UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

Série Q

Supplément 28
(12/1999)

SÉRIE Q: COMMUTATION ET SIGNALISATION

Rapport technique: cadre général de signalisation et de protocole pour un environnement évolutif (SPFEE) – Spécification pour l'accès au service

Recommandations UIT-T de la série Q – Supplément 28

(Antérieurement Recommandations du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE Q

COMMUTATION ET SIGNALISATION

SIGNALISATION DANS LE SERVICE MANUEL INTERNATIONAL	Q.1–Q.3
EXPLOITATION INTERNATIONALE AUTOMATIQUE ET SEMI-AUTOMATIQUE	Q.4–Q.59
FONCTIONS ET FLUX D'INFORMATION DES SERVICES DU RNIS	Q.60–Q.99
CLAUSES APPLICABLES AUX SYSTÈMES NORMALISÉS DE L'UIT-T	Q.100–Q.119
SPÉCIFICATIONS DES SYSTÈMES DE SIGNALISATION N° 4 ET N° 5	Q.120–Q.249
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 6	Q.250–Q.309
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION R1	Q.310–Q.399
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION R2	Q.400–Q.499
COMMUTATEURS NUMÉRIQUES	Q.500–Q.599
INTERFONCTIONNEMENT DES SYSTÈMES DE SIGNALISATION	Q.600–Q.699
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 7	Q.700–Q.849
SYSTÈME DE SIGNALISATION D'ABONNÉ NUMÉRIQUE N° 1	Q.850–Q.999
RÉSEAUX MOBILES TERRESTRES PUBLICS	Q.1000–Q.1099
INTERFONCTIONNEMENT AVEC LES SYSTÈMES MOBILES À SATELLITES	Q.1100–Q.1199
RÉSEAU INTELLIGENT	Q.1200–Q.1699
PRESCRIPTIONS ET PROTOCOLES DE SIGNALISATION POUR LES IMT-2000	Q.1700–Q.1799
RNIS À LARGE BANDE	Q.2000–Q.2999

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

Supplément 28 aux Recommandations UIT-T de la série Q

Rapport technique: cadre général de signalisation et de protocole pour un environnement évolutif (SPFEE) – Spécification pour l'accès au service

Résumé

Ce supplément spécifie le modèle d'information et de calcul (niveau session) au point de référence consommateur-revendeur. Il définit la spécification d'information et de calcul que l'accès au service défini en SPFEE (*signalling and protocol framework of an evolving environment*: cadre de signalisation et de protocole d'un environnement évolutif). Ces spécifications d'information et de calcul sont décrites dans des langages de description formelle et informelle tel l'IDL (Langage de définition d'interface).

Source

Le Supplément 28 aux Recommandations UIT-T de la série Q, élaboré par la Commission d'études 11 (1997-2000) de l'UIT-T, a été approuvé le 3 décembre 1999 selon la procédure définie dans la Résolution 5 de l'AMNT.

Mots clés

Accès, IDL, modèle de calcul, modèle d'information, ODP, point de référence, session.

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.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), 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 1 de l'AMNT.

Dans certains secteurs des technologies 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 publication, 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 publication 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 publications.

A la date d'approbation de la présente publication, 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 publication. 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

Supplément 28 aux Recommandations UIT-T de la série Q

Rapport technique: cadre général de signalisation et de protocole pour un environnement évolutif (SPFEE) – Spécification pour l'accès au service

(Genève, 1999)

1 Domaine d'application

Le présent supplément fournit:

– la spécification informationnelle;

– la spécification informatique

pour l'accès au service défini dans le cadre SPFEE (Cadre général de signalisation et de protocole pour un environnement évolutif). Ces spécifications englobent les spécifications du point de référence consommateur-revendeur. Les spécifications informationnelles et informatiques sont décrites en langage ordinaire et dans un langage formel tel que IDL (Langage de définition d'interface).

2 Références

Les Rapports techniques et autres références suivants contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour le présent supplément. Au moment de la publication, les éditions indiquées étaient en vigueur. Tout supplément ou autre référence est sujet à révision; tous les utilisateurs du présent supplément sont donc invités à rechercher la possibilité d'appliquer les éditions les plus récentes des suppléments et autres références indiqués ci-après. Une liste des Recommandations et des suppléments UIT-T en vigueur est publiée régulièrement.

[1] Recommandation UIT-T X.901 (1997) | ISO/CEI 10746-1:1998, *Technologies de l'information – Traitement réparti ouvert – Modèle de référence: aperçu général*.

