées de deux façons (l'une d'elles ou une combinaison des deux). Le résultat de retour «offers» contient une liste des offres et l'élément «offer_itr» est une référence à une interface auprès de laquelle des offres supplémentaires peuvent être obtenues. Le paramètre «how_many» indique le nombre d'offres qui doivent être renvoyées via le résultat de retour d'offres. Toutes les offres restantes sont disponibles via l'interface d'itérateur. Si le paramètre «how_many» dépasse le nombre d'offres à renvoyer, l'élément «offer_itr» aura la valeur nulle.

Si une règle de cardinalité ou une autre limite est appliquée par un ou par plusieurs courtiers en réponse à une interrogation particulière, le paramètre «limits_applied» contiendra les noms des politiques qui limitent l'interrogation. La séquence des noms retournés dans le paramètre «limits_applied» à partir de toute interrogation (fédérée ou de procuration) doit être concaténée avec les noms des limites appliquées localement et retournées.

#### 8.5.1.1.3 Spécification des politiques d'importateur

```
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 politique «search_card» indique au courtier le nombre maximal d'offres qu'il doit prendre en considération lors de l'évaluation des conformités de type et des concordances d'expressions de contrainte. Le courtier utilise la plus petite valeur fournie ainsi que son propre attribut «max_search_card». Si cette politique n'est pas spécifiée, le courtier utilise la valeur de son attribut «def_search_card».

La politique «match_card» indique au courtier le nombre maximal d'offres compatibles auxquelles il y a lieu d'appliquer les critères de préférence. Le courtier utilise la plus petite valeur fournie ainsi que son propre attribut «max_match_card». Si cette politique n'est pas spécifiée, le courtier utilise la valeur de son attribut «def_match_card».

La politique «return_card» indique au courtier le nombre maximal d'offres compatibles à retourner en réponse à cette interrogation. Le courtier utilise la plus petite valeur fournie ainsi que son propre attribut «max_return_card». Si cette politique n'est pas spécifiée, le courtier utilise la valeur de son attribut «def_return_card».

La politique «use_modifiable_properties» indique si le courtier doit prendre en considération les offres qui ont des propriétés modifiables, lors de la construction de l'ensemble des offres auxquelles il y a lieu d'appliquer les règles de conformité de type et de traitement de contrainte. Si la valeur de cette politique est TRUE, ces offres seront incluses; si la valeur est FALSE, elles ne seront pas incluses. Si cette politique n'est pas spécifiée, ces offres seront incluses.

La politique «use_dynamic_properties» indique si le courtier doit prendre en considération les offres qui ont des propriétés dynamiques, lors de la construction de l'ensemble des offres auxquelles il y a lieu d'appliquer les règles de conformité de type et de traitement de contrainte. Si la valeur de cette politique est TRUE, ces offres seront incluses; si la valeur est FALSE, elles ne seront pas incluses. Si cette politique n'est pas spécifiée, ces offres seront incluses.

La politique «use_proxy_offers» indique si le courtier doit prendre en considération les offres de procuration lors de la construction de l'ensemble des offres auxquelles il y a lieu d'appliquer les règles de conformité de type et de traitement de contrainte. Si la valeur de cette politique est TRUE, ces offres seront incluses; si la valeur est FALSE, elles ne seront pas incluses. Si cette politique n'est pas spécifiée, ces offres seront incluses.

La politique «starting_trader» facilite la distribution du service de courtage proprement dit. Elle permet à un importateur de délimiter le domaine de visibilité d'une recherche en choisissant de naviguer explicitement sur les liaisons du graphe de courtage. Si cette politique est utilisée dans une invocation d'interrogation, il est recommandé qu'elle soit la première paire politique-valeur: cela permet de donner la meilleure suite possible à l'opération d'interrogation. Un paramètre «policies» n'a pas besoin d'inclure une valeur pour la politique «starting_trader». Lorsque cette politique est présente, la première composante nominative est comparée au nom contenu dans chaque liaison. Si aucune concordance n'est trouvée, l'exception de valeur de politique invalide est propagée. Sinon, le courtier invoque l'interrogation en cause à l'interface de consultation détenue par la liaison nommée, mais avec suppression de la préférence *first* lors de la transmission de la politique «starting_trader».

La politique «link_follow_rule» indique comment le client souhaite que les liaisons soient suivies au cours de la résolution de son interrogation. Voir plus de détails à ce sujet au 8.2.7 ci-dessus.

La politique «hop_count» indique au courtier le nombre maximal de sauts entre liaisons fédérées qu'il y a lieu de tolérer lors de la résolution de son interrogation. Le décompte des sauts chez le courtier actuel est déterminé par la valeur minimale de l'attribut «max_hop_count» du courtier et de la politique «hop_count» de l'importateur si ces valeurs sont fournies, ou de l'attribut «def_hop_count» du courtier dans le cas contraire. Si la valeur résultante est zéro, aucune interrogation fédérée n'est permise. Si elle est plus grande que zéro, cette valeur doit être décrémentée avant d'être transmise à un courtier fédéré.

La politique «exact_type_match» indique au courtier si le type de service de l'importateur doit concorder exactement avec un type de service d'une offre; si ce n'est pas le cas (et par défaut), toute offre d'un type conforme au type de service de l'importateur est prise en considération.

La politique «request_id» indique au courtier l'identificateur d'une opération d'interrogation qui a été lancée par un courtier-source agissant comme importateur dans une liaison. Cet identificateur est produit au moyen de l'attribut de courtier request_id_stem. Un courtier n'est pas obligé de produire un tel identificateur pour une opération d'interrogation destinée à un autre courtier mais il est obligé de la faire suivre sur une liaison vers un autre courtier.

### 8.5.2 Itérateur d'offre

#### 8.5.2.1 Signature

```
interface OfferIterator {

    unsigned long max_left (
    ) raises (
        UnknownMaxLeft
    );

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

    void destroy ();
};
```

#### 8.5.2.2 Fonctions des opérations d'itérateur d'offre

L'interface d'itérateur d'offre est utilisée pour renvoyer un ensemble d'offres de service en réponse à l'opération d'interrogation. Cette interface permet d'extraire les offres de service par opérations successives.

L'opération «next_n» renvoie un ensemble d'offres de service dans le paramètre de sortie «offers». Cette opération renvoie *n* offres de service s'il en reste au moins *n* dans l'itérateur. S'il y en a moins, toutes les offres de service restantes sont renvoyées. Le nombre réel d'offres de service renvoyées peut être déterminé à partir de la longueur de la séquence «offers». L'opération «next_n» renvoie la valeur TRUE s'il reste des offres de service à extraire de l'itérateur. Elle renvoie la valeur FALSE s'il n'y en a plus.

L'opération «max_left» renvoie le nombre d'offres de service qui restent dans l'itérateur. L'exception pour nombre maximal restant inconnu («UnknownMaxLeft») est propagée si l'itérateur ne peut pas déterminer le nombre restant d'offres de service (par exemple si l'itérateur détermine son ensemble d'offres de service par évaluation lente).

L'opération «destroy» détruit l'itérateur. Aucune autre opération ne peut être invoquée auprès d'un itérateur une fois que celui-ci a fait l'objet de l'opération «destroy».

### 8.5.3 Enregistrement

```
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 Opération d'exportation