[2] Recommandation UIT-T X.902 (1995) | ISO/CEI 10746-2:1996, *Technologies de l'information – Traitement réparti ouvert – Modèle de référence: fondements*.

[3] Recommandation UIT-T X.903 (1995) | ISO/CEI 10746-3:1996, *Technologies de l'information – Traitement réparti ouvert – Modèle de référence: architecture*.

[4] Recommandation UIT-T X.920 (1997) | ISO/CEI 14750:1999, *Technologies de l'information – Traitement réparti ouvert – Langage de définition d'interface*.

[5] Recommandation UIT-T Z.130 (1999), *Langage de définition d'objet de l'UIT*.

[6] Recommandations UIT-T de la série Q – Supplément 27 (1999), *Rapport technique: aperçu du cadre général de signalisation et de protocole pour un environnement évolutif (SPFEE)*.

3 Définitions

Le présent supplément définit les termes suivants en plus de ceux qui sont définis dans la référence [6]:

3.1 session d'accès au domaine (D_AS): objet abstrait représentant les informations générales nécessaires à l'établissement et à la prise en charge de l'accès entre deux domaines.

3.2 session d'accès au domaine utilisateur (UD_AS): objet géré par l'utilisateur représentant l'ensemble de capacités et la configuration employés par l'utilisateur pour contacter un fournisseur.

3.3 session d'accès au domaine fournisseur (PD_AS): objet géré par le fournisseur possédant des capacités et des données propres au sein du domaine fournisseur; cet objet est créé lorsque l'utilisateur devient une entité reconnue et identifiable.

3.4 session de service fournisseur (PSS): vue centrale de la session de service incluant tous ses membres et toutes les informations et logiques supplémentaires nécessaires à l'exécution des demandes de service et la maintenance de la session.

3.5 session de service d'utilisation (USS): vue personnalisée d'un service pour un membre de la session (par exemple, un utilisateur final).

3.6 session de service d'utilisation de domaine (D_USS): objet abstrait représentant, pour un membre de session associé, les informations générales nécessaires à l'établissement et à la prise en charge d'un service entre deux domaines.

3.7 session de service d'utilisation de domaine utilisateur (UD_USS): fonctionnalités et informations présentes au niveau d'un domaine utilisateur final, tel qu'un domaine consommateur, nécessaires à la prise en charge de la session de service d'utilisation et permettant l'interaction de l'utilisateur avec le service.

3.8 session de service d'utilisation de domaine fournisseur (PD_USS): fonctionnalités et informations présentes au niveau d'un domaine fournisseur, tel qu'un revendeur ou un fournisseur tiers, nécessaires à la prise en charge de la session de service d'utilisation dans un rôle de fournisseur d'utilisation.

3.9 attachement de session de service d'utilisation de domaine (D_USS_Binding): informations dynamiques associées à l'attachement de deux sessions D_USS.

4 Abréviations

Le présent supplément utilise les abréviations suivantes:

3Pty point de référence entre domaines pour un participant tiers (*third-party inter-domain reference point*)

anonUA agent utilisateur anonyme (*anonymous user agent*)

as-UAP application utilisateur (liée à une session d'accès) [(*user application (access session related*)]

AS session d'accès (*access session*)

Bkr point de référence entre domaines pour un courtier (*broker inter-domain reference point*)

CO objet informatique (*computational object*)

ConS point de référence entre domaines pour un service de connectivité (*connectivity service inter-domain reference point*)

CS session de communication (*communication session*)

CSLN point de référence entre domaines pour la couche client-serveur (*client-server layer inter-domain reference point*)

DPE environnement de traitement réparti (*distributed processing environment*)

FCAPS fautes, configuration, comptabilisation, performances, sécurité (*fault, configuration, accounting, performance, security*)

IA agent initial (*initial agent*)

IDL langage de définition d'interface (*interface definition language*)

LNFed point de référence entre domaines pour une fédération de réseau de couche (*layer network federation inter-domain reference point*)

NamedUA agent utilisateur nommé (*named user agent*)

ODL langage de définition d'objet (*object definition language*)

ODP traitement réparti ouvert (*open distributed processing*)

OMT technique de modélisation par objets (*object modelling technique*)

PA agent fournisseur (*provider agent*)

Ret point de référence entre domaines pour un revendeur (*retailer inter-domain reference point*)

RP point de référence (*reference point*)

RtR point de référence revendeur-revendeur entre domaines (*retailer to retailer inter-domain reference point*)

SC composant de service (*service component*)

SF atelier de service (*service factory*)

SPFEE cadre général de signalisation et de protocole pour un environnement évolutif (*signalling and protocol framework for an evolving environment*)