##### 8.5.3.1.1 Signature

```
        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 Fonction

L'opération d'exportation permet de signaler un service, via un courtier, à une communauté d'importateurs éventuels. L'identificateur d'offre renvoyé est le pointeur qui permet à l'exportateur d'identifier l'offre exportée lorsqu'il tente d'y accéder par d'autres opérations. L'identificateur d'offre n'a de sens que dans le contexte du courtier qui l'a produit.

Le paramètre «reference» est l'information qui permet à un client d'interagir avec un serveur distant. Si une réalisation de courtier choisit de considérer certains types de références d'objet (par exemple une référence d'objet nulle) comme étant inexportables, cette mise en œuvre peut alors renvoyer l'exception pour référence d'objet invalide.

Le paramètre «type» identifie le type de service qui contient le type d'interface du paramètre «reference» ainsi qu'un ensemble de types de propriété nommés, qui peuvent être utilisés pour décrire plus complètement cette offre – c'est-à-dire qu'il limite ce qui est acceptable dans le paramètre de propriété. Si la chaîne représentant le paramètre «type» ne suit pas les règles applicables aux identificateurs, une exception pour type de service incorrect est propagée. Si le paramètre «type» est syntaxiquement correct mais qu'un courtier soit en mesure de déterminer sans ambiguïté qu'il ne s'agit pas d'un type de service reconnu, une exception pour type de service inconnu est propagée. Si le courtier peut déterminer que le type d'interface du paramètre «reference» n'est pas un sous-type du type d'interface spécifié dans le paramètre «type», une exception pour discordance de type d'interface est propagée.

Le paramètre «properties» est une liste de valeurs d'éléments nommés qui sont conformes aux types de valeur de propriété définis pour ces noms. Ces éléments décrivent le service qui est offert. Cette description couvre normalement les aspects comportementaux, les aspects non fonctionnels et les aspects autres que de traitement du service. Si l'un quelconque des noms de propriété ne suit pas les règles syntaxiques applicables aux noms de propriété, une exception pour nom de propriété incorrect est propagée. Si le type de l'une quelconque des valeurs de propriété n'est pas le même que le type déclaré (dans le paramètre de type de service), une exception pour discordance de type de propriété est propagée. Si une tentative est faite pour assigner une valeur de propriété dynamique à une propriété en lecture seulement, l'exception pour propriété dynamique en lecture seulement est propagée. Si le paramètre «properties» omet une quelconque propriété déclarée dans le type de service avec un mode de propriété obligatoire, une exception pour propriété obligatoire manquante est propagée. Si deux ou plus de deux propriétés ayant le même nom de propriété sont insérées dans ce paramètre, l'exception pour nom de propriété dupliqué est propagée.

#### 8.5.3.2 Opération de retrait

##### 8.5.3.2.1 Signature

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

##### 8.5.3.2.2 Fonction

L'opération de retrait supprime l'offre de service se trouvant chez le courtier – c'est-à-dire qu'après retrait, l'offre ne peut plus être retournée comme résultat d'une interrogation. L'offre est identifiée par le paramètre «id» qui a été initialement retourné par l'opération d'exportation. Si la chaîne représentant le paramètre «id» ne suit pas les règles applicables aux identificateurs d'offre, une exception pour identificateur d'offre incorrect est propagée. Si le paramètre «id» est correct mais qu'il n'y ait chez le courtier aucune offre ayant cet identificateur, une exception pour identificateur d'offre inconnu est propagée. Si le paramètre «id» désigne une offre de procuration et non pas une offre ordinaire, une exception pour identificateur d'offre de procuration est propagée.

#### 8.5.3.3 Opération de description

##### 8.5.3.3.1 Signature

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

##### 8.5.3.3.2 Fonction

L'opération de description retourne les informations relatives à un service offert qui est détenu par le courtier. Cette opération se compose du paramètre «reference» du service offert, du paramètre «type» de l'offre de service et du paramètre «properties» qui décrit cette offre de service. L'offre est identifiée par le paramètre «id» qui a été initialement retourné par l'exportation. Si la chaîne représentant le paramètre «id» ne suit pas les règles applicables aux identificateurs d'objet, une exception pour identificateur d'offre incorrect est propagée. Si le paramètre «id» est correct mais qu'il n'y ait chez le courtier aucune offre ayant cet identificateur, une exception pour identificateur d'offre inconnu est propagée. Si le paramètre «id» désigne une offre de procuration et non pas une offre ordinaire, une exception pour identificateur d'offre de procuration est propagée.

#### 8.5.3.4 Opération de modification

##### 8.5.3.4.1 Signature

```
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 Fonction

L'opération de modification sert à modifier la description d'un service faisant partie d'une offre de services. Les paramètres de référence d'objet et de type de service associés à l'offre ne peuvent pas être modifiés. Cette opération peut:

a) ajouter de nouvelles propriétés (non obligatoires) pour décrire une offre;

b) modifier les valeurs de certaines propriétés (non en lecture seulement) existantes;

c) supprimer des propriétés (ni obligatoires ni en lecture seulement) existantes.

L'opération de modification réussit complètement ou échoue complètement.

L'offre est identifiée par le paramètre «id» qui a été initialement retourné par l'exportation. Si la chaîne représentant le paramètre «id» ne suit pas les règles applicables aux identificateurs d'offre, une exception pour identificateur d'offre incorrect est propagée. Si le paramètre «id» est correct mais qu'il n'y ait chez le courtier aucune offre ayant cet identificateur, une exception pour identificateur d'offre inconnu est propagée. Si le paramètre «id» désigne une offre de procuration et non pas une offre ordinaire, une exception pour identificateur d'offre de procuration est propagée.

Le paramètre «del_list» indique les noms des propriétés qui ne doivent plus être enregistrées pour l'offre identifiée. Les futures opérations d'interrogation et de description ne verront pas ces propriétés. Si l'un quelconque des noms contenus dans le paramètre «del_list» ne suit pas les règles applicables aux noms de propriété, une exception pour nom de propriété incorrect est propagée. Si un paramètre «name» est correct mais qu'il ne contienne aucune propriété pour l'offre ayant ce «name», une exception pour nom de propriété inconnu est propagée. Si la liste comporte une propriété qui est en mode obligatoire, une exception pour propriété obligatoire est propagée. Si le même nom de propriété est inclus deux ou plus de deux fois dans ce paramètre, l'exception pour nom de propriété dupliqué est propagée.

Le paramètre «modify_list» donne les noms et les valeurs des propriétés à modifier. Si la propriété n'est pas contenue dans l'offre, l'opération de modification l'y ajoute. Les valeurs modifiées (ou ajoutées) de propriété sont renvoyées dans une future interrogation et décrivent les opérations et non les valeurs initiales. Si l'un quelconque des noms contenus dans la liste de modification ne suit pas les règles applicables aux noms de propriété, une exception pour nom de propriété incorrect est propagée. Si la liste comporte une propriété qui est en mode de lecture seulement, une exception pour propriété en lecture seulement est propagée, à moins que cette propriété en lecture seulement ne soit en cours d'enregistrement pour l'offre. L'exception pour propriété dynamique en lecture seulement est propagée si l'on tente d'assigner une valeur de propriété dynamique à une propriété en lecture seulement. Si la valeur d'une quelconque propriété modifiée est d'un type qui n'est pas celui qui est attendu, l'exception pour discordance de types de propriété est propagée. Si au moins deux propriétés ayant le même nom sont incluses dans l'argument, l'exception pour nom de propriété dupliqué est propagée.

L'exception pour opération non mise en œuvre doit être propagée si et seulement si l'attribut «supports_modifiable_properties» donne la valeur FALSE.

Il n'est pas possible de modifier le type de service d'une offre ou la référence d'objet du service. Cette modification doit être réalisée par retrait puis réexportation. L'objet de l'opération de modification est de modifier la description du service offert tout en préservant l'identificateur d'offre. Cela peut être important si l'identificateur d'offre a été propagé dans une communauté d'objets.

#### 8.5.3.5 Opération de retrait au moyen d'une contrainte

##### 8.5.3.5.1 Signature

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

##### 8.5.3.5.2 Fonction

L'opération de retrait au moyen d'une contrainte supprime un ensemble d'offres se trouvant chez un courtier donné. Cet ensemble est identifié de la même façon qu'une opération d'interrogation identifie un ensemble d'offres à renvoyer à un importateur.

Le paramètre «type» achemine le type de service requis. L'expression de contrainte sera appliquée à chaque offre du type spécifié. Si l'offre concorde avec l'expression de contrainte, elle est retirée.

Si le paramètre «type» ne suit pas les règles applicables aux types de service, une exception pour type de service incorrect est propagée. Si le paramètre «type» est syntaxiquement correct mais n'est pas reconnu par le courtier comme étant un type de service, une exception pour type de service inconnu est propagée.

La contrainte «constr» est le moyen qui permet au client de limiter l'ensemble des offres à celles qui sont destinées au retrait. Si la contrainte «constr» ne suit pas les règles syntaxiques applicables à une contrainte, l'exception pour contrainte incorrecte est propagée. Si la contrainte ne trouve aucune concordance avec une offre de service ayant le type spécifié, une exception pour offres non concordantes est propagée.

#### 8.5.3.6 Opération de résolution

##### 8.5.3.6.1 Signature

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

##### 8.5.3.6.2 Fonction

Cette opération sert à résoudre un nom relatif au contexte pour un autre courtier. Il sert en particulier à exporter un service vers un courtier qui est désigné par un nom plutôt que par une référence d'interface. Le client fournit le nom, qui sera une séquence de composantes nominatives. Si le contenu du paramètre ne peut pas produire une syntaxe correcte pour la première composante, l'exception pour nom de courtier incorrect est propagée. Sinon, la première composante nominative est comparée au nom associé à chaque liaison. Si aucune concordance n'est trouvée, ou si le courtier ne prend pas en charge les liaisons, l'exception pour nom de courtier inconnu est propagée. Sinon, le courtier obtient la fonction d'enregistrement conditionnel qui est associée à la liaison mise en concordance. Si la référence de l'interface d'enregistrement n'est pas nulle, le courtier se met en liaison avec cette interface et invoque l'opération de résolution tout en transmettant le nom de courtier avec la première composante supprimée; si l'interface d'enregistrement a la valeur nulle, l'exception pour enregistrement non pris en charge est propagée. Lorsqu'un courtier est en mesure de trouver une concordance avec la première composante nominative sans laisser de nom résiduel, ce courtier renvoie la référence de l'interface d'enregistrement pour ce courtier lié. Lorsqu'ils remontent la chaîne récursive, les courtiers intermédiaires retournent à leurs clients (d'autres courtiers) la référence de l'interface d'enregistrement.

### 8.5.4 Itérateur d'identificateur d'offre

#### 8.5.4.1 Signature

```
interface OfferIdIterator {

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

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

        void destroy ();
    };
```

#### 8.5.4.2 Fonctions des opérations d'itérateur d'identificateur d'offre

L'interface d'itérateur d'identificateur d'offre sert à renvoyer un ensemble d'identificateurs d'offre à partir de l'opération de listage d'offres et de l'opération de listage de procurations dans l'interface d'administration. L'interface d'itérateur d'identificateur d'offre permet d'extraire les identificateurs d'offre par opérations successives.

L'opération «next_n» renvoie un ensemble d'identificateurs d'offre, dans le paramètre de sortie «ids». L'opération renvoie *n* identificateurs d'offre si au moins *n* identificateurs d'offre restent dans l'itérateur. S'il y en a moins, tous les identificateurs d'offre restants sont renvoyés. Le nombre réel d'identificateurs d'offre renvoyés peut être déterminé d'après la longueur de la séquence «ids». L'opération «next_n» renvoie la valeur TRUE s'il reste d'autres identificateurs d'offre à extraire de l'itérateur. Elle renvoie la valeur FALSE dans le cas contraire.

L'opération «max_left» renvoie le nombre d'identificateurs d'offre qui restent dans l'itérateur. L'exception pour nombre maximal d'identificateurs restant inconnu est propagée si l'itérateur ne peut pas déterminer le nombre d'identificateurs d'offre restant (par exemple si l'itérateur détermine son ensemble d'identificateurs d'offre par évaluation lente).

L'opération «destroy» détruit l'itérateur. Aucune autre opération ne peut être invoquée auprès d'un itérateur une fois que celui-ci a fait l'objet de l'opération «destroy».

### 8.5.5 Administration

```
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 Opérations sur les attributs et sur les ensembles

L'interface d'administration permet de lire et d'écrire les valeurs des attributs de courtier. Tous les attributs sont définis comme étant en lecture seulement, qu'il s'agisse d'attributs de support, d'attributs d'importation, d'attributs de liaison ou d'attributs d'administration. Dans l'interface d'administration, on définit des opérations de dénomination d'attribut («set_<attribute_name>») afin de donner une nouvelle valeur au paramètre «attribute» des courtiers. Chacune de ces opérations de dénomination renvoie la valeur précédente de l'attribut, qui était sa valeur fonctionnelle.

Si l'on invoque à l'interface d'administration l'opération de fixation de la valeur de support des offres de procuration («set_support_proxy_offers») avec une valeur mise à FALSE, pour un courtier qui prend en charge l'interface de procuration, cette valeur de support des offres de procuration n'a pas d'incidence sur le fonctionnement des opérations effectuées dans l'interface de procuration. Mais, dans ce cas, cette valeur FALSE a pour effet qu'aucune offre de procuration exportée via l'interface de procuration pour ce courtier n'est plus disponible pour répondre à des interrogations relatives à l'interface de consultation de ce courtier.

#### 8.5.5.2 Opération de listage des offres

##### 8.5.5.2.1 Signature

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

##### 8.5.5.2.2 Fonction

L'opération de listage des offres permet à l'administrateur d'un courtier d'effectuer des opérations d'intendance en obtenant un pointeur sur chacune des offres se trouvant chez un courtier, par exemple pour le ramassage des déchets, etc. Seuls les identificateurs des offres ordinaires sont retournés. Les identificateurs des offres de procuration ne sont pas retournés par cette opération. Si le courtier ne prend pas en charge l'interface d'enregistrement, l'exception pour non-mise en œuvre est propagée.

Les identificateurs renvoyés sont réacheminés de deux façons (l'une d'elles ou une combinaison des deux). Le résultat de retour «ids» contient une liste des identificateurs d'offre et l'élément «id_itr» est une référence à une interface auprès de laquelle des identificateurs d'offre additionnels peuvent être obtenus. Le paramètre «how_many» indique le nombre d'offres qui doivent être renvoyées via le résultat de retour d'identificateurs («ids»). Tous les identificateurs restants sont disponibles via l'interface d'itérateur. Si le paramètre «how_many» dépasse le nombre d'identificateurs à renvoyer, l'élément «id_itr» a la valeur nulle.

#### 8.5.5.3 Opération de listage des offres d'application

##### 8.5.5.3.1 Signature

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

##### 8.5.5.3.2 Fonction

L'opération de listage des offres de procuration renvoie l'ensemble des identificateurs d'offres de procuration se trouvant chez un courtier. Le nombre maximal d'identificateurs d'offre indiqué par le paramètre «how_many» est renvoyé via le paramètre «ids». S'il y a plus d'identificateurs d'offre qu'indiqué par le paramètre «how_many», les identificateurs restants sont renvoyés via l'itérateur «id_itr». S'il n'y a que le nombre d'identificateurs d'offre indiqué par le paramètre «how_many» ou moins, l'itérateur «id_itr» a la valeur nulle. Si le courtier ne supporte pas l'interface de procuration, l'exception pour non-mise en œuvre est propagée.

### 8.5.6 Liaison

```
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 Opération d'adjonction de liaison