SS session de service

SSM gestionnaire de session de service (*service session manager*)

ss-UAP application utilisateur (liée à une session de service) [*user application (service session related*)]

Tcon point de référence entre domaines pour une connexion de terminal (*terminal connection inter-domain reference point*)

UA agent utilisateur (*user agent*)

UAP application utilisateur (*user application*)

USM gestionnaire de session de service utilisateur (*user service session manager*)

5 Modèle d'informations des niveaux session et ressource

La description du modèle d'informations utilise la notation du modèle objet OMT. La Figure 5-1 indique les symboles usuels. Les classes abstraites (pour lesquelles il n'existe aucune instance d'objet) sont indiquées entre accolades[1]; leur seule fonction est de simplifier la modélisation par objets.

[1] Ceci signifie que les caractères {} sont insérés en tête du nom de la classe dans les diagrammes.

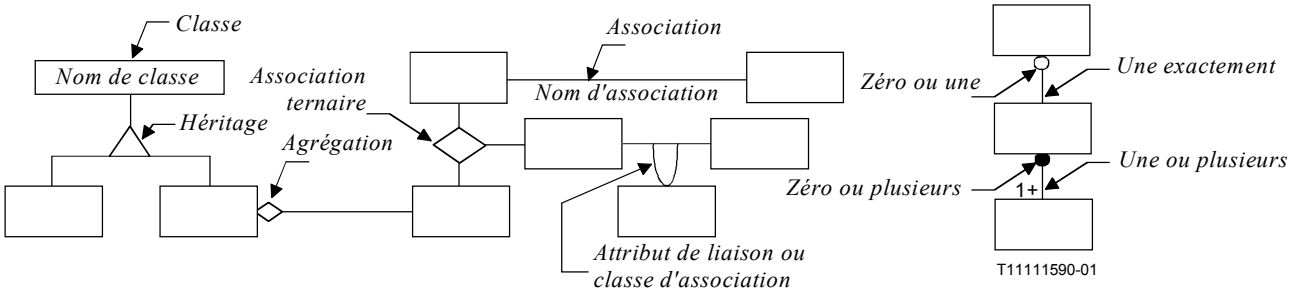


Figure 5-1 – Symboles usuels dans un diagramme OMT

5.1 Sessions, services et domaines

Le présent sous-paragraphe présente le modèle informationnel du point de vue de ses relations avec les services liés à la session[2]. Un service lié à une session peut s'étendre sur plusieurs domaines traités comme des domaines administratifs d'affaire; il est de ce fait utile de modéliser le service sous la forme d'un agrégat d'un ou de plusieurs services de domaine, dont chacun représente une partie d'un service limité à un domaine unique. Une session de domaine représente une instance de service de domaine, de la même manière qu'un service de domaine représente une instance de service. Les sessions de domaine peuvent interagir pour mettre en place des services qui s'étendent sur plusieurs domaines.

La Figure 5-2 indique les relations entre les objets domaine, service et session. Un service lié à une session est un agrégat d'autres services liés à une session et de services de domaine. Chaque service de domaine est instancié par une unique session de domaine qui est établie et gérée par le domaine associé. Une session de domaine peut être liée à une autre session de domaine par un attachement de session de domaine, qui représente les informations dynamiques utilisées pour établir la liaison entre ces sessions.

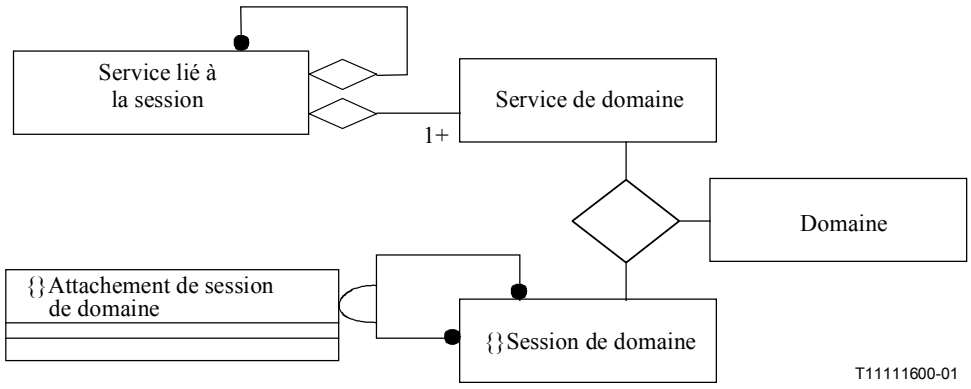


Figure 5-2 – Modèle informationnel de session

---

[2] Un service est classé selon deux types de service, à savoir les services liés à une session et les services non liés à une session. Les services non liés à une session sont en dehors du domaine d'application. Les services liés à une session sont définis comme suit: un service lié à une session représente un service de télécommunication en ligne qui est offert aux utilisateurs **en même temps** que des équipements et des ressources de télécommunication. Par exemple, tous les services de télécommunication, allant de la téléphonie classique aux services multimédias et/ou les services liés à l'Internet, sont classés comme services liés à une session.