##### 8.5.6.1.1 Signature

```
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 Fonction

L'opération d'adjonction de liaison permet à un courtier de faire successivement appel aux services d'un autre courtier lors de l'exécution de ses propres opérations de service de courtage.

Le paramètre «name» est utilisé lors des opérations successives de gestion de liaison pour identifier la liaison visée. Si ce paramètre n'est pas formé correctement, l'exception pour nom de liaison incorrect est propagée. Une exception pour nom de liaison dupliqué est propagée si le nom de la liaison existe déjà. Le nom de liaison est également utilisé comme composante dans une séquence de composantes nominatives désignant un courtier afin d'effectuer des opérations de résolution ou de renvoi. La séquence des noms de liaison relatifs au contexte fournit un chemin au courtier.

Le paramètre «target» désigne l'interface de consultation qui donne accès au service de courtage fourni par le courtier-cible. Si le paramètre d'interface de consultation a la valeur nulle, une exception pour référence d'interface de consultation invalide est propagée. L'interface cible sert à obtenir l'interface d'enregistrement associée, qui sera par la suite renvoyée dans le cadre de l'opération de description de liaison et qui sera invoquée dans le cadre d'une opération de résolution.

Le paramètre de transmission de règle de suivi par défaut («def_pass_on_follow_rule») spécifie le comportement de liaison par défaut pour la liaison si aucun autre comportement de liaison n'est spécifié dans une demande d'interrogation d'importateur. Si ce paramètre dépasse la valeur spécifiée dans le paramètre suivant pour la règle de limitation de suivi de liaison («limiting_follow_rule»), une exception pour règle par défaut de suivi de liaison trop peu contraignante est propagée.

Le paramètre de règle de limitation de suivi de liaison («limiting_follow_rule») spécifie le comportement de suivi de liaison le moins contraignant que la liaison est prête à tolérer. L'exception pour règle de limitation de suivi trop peu contraignante est propagée si la valeur de ce paramètre dépasse celle de l'attribut de politique maximale de suivi de liaison («max_link_follow_policy») du courtier au moment de la création de la liaison.

NOTE – Il est possible qu'une règle de limitation de suivi de liaison dépasse la valeur de l'attribut de politique maximale de suivi de liaison à un moment ultérieur de la durée de vie de cette liaison, car il est possible que le courtier assigne une valeur plus restrictive à cet attribut après la création de la liaison.

#### 8.5.6.2 Opération de suppression de liaison

##### 8.5.6.2.1 Signature

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

##### 8.5.6.2.2 Fonction

L'opération de suppression de liaison élimine toute connaissance du courtier-cible. Celui-ci ne peut donc plus être utilisé par la suite pour résoudre, faire suivre ou propager des opérations de courtage, à partir de ce courtier.

Le paramètre «name» désigne la liaison à supprimer. L'exception pour nom de liaison incorrect est propagée si la liaison est mal formée et l'exception pour nom de liaison inconnu est propagée si la liaison désignée ne se trouve pas chez le courtier.

#### 8.5.6.3 Opération de description de liaison

##### 8.5.6.3.1 Signature

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

##### 8.5.6.3.2 Fonction

L'opération de description de liaison renvoie des informations sur une liaison se trouvant chez le courtier.

Le paramètre «name» désigne la liaison dont la description est requise. En cas de nom de liaison mal formé, l'exception pour nom de liaison incorrect est propagée. Une exception pour nom de liaison inconnu est propagée si la liaison nommée n'est pas trouvée chez le courtier.

L'opération retourne une structure d'informations sur la liaison, comprenant ce qui suit: l'interface de consultation du service de courtage, l'interface d'enregistrement du service de courtage cible, et le comportement de suivi de liaison par défaut ainsi que le comportement de limitation de suivi de la liaison nommée. Si le service cible ne supporte pas l'interface d'enregistrement, ce champ a la valeur nulle dans la structure d'informations de liaison.

NOTE – Etant donné la description de l'opération de résolution-enregistrement («Register::resolve()») figurant au 8.5.3.6 ci-dessus, la plupart des mises en œuvre pourront opter pour la détermination de l'interface d'enregistrement au moment de l'appel d'une liaison additionnelle et de la mise en mémoire statique de cette information avec le reste des états de la liaison.

#### 8.5.6.4 Opération de listage de liaisons

##### 8.5.6.4.1 Signature

```
LinkNameSeq list_links ();
```

##### 8.5.6.4.2 Fonction

L'opération de listage des liaisons renvoie une liste des noms de toutes les liaisons de courtage se trouvant chez le courtier. Ces noms pourront être utilisés par la suite pour d'autres opérations de gestion, comme la description ou la suppression de liaison.

#### 8.5.6.5 Opération de modification de liaison

##### 8.5.6.5.1 Signature

```
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 Fonction

L'opération de modification de liaison sert à modifier les comportements de suivi de liaison existant pour une liaison désignée. La référence d'interface de consultation chez le courtier-cible et le nom de liaison ne peuvent pas être modifiés.

Le paramètre «name» désigne la liaison dont les comportements de suivi doivent être changés. Un paramètre «name» mal formé propage l'exception pour nom de liaison incorrect. Une exception pour nom de liaison inconnu est propagée si le courtier n'a pas connaissance du nom de la liaison.

Le paramètre «def_pass_on_follow_rule» spécifie le nouveau comportement par défaut de suivi pour cette liaison. Si ce paramètre dépasse la valeur de la règle «limiting_follow_rule» spécifiée dans le paramètre suivant, une exception pour comportement par défaut de suivi trop peu contraignant est propagée.

Le paramètre «limiting_follow_rule» spécifie la nouvelle limite du comportement par défaut de suivi de cette liaison. L'exception pour comportement de limitation de suivi de liaison trop peu contraignant est propagée si la valeur de ce paramètre dépasse celle de la politique maximale de suivi de liaison actuelle du courtier.

### 8.5.7 Procuration

```
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 Opération d'exportation de procuration

##### 8.5.7.1.1 Signature

```
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 Fonction

L'interface de procuration permet l'exportation puis la manipulation d'offres de procuration. Celles-ci permettent de déterminer en cours d'exécution du programme l'interface qui fournit un service. L'opération d'exportation de procuration ajoute une offre de procuration à l'ensemble des offres de service d'un courtier.

Comme les offres de service normales, les offres de procuration possèdent un paramètre «type» contenant le type de service et un paramètre «properties» contenant les valeurs nommées de la propriété. Une offre de procuration ne contient cependant pas de référence à l'objet qui fournit le service offert. Cette référence d'objet est au contraire obtenue lorsqu'elle est nécessaire à une opération d'interrogation. On l'obtient en invoquant une autre opération d'interrogation à l'interface de consultation «cible» se trouvant dans l'offre de procuration.

Le paramètre «if_match_all» (si concordance avec toutes les offres) a la valeur TRUE, cela indique que le courtier doit considérer cette offre de procuration comme étant en concordance avec une interrogation d'importateur fondée sur la conformité du type seulement – c'est-à-dire que le courtier ne cherche pas à faire concorder l'expression de contrainte de l'importateur avec les propriétés associées à l'offre de procuration. C'est le plus souvent utile lorsque l'expression de contrainte fournie par l'importateur est simplement retransmise dans l'opération d'interrogation secondaire.

Le paramètre «recipe» indique au courtier comment construire l'expression de contrainte pour l'opération d'interrogation secondaire adressée à la cible «target». Le langage de recette est décrit dans l'Annexe C; il permet de construire l'expression de contrainte secondaire avec des littéraux, avec des valeurs de propriétés de l'offre de procuration et avec l'expression de contrainte primaire.

Le paramètre «policies_to_pass_on» fournit un ensemble statique de paires <nom, valeur> pour retransmission au courtier «cible». Le tableau ci-dessous décrit comment le paramètre de politique secondaire est construit à partir du paramètre de politique primaire et du paramètre «policies_to_pass_on».

Si une opération d'interrogation concorde avec l'offre de procuration (au moyen des algorithmes normaux de mise en concordance du type de service, des algorithmes de mise en concordance des propriétés et des algorithmes de préférence), l'opération d'interrogation primaire invoque une opération d'interrogation secondaire à l'interface de consultation désignée par l'offre de procuration. Bien que l'offre de procuration désigne une interface de consultation, cette interface n'est appelée qu'à être conforme syntaxiquement à l'interface de consultation. Il n'est pas nécessaire qu'elle soit conforme au comportement d'interface de consultation spécifié ci-dessus.

L'opération d'interrogation secondaire est invoquée comme suit:

| | |
|---|---|
| – dans le nom de type de service (paramètre «type»): | le type est copié sur la base de l'interrogation primaire; |
| – dans la contrainte «constr»: | la recette contenue dans l'offre de procuration est «évaluée» pour fournir le paramètre «constr»; |
| – dans les préférences (paramètre «pref»): | la préférence est copiée à partir de l'interrogation primaire; |
| – dans les séquences de politiques (paramètre «policies»): | les paramètres «policies» (noms et valeurs) contenus dans le champ «policies_to_pass_on» de l'offre de procuration sont ajoutés aux politiques de l'interrogation primaire; |
| – dans les propriétés spécifiées (paramètre «desired_props»): | les paramètres «desired_props» sont copiés à partir de l'interrogation primaire; |
| – dans les nombres non signés longs (paramètre «how_many»): | le paramètre «how_many» est fixé par le courtier de façon à représenter sa préférence de mise en œuvre pour la réception de l'offre résultante, sous forme de liste ou par l'intermédiaire d'un itérateur; |

| | |
|---|---|
| – à partir des séquences d'offres (paramètre «offers»): | un nombre d'offres au plus égal à la valeur du paramètre «how_many» est retourné à partir de l'opération d'interrogation secondaire, au moyen du paramètre «offers»; |
| – à partir de l'itérateur d'offres (paramètre «offer_itr»): | si l'interrogation secondaire a besoin de retourner plus d'offres qu'indiqué par le paramètre «how_many», les offres restantes peuvent être récupérées par l'intermédiaire du paramètre «offer_itr» de l'itérateur. S'il n'y a que «how_many» offres ou moins, le paramètre «offer_itr» a la valeur nulle; |
| – à partir d'une séquence de noms de limites politiques (paramètre «limits_applied»): | les noms de toutes limites de politique qui ont été appliquées par le courtier d'offres de procuration. |

L'exception pour type de service incorrect est propagée si le nom du type de service (paramètre «type») n'est pas correctement formé. L'exception pour type de service inconnu est propagée si le nom du type de service (paramètre «type») n'est pas connu du courtier. L'exception pour référence de consultation invalide est propagée si la cible n'est pas une référence valide d'interface de consultation (par exemple si la cible est une référence d'objet nulle). L'exception pour nom de propriété incorrect est propagée si un nom de propriété contenu dans le paramètre «properties» n'est pas correctement formé. L'exception pour discordance de types de propriété est propagée si une valeur de propriété n'est pas d'un type approprié, tel que déterminé par le type du service. L'exception pour propriété dynamique en lecture seulement est propagée si une valeur de propriété dynamique a été fournie pour une propriété qui a été étiquetée comme étant en lecture seulement. L'exception pour propriété obligatoire manquante est propagée si le paramètre «properties» ne contient pas une des propriétés obligatoires définies par le type du service. L'exception pour recette incorrecte est propagée si la recette n'est pas correctement formée. L'exception pour nom de propriété dupliqué est propagée si deux ou plus de deux propriétés portant le même nom de propriété sont incluses dans le paramètre «properties». L'exception pour nom de politique dupliqué est propagée si deux ou plus de deux politiques portant le même nom de politique sont incluses dans le paramètre «policies_to_pass_on».

Les offres de procuration ne peuvent pas être modifiées; elles doivent être supprimées puis réexportées.

#### 8.5.7.2 Opération de retrait de procuration

##### 8.5.7.2.1 Signature

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

##### 8.5.7.2.2 Fonction

L'opération de retrait de procuration supprime l'offre de procuration identifiée par le paramètre «id» issu du courtier.

L'exception pour identificateur d'offre incorrect est propagée si le paramètre «id» n'est pas correctement formé. L'exception pour identificateur d'offre inconnu est propagée si le paramètre «id» ne désigne pas une offre contenue chez le courtier. L'exception pour identificateur d'offre autre que de procuration est propagée si le paramètre «id» désigne une offre de service normale et non une offre de procuration.

#### 8.5.7.3 Opération de description d'offre de procuration

##### 8.5.7.3.1 Signature

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

#### 8.5.7.3.2 Fonction

L'opération de description d'offre de procuration renvoie les informations contenues dans l'offre de procuration désignée par le paramètre «id» chez le courtier.

L'exception pour identificateur d'offre incorrect est propagée si le paramètre «id» n'est pas correctement formé. L'exception pour identificateur d'offre inconnu est propagée si le paramètre «id» ne désigne pas une offre contenue chez le courtier. L'exception pour identificateur d'offre autre que de procuration est propagée si le paramètre «id» désigne une offre de service normale et non une offre de procuration.

## 8.6 Interface d'évaluation de propriété dynamique

### 8.6.1 Signature

```
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 Fonctions des opérations de l'interface d'évaluation de propriété dynamique

L'interface d'évaluation de propriété dynamique est fournie par un exportateur qui souhaite fournir une valeur de propriété dynamique dans une offre de service détenue par le courtier.

Lors de l'exportation d'une offre de service (ou de procuration), la propriété à valeur dynamique possède une valeur de type «any» qui contient une structure de propriété dynamique au lieu de la valeur de propriété normale. Un courtier qui supporte les propriétés dynamiques accepte cette valeur de propriété dynamique comme contenant les informations qui permettent d'obtenir une valeur de propriété de type correct au cours de l'évaluation d'une interrogation. L'opération d'exportation (ou l'exportation de procuration) propage l'exception pour discordance de types de propriété si le type retourné n'est pas approprié au nom de propriété défini par le type de service.

Les propriétés en lecture seulement ne peuvent pas avoir de valeurs dynamiques. Les opérations d'exportation et de modification à l'interface d'enregistrement, ainsi que l'opération d'exportation de procuration à l'interface de procuration, propagent l'exception pour propriété dynamique en lecture seulement si des valeurs dynamiques sont assignées aux propriétés en lecture seulement.

Lorsqu'une interrogation nécessite une valeur de propriété dynamique, l'opération d'évaluation de propriété dynamique est invoquée à l'interface d'évaluation conditionnelle («eval_if») dans la structure de propriété dynamique. Le paramètre de nom de propriété contient le nom de la propriété dont on cherche à obtenir la valeur. Les paramètres «returned_type» et «extra_info» sont copiés à partir de la structure de propriété dynamique. L'opération d'évaluation de propriété dynamique renvoie toute valeur qui doit convenir à cette propriété. La valeur doit être du type indiqué par le paramètre «returned_type».

L'exception pour échec d'évaluation de propriété dynamique est propagée si la valeur de la propriété ne peut pas être déterminée. Si la valeur est requise pour l'évaluation d'une contrainte ou d'une préférence, cette évaluation est considérée comme ayant échoué pour cette offre de service (ou de procuration).

En dehors des règles qui précèdent, la présente Spécification ne précise pas d'autre comportement de l'opération d'évaluation de propriété dynamique. En particulier, l'objet des données autres que d'information («extra_info data») est, lors de la détermination de la valeur de propriété dynamique, fonction de la réalisation.

Si le courtier ne supporte pas les propriétés dynamiques (indiquées par l'attribut de courtier «supports_dynamic_properties»), les opérations d'exportation et d'exportation de procuration ne doivent pas être paramétrées par des propriétés dynamiques. Le comportement de ces courtiers dans ces circonstances n'est pas spécifié dans la présente Spécification.

Si le courtier ne supporte pas les propriétés dynamiques ou que l'importateur ait demandé que celles-ci ne soient pas utilisées (au moyen du paramètre «policies» de l'opération d'interrogation), l'évaluation de propriété dynamique n'est pas effectuée. Si la valeur d'une propriété dynamique est requise pour l'évaluation d'une contrainte ou d'une préférence, cette évaluation est considérée comme ayant échoué pour cette offre de service (ou de procuration).

L'opération de description de l'interface d'enregistrement et l'opération de description de procuration de l'interface de procuration n'effectuent pas d'évaluation de propriété dynamique mais renvoient la structure de propriété dynamique sous la forme d'une valeur de propriété. Etant donné que ces interfaces sont utilisées pour créer des propriétés dynamiques au moyen des opérations d'exportation et d'exportation de procuration, les autres opérations sur ces interfaces doivent s'assurer que la nature dynamique des propriétés reste visible pour les exportateurs.

L'opération de modification à l'interface d'enregistrement d'un courtier qui supporte les propriétés dynamiques doit accepter l'établissement et la modification de propriétés dynamiques, de manière cohérente avec l'opération d'exportation. Il n'est pas interdit qu'une valeur de propriété passe du statut statique, enregistré par le courtier, au statut de valeur dynamique et inversement; mais les propriétés statiques en lecture seulement ne peuvent pas être transformées en propriétés dynamiques.

## 8.7 Gabarit d'objet-courtier

Un gabarit d'objet de traitement de base pour le courtier contient:

- des gabarits d'interface de traitement pour les interfaces qu'il peut instancier;
- une spécification de comportement;
- une spécification contractuelle d'environnement.

### 8.7.1 Gabarits d'interface

La présente Spécification fournit les gabarits d'interface suivants:

- gabarit d'interface de composantes de courtier;
- gabarit d'interface d'attributs de support;
- gabarit d'interface d'attributs d'importation;
- gabarit d'interface d'attributs de liaison;
- gabarit d'interface de consultation;
- gabarit d'interface d'enregistrement;
- gabarit d'interface d'administration;
- gabarit d'interface de liaison;
- gabarit d'interface de procuration;
- gabarit d'interface d'itérateur d'offre;
- gabarit d'interface d'itérateur d'identificateur d'offre;
- gabarit d'interface d'évaluateur de propriété dynamique.

### 8.7.2 Spécification de comportement

L'instanciation d'un gabarit d'interface jouant le rôle de serveur nécessite qu'un objet réponde aux prescriptions de comportement associées aux opérations contenues dans l'interface.

L'instanciation d'un gabarit d'interface jouant le rôle de client nécessite qu'un objet soit capable de recevoir toutes les terminaisons possibles pour les opérations qu'il invoque à cette interface.

Lorsqu'un objet-courtier instancie une interface de consultation jouant le rôle de client afin de prendre en charge des services invoqués par une interface de consultation jouant son rôle de serveur, cet objet-courtier doit transmettre à cette interface-serveur les réponses reçues de cette interface-client, conformément à la spécification de comportement du gabarit d'interface de consultation.

Lorsqu'un objet-courtier instancie une interface d'enregistrement jouant le rôle de client afin de prendre en charge des services invoqués par une interface de consultation jouant son rôle de serveur, cet objet-courtier doit transmettre à cette interface-serveur les réponses reçues de cette interface-client, conformément à la spécification de comportement du gabarit d'interface de consultation.

Lorsqu'un objet-courtier instancie une interface d'enregistrement jouant le rôle de client afin de prendre en charge des services invoqués par une interface d'enregistrement jouant son rôle de serveur, cet objet-courtier doit transmettre à cette interface-serveur les réponses reçues de cette interface-client, conformément à la spécification de comportement du gabarit d'interface d'enregistrement.

Il existe encore d'autres actions que le courtier peut exécuter.

#### 8.7.2.1 Instanciation de gabarits d'interface

Les gabarits suivants peuvent être instanciés:

- gabarit d'interface de consultation (rôle de serveur);
- gabarit d'interface de consultation (rôle de client);
- gabarit d'interface d'enregistrement (rôle de serveur);
- gabarit d'interface d'administration (rôle de serveur);
- gabarit d'interface de liaison (rôle de serveur);
- gabarit d'interface de procuration (rôle de serveur);
- gabarit d'interface d'itérateur d'offre (rôle de serveur);
- gabarit d'interface d'itérateur d'identificateur d'offre (rôle de serveur).

NOTE – Il peut y avoir de nombreuses instances de chaque interface de consultation (rôle client), d'itérateur d'offre, d'itérateur d'identificateur d'offre.

Les mises en œuvre peuvent instancier d'autres interfaces à rôle client, par exemple une interface de conteneur de types ou une interface de stockage. Il s'agit cependant de fonctions de traitement ODP distinctes, dont la normalisation est hors du domaine d'application de la présente Spécification.

#### 8.7.2.2 Actions concernant les états

Le courtier peut accéder à son état et le modifier. Les composantes d'état sont les suivantes:

- l'ensemble des offres de service;
- l'ensemble des offres de procuration;
- l'ensemble des liaisons;
- l'ensemble des attributs de courtier.

L'interface d'enregistrement met à jour les offres de service.

L'interface de procuration met à jour les offres de procuration.

L'interface de liaison met à jour les liaisons.

L'interface d'administration met à jour les attributs de courtier.

#### 8.7.2.3 Comportement d'objet combiné consultation-enregistrement

Toutes les offres de service se trouvant à un moment donné chez le courtier en raison de l'exécution d'opérations à l'interface d'enregistrement sont disponibles pour retour au moyen de l'opération d'interrogation de l'interface de consultation.

#### 8.7.2.4 Comportement d'objet combiné consultation-procuration

Toutes les offres de service se trouvant à un moment donné chez le courtier en raison de l'exécution d'opérations à l'interface de procuration sont disponibles pour retour au moyen de l'opération d'interrogation de l'interface de consultation.

#### 8.7.2.5 Comportement d'objet combiné administration-consultation

L'ensemble des offres de service qui peuvent être retournées est affecté par la valeur de l'attribut «support_dynamic_properties» fixée au moyen des opérations d'interface d'administration. Si la valeur de cet attribut est mise à FALSE, les opérations d'interrogation suivantes ne doivent pas retourner d'offres comportant des propriétés dynamiques.

L'ensemble des offres de procuration qui peuvent être retournées est affecté par la valeur de l'attribut «support_proxy_offer» fixée au moyen des opérations d'interface d'administration. Si la valeur de cet attribut est mise à FALSE, les opérations d'interrogation suivantes ne doivent pas retourner d'offres issues d'une offre de procuration, même si la valeur «use_proxy_offer» du paramètre de politique d'importation indiquée par l'opération d'interrogation a été mise à la valeur TRUE.

L'ensemble des offres de service qui peuvent être retournées est affecté par la valeur de l'attribut «support_modifiable_properties» fixée au moyen des opérations d'interface d'administration. Si la valeur de cet attribut est mise à FALSE, les opérations d'interrogation suivantes ne doivent pas retourner d'offres comportant des propriétés modifiables.

#### 8.7.2.6 Comportement d'objet combiné administration-enregistrement

La disponibilité de l'opération de modification à l'interface d'enregistrement est affectée par la valeur de l'attribut «supports_modifiable_properties» du courtier.

La capacité d'exporter des offres de service ayant des propriétés dynamiques est affectée par la valeur de l'attribut «supports_dynamic_properties» du courtier. Si cette valeur est mise à FALSE, on peut rejeter les demandes d'exportation d'offre comportant des propriétés dynamiques.

#### 8.7.2.7 Comportement d'objet combiné administration-procuration

Les fonctions associées à l'interface de procuration ne sont pas affectées par la valeur de l'attribut «supports_proxy_offer». Si cette valeur est mise à FALSE, aucune offre de procuration se trouvant chez le courtier ne doit être mise à la disposition d'opérations d'interrogation à l'interface de consultation.

La capacité d'exporter des offres de procuration ayant des propriétés dynamiques est affectée par la valeur de l'attribut «supports_dynamic_properties» du courtier. Si cette valeur est mise à FALSE, on peut rejeter les demandes d'exportation de procuration comportant des propriétés dynamiques.

#### 8.7.2.8 Comportement d'objet combiné liaison-consultation

Toutes les offres de service accessibles par les interfaces de consultation de courtiers liés doivent être accessibles par récurrence lors d'une opération d'interrogation visant une interface de consultation initiale.

#### 8.7.2.9 Comportement d'objet combiné liaison-enregistrement

L'opération de résolution à l'interface d'enregistrement n'est disponible que si le courtier est lié à d'autres courtiers.

#### 8.7.2.10 Action d’arbitrage

Un gabarit d'action d'arbitrage est destiné aux actions qui combinent un argument-critère (fourni à une interface) avec des critères de courtier et des valeurs de propriété (que l'on peut déduire de l'état particulier du courtier). L'action d'arbitrage produit un critère combiné qui correspond à la politique (en vue d'entreprise) d'exécution d'une opération donnée.

L'action d'arbitrage représente un certain algorithme de traitement situé à l'intérieur de l'objet-courtier. Elle correspond à la politique d'arbitrage de la spécification d'entreprise.

#### 8.7.2.11 Contraintes sur l'exécution d'actions

Le comportement de l'interface d'exportation modifie l'espace des offres de service et les identités de ces offres. Le comportement de l'interface de liaison modifie l'espace des liaisons. Le comportement de l'interface d'administration modifie les attributs du courtier. Afin d'assurer la cohérence des données, aucune contrainte n'est donc nécessaire quant à la combinaison d'actions issues d'opérations sur une interface donnée avec des actions issues d'opérations sur l'autre interface. Le comportement de l'interface de procuration modifie l'espace des offres de procuration ainsi que les identificateurs d'offre.

### 8.7.3 Contrat d’environnement

Il peut y avoir une contrainte d'environnement telle que la première interface de service de courtage qui est instanciée se trouve à un emplacement donné, ce qui permet d'instancier d'autres objets ayant connaissance d'un courtier.

Aucune autre contrainte d'environnement n'est identifiée.

# 9 Déclarations de conformité et points de référence

Les interfaces opérationnelles suivantes sont des points de référence de programmation destinés aux tests de conformité:

- interface de consultation (en tant que serveur) fournie par la réalisation de courtage vérifiée;
- interface d'enregistrement (en tant que serveur) fournie par la réalisation de courtage vérifiée;
- interface de liaison (en tant que serveur) fournie par la réalisation de courtage vérifiée;
- interface d'administration (en tant que serveur) fournie par la réalisation de courtage vérifiée;
- interface de procuration (en tant que serveur) fournie par la réalisation de courtage vérifiée;
- interface de consultation (en tant que client) d'un courtier lié, utilisée par la réalisation de courtage vérifiée;
- interface d'enregistrement (en tant que client) d'un courtier lié, utilisée par la réalisation de courtage vérifiée;
- interface d'évaluation de propriété dynamique (en tant que client) d'un objet, utilisée par la réalisation de courtage vérifiée au cours de l'évaluation d'une propriété dynamique.

Le comportement défini pour chacune des opérations normatives contenues dans la spécification de traitement doit être manifesté aux points de référence de conformité associés à ce comportement.

Pour revendiquer la conformité, les réalisateurs doivent indiquer quelle est l'interface d'ingénierie qui correspond à chacune des interfaces de traitement assimilées à des points de conformité.

Par ailleurs, le réalisateur doit indiquer le mécanisme de communication utilisé et les transparences assurées aux interfaces.

La classification suivante est définie pour les classes spécifiques de conformité des mises en œuvre d'objets fonctionnels de courtage:

- courtier d'interrogation: peut servir une interface de consultation;
- courtier simple: peut servir une interface de consultation et une interface d'enregistrement;
- courtier autonome: peut servir une interface de consultation, une interface d'enregistrement et une interface d'administration;
- courtier lié: peut servir une interface de consultation, une interface d'enregistrement, une interface d'administration et une interface de liaison; peut également être client d'une interface de consultation et d'une interface d'enregistrement;
- courtier de procuration: peut servir une interface de consultation, une interface d'enregistrement, une interface d'administration et une interface de procuration; peut également être client d'une interface de consultation;
- courtier tous services: peut servir une interface de consultation, une interface d'enregistrement, une interface d'administration, une interface de liaison et une interface de procuration; peut également être client d'une interface de consultation et d'une interface d'enregistrement.

L'une quelconque de ces classes spécifiques de conformité d'objet de courtage peut aussi être cliente de l'interface d'évaluation de propriété dynamique si cette classe est compatible avec les propriétés dynamiques.

La classification ci-dessus se traduit par cinq types spécifiques d'objets fonctionnels de courtage, répondant aux conditions de conformité.

## 9.1 Prescription de conformité des interfaces ayant la fonctio n de courtage en tant que serveurs

Etant donné que les interfaces avec un objet de courtage sont séparables, et que la prise en charge de ces interfaces peut être sélectionnée (sous réserve des classes de conformité définies ci-dessus), le présent paragraphe spécifie les prescriptions de conformité interface par interface.

### 9.1.1 Interface de consultation

Une réalisation revendiquant la conformité à l'interface de consultation en tant que serveur doit mettre en œuvre le comportement complet qui est associé à toutes les opérations et à tous les attributs en lecture seulement qui sont définis dans le domaine du type «Interface de consultation».

Une réalisation revendiquant la conformité à l'interface de consultation en tant que serveur doit également prendre en charge le type «Interface d'itérateur d'offres» en tant que serveur.

### 9.1.2 Interface d'enregistrement

Une réalisation revendiquant la conformité à l'interface d'enregistrement en tant que serveur doit mettre en œuvre le comportement complet qui est associé à toutes les opérations et à tous les attributs en lecture seulement qui sont définis dans le domaine du type «Interface d'enregistrement», avec les exceptions autorisées suivantes:

- une réalisation qui n'autorise que la valeur FALSE pour l'attribut «supports_modifiable_properties» est conforme, auquel cas elle peut rejeter une offre de service qui comporte des propriétés modifiables transmises dans le cadre d'une opération d'exportation; une telle réalisation peut toujours répondre par une exception à des demandes d'opération;
- une réalisation qui n'autorise que la valeur FALSE pour l'attribut «supports_dynamic_properties» est conforme, auquel cas elle peut rejeter une offre de service qui comporte des propriétés dynamiques transmises dans le cadre d'une opération d'exportation;
- une réalisation qui revendique la conformité à l'interface d'enregistrement en tant que serveur, avec l'attribut «supports_dynamic_properties» mis à la valeur TRUE, doit également être en mesure de jouer le rôle de client pour l'interface de type «Evaluateur de propriété dynamique»;
- une réalisation qui revendique la conformité à l'interface d'enregistrement en tant que serveur, avec l'attribut en lecture seulement «supports_proxy_offers» mis à la valeur TRUE, doit également prendre en charge l'interface de procuration.

### 9.1.3 Interface de liaison

Une réalisation qui revendique la conformité à l'interface de liaison en tant que serveur doit manifester l'ensemble du comportement associé à toutes les opérations et à tous les attributs en lecture seulement qui sont définis dans le domaine du type «Interface de liaison».

### 9.1.4 Interface d'administration

Une réalisation qui revendique la conformité à l'interface d'administration en tant que serveur doit manifester l'ensemble du comportement associé à toutes les opérations et à tous les attributs en lecture seulement qui sont définis dans le domaine du type «Interface d'administration».

Une réalisation revendiquant la conformité à l'interface d'administration en tant que serveur doit également prendre en charge, en tant que serveur, l'interface de type «Itérateur d'identificateur d'offre».

### 9.1.5 Interface de procuration

Une réalisation qui revendique la conformité à l'interface de procuration en tant que serveur doit manifester l'ensemble du comportement associé à toutes les opérations qui sont définies dans le domaine du type «Interface de procuration».

Une réalisation revendiquant la conformité à l'interface de procuration en tant que serveur doit également prendre en charge, en tant que serveur, l'interface de type «Itérateur d'identificateur d'offre».

### 9.1.6 Interface d'itérateur d'offre

Une réalisation qui revendique la conformité à l'interface d'itérateur d'offre en tant que serveur doit manifester l'ensemble du comportement associé à toutes les opérations qui sont définies dans le domaine du type «Interface d'itérateur d'offre».

### 9.1.7 Interface d'itérateur d'identificateur d'offre

Une réalisation qui revendique la conformité à l'interface d'itérateur d'identificateur d'offre en tant que serveur doit manifester l'ensemble du comportement associé à toutes les opérations qui sont définies dans le domaine du type «Interface d'itérateur d'identificateur d'offre».

### 9.1.8 Interface d'évaluation de propriété dynamique

Une réalisation qui revendique la conformité à l'interface d'évaluation de propriété dynamique en tant que serveur doit manifester le comportement prescrit qui est associé à toutes les opérations définies dans le domaine d'interface de type «évaluation de propriété dynamique».

NOTE – Le comportement de cette interface est limité à la prise en charge de la signature d'interface.

### 9.1.9 Conformité à la règle de sous-typage des services

Le langage IDL-ODP est utilisé dans la présente Spécification pour exprimer des signatures d'interface d'opération de traitement. L'utilisation de cette notation n'implique pas l'utilisation de mécanismes et de protocoles spécifiques d'appui.

Pour assurer la conformité, les règles neutres de sous-typage architectural des services, indiquées au 8.2.3.2, constituent l'interprétation la plus large du concept de sous-typage pour un courtier conforme. Une réalisation de ces règles neutres de sous-typage architectural des services dans un mécanisme ou protocole spécifique ne peut pas être moins contraignante que ces règles de sous-typage de service. Un mécanisme ou protocole spécifique ne peut pas prendre en charge une définition de sous-typage non autorisée par la fonction de courtage du protocole ODP.

Les règles les plus contraignantes (en limite inférieure) pour le sous-typage d'une réalisation de courtage conforme, construite sur la base d'un mécanisme ou protocole d'appui spécifique, autorisées pour assurer la conformité, sont les suivantes:

- un type de service *b* est un sous-type d'un type de service *a* si et seulement si:
  - le type d'interface du sous-type *b* peut ajouter des opérations au type d'interface du sous-type *a*; et si
  - toutes les propriétés nommées du sous-type *a* sont dans le sous-type *b*; et si
  - toutes les propriétés nommées du sous-type *a* sont d'un type identique à celui de la propriété de nom identique du sous-type *b*; et si
  - toutes les propriétés nommées du sous-type *a* sont d'un mode identique à celui de la propriété de nom identique du sous-type *b*.

## 9.2 Prescriptions de conformité pour la classe de conformité de courtier d'interrogation

Une réalisation de système de courtage revendiquant la conformité à la classe de conformité de courtier d'interrogation doit être conforme aux règles de conformité du type «Interface de consultation» dans le rôle de serveur.

Les réalisations conformes à cette classe de conformité ne revendiqueront pas la conformité à l'interface d'enregistrement et utiliseront des moyens non normalisés pour introduire des offres de service chez le courtier.

## 9.3 Prescriptions de conformité pour la classe de conformité de courtier simple

Une réalisation de système de courtage revendiquant la conformité à la classe de conformité de courtier simple doit être conforme, en tant que serveur, aux règles de conformité des types «Interface de consultation» et «Interface d'enregistrement».

Un courtier simple doit être conforme au comportement d'objet combiné enregistrement-consultation qui est spécifié au 8.7.2.

## 9.4 Prescriptions de conformité pour la classe de conformité de courtier autonome

Une réalisation de système de courtage revendiquant la conformité à la classe de conformité de courtier autonome doit être conforme, en tant que serveur, aux règles de conformité des types «Interface de consultation», «Interface d'enregistrement» et «Interface d'administration».

Un courtier autonome doit être conforme au comportement d'objet combiné enregistrement-consultation qui est spécifié au 8.7.2.

Un courtier autonome doit être conforme au comportement d'objet combiné administration-enregistrement qui est spécifié au 8.7.2.

Un courtier autonome doit être conforme au comportement d'objet combiné administration-consultation qui est spécifié au 8.7.2.

## 9.5 Prescriptions de conformité pour la classe de conformité de courtier lié

Une réalisation de système de courtage revendiquant la conformité à la classe de conformité de courtier lié doit être conforme, en tant que serveur, aux règles de conformité des types «Interface de consultation», «Interface d'enregistrement», «Interface d'administration» et «Interface de liaison». Elle doit également être en mesure de jouer le rôle de client pour les opérations d'invocation du type «Interface de consultation».

Un courtier lié doit être conforme au comportement d'objet combiné consultation-enregistrement spécifié au 8.7.2.

Un courtier lié doit être conforme au comportement d'objet combiné administration-enregistrement spécifié au 8.7.2.

Un courtier lié doit être conforme au comportement d'objet combiné administration-consultation spécifié au 8.7.2.

Un courtier lié doit être conforme au comportement d'objet combiné consultation-liaison spécifié au 8.7.2.

## 9.6 Prescriptions de conformité pour la classe de conformité de courtier de procuration

Une réalisation de système de courtage revendiquant la conformité à la classe de conformité de courtier de procuration doit être conforme aux règles de conformité des types «Interface de consultation», «Interface d'enregistrement», «Interface d'administration» et «Interface de procuration» dans le rôle de serveur. Elle doit également être en mesure de jouer le rôle de client pour les opérations d'invocation par interface de type «consultation».

Un courtier de procuration doit être conforme au comportement d'objet combiné consultation-enregistrement spécifié au 8.7.2.

Un courtier de procuration doit être conforme au comportement d'objet combiné administration-enregistrement spécifié au 8.7.2.

Un courtier de procuration doit être conforme au comportement d'objet combiné administration-consultation spécifié au 8.7.2.

Un courtier de procuration doit être conforme au comportement d'objet combiné consultation-procuration spécifié au 8.7.2.

Un courtier de procuration doit être conforme au comportement d'objet combiné administration-procuration spécifié au 8.7.2.

## 9.7 Prescriptions de conformité pour la classe de conformité de courtier tous services

Une réalisation de système de courtage revendiquant la conformité à la classe de conformité de courtier tous services doit être conforme aux règles de conformité des types «Interface de consultation», «Interface d'enregistrement», «Interface d'administration», «Interface de liaison» et «Interface de procuration» dans le rôle de serveur. Elle doit également être en mesure de jouer le rôle de client pour les opérations d'invocation par interface de type «consultation».

Un courtier tous services doit être conforme au comportement d'objet combiné consultation-enregistrement spécifié au 8.7.2.

Un courtier tous services doit être conforme au comportement d'objet combiné administration-enregistrement spécifié au 8.7.2.

Un courtier tous services doit être conforme au comportement d'objet combiné administration-consultation spécifié au 8.7.2.

Un courtier tous services doit être conforme au comportement d'objet combiné consultation-liaison spécifié au 8.7.2.

Un courtier tous services doit être conforme au comportement d'objet combiné consultation-procuration spécifié au 8.7.2.

Un courtier tous services doit être conforme au comportement d'objet combiné administration-procuration spécifié au 8.7.2.

## 9.8 Tests de conformité

Pour chaque test de conformité, la réalisation doit identifier les points de référence de programmation dont on revendique la conformité.

Pour chaque test de conformité, le réalisateur doit indiquer:

- la politique en vigueur pendant la durée du test;
- l'état d'objet pris en compte sous la forme d'un schéma statique, par rapport à la spécification d'information, par exemple les offres de service détenues.

Une réalisation revendiquant la conformité est tenue de fournir l'ensemble des politiques ou des circonstances par lesquelles une offre exportée peut être récupérée par une opération ultérieure d'interrogation.

Les réalisations revendiquant la conformité à la classe de conformité de courtier lié ou de courtier tous services doivent être en mesure de démontrer leur capacité de propager une opération d'interrogation jusqu'à l'interface de consultation d'un courtier distant.

Les réalisations revendiquant la conformité aux classes de conformité de courtier de procuration ou de courtier tous services doivent être en mesure de démontrer leur capacité à faire suivre une opération d'interrogation jusqu'à un objet distant via une offre de procuration.

# Annex A

## ODP-IDL based specification of the Trading Function

(This annex forms an integral part of this Recommendation | International Standard)

### 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 */
```

# Annex B

## ODP Trading Function Constraint Language BNF

(This annex forms an integral part of this Recommendation | International Standard)

### 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>**

# Annex C

## ODP Trading Function constraint recipe language

(This annex forms an integral part of this Recommendation | International Standard)

### 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"**

# Annex D

## Service type repository

(This annex forms an integral part of this Recommendation | International Standard)

### 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.

## SÉRIES DES RECOMMANDATIONS UIT-T

| | |
|---|---|
| Série A | Organisation du travail de l'UIT-T |
| Série B | Moyens d'expression: définitions, symboles, classification |
| Série C | Statistiques générales des télécommunications |
| Série D | Principes généraux de tarification |
| Série E | Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains |
| Série F | Services de télécommunication non téléphoniques |
| Série G | Systèmes et supports de transmission, systèmes et réseaux numériques |
| Série H | Systèmes audiovisuels et multimédias |
| Série I | Réseau numérique à intégration de services |
| Série J | Transmission des signaux radiophoniques, télévisuels et autres signaux multimédias |
| Série K | Protection contre les perturbations |
| Série L | Construction, installation et protection des câbles et autres éléments des installations extérieures |
| Série M | RGT et maintenance des réseaux: systèmes de transmission, de télégraphie, de télécopie, circuits téléphoniques et circuits loués internationaux |
| Série N | Maintenance: circuits internationaux de transmission radiophonique et télévisuelle |
| Série O | Spécifications des appareils de mesure |
| Série P | Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux |
| Série Q | Commutation et signalisation |
| Série R | Transmission télégraphique |
| Série S | Equipements terminaux de télégraphie |
| Série T | Terminaux des services télématiques |
| Série U | Commutation télégraphique |
| Série V | Communications de données sur le réseau téléphonique |
| **Série X** | **Réseaux pour données et communication entre systèmes ouverts** |
| Série Y | Infrastructure mondiale de l'information |
| Série Z | Langages de programmation |