5.2 Classification des sessions

Le présent sous-paragraphe traite de la classification des sessions. Il est possible d'utiliser un certain nombre de procédés de classification, qui sont tous basés sur l'objet "racine de session". Cet objet générique définit les propriétés communes d'une session. Tous les objets, quelle que soit leur classification, héritent ces attributs et opérations communs, tels que l'identificateur de session, le type de session, son état, ainsi que les opérations "state", "terminate", "suspend" et "resume" (respectivement *état*, *terminer*, *suspension* et *reprise*).

L'une des classifications se base sur une partition (fonctionnelle) du service en accès, service et communication. Des sessions ont été identifiées pour la prise en charge de chacune de ces partitions. Chaque session conserve les propriétés de l'objet "racine de session" tout en faisant l'objet d'une spécialisation pour la prise en charge des prescriptions propres à un domaine fonctionnel.

Il est également commode d'utiliser les concepts de session de domaine et d'attachement de session de domaine comme aide pour la classification des sessions. Dans ce cas également, l'objet "racine de session" constitue la base de la hiérarchie[3]; la session de domaine et l'attachement de session de domaine héritent de ses propriétés. Cette classification peut être combinée avec la précédente pour fournir une classification de tous les objets de chaque partition. Les paragraphes qui suivent utilisent cette combinaison pour une classification des objets tenant également compte des caractéristiques suivantes:

– caractéristiques utilisateur: représentant les entités, la sémantique, les contraintes et les règles de disponibilité et d'utilisation des capacités du service pour un utilisateur donné, ainsi que les ressources et les procédés nécessaires à la prise en charge effective de ces capacités de service pour un utilisateur donné;

– caractéristiques fournisseur: représentant les entités, la sémantique, les contraintes et les règles de disponibilité et d'utilisation des capacités du service par rapport aux utilisateurs, ainsi que les ressources et les procédés nécessaires à la prise en charge effective de ces capacités.

5.3 Classification des sessions d'accès

Il est possible de classer les sessions d'accès selon la hiérarchie de spécialisation représentée par la Figure 5-3. Cette classification utilise la distinction entre les caractéristiques utilisateur et fournisseur.

La *session d'accès au domaine* constitue, par rapport à la Figure 5-2, une classification secondaire de la session de domaine et la *session d'accès* une classification secondaire de l'attachement de session de domaine.

[3] L'objet "racine de session" est un objet abstrait, ce qui signifie qu'il ne peut pas être instancié.

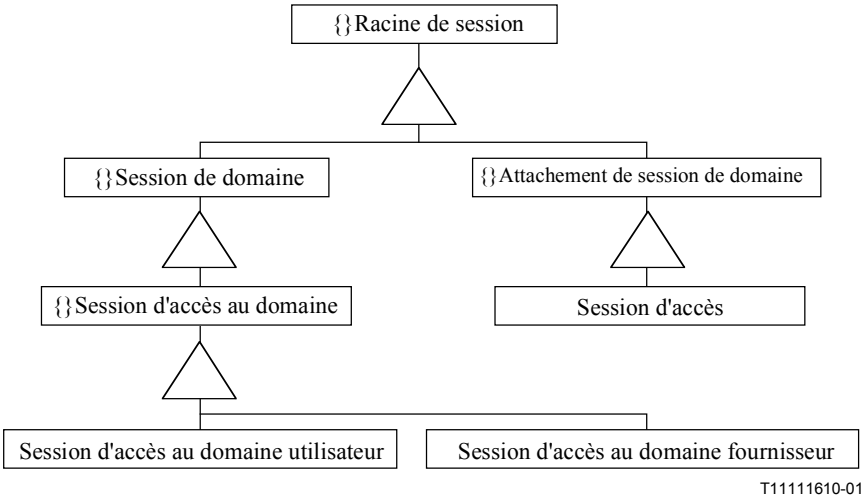


Figure 5-3 – Classification de la session d'accès

5.4 Session d'accès

La Figure 5-4 représente les objets informationnels et leurs relations en ce qui concerne l'accès.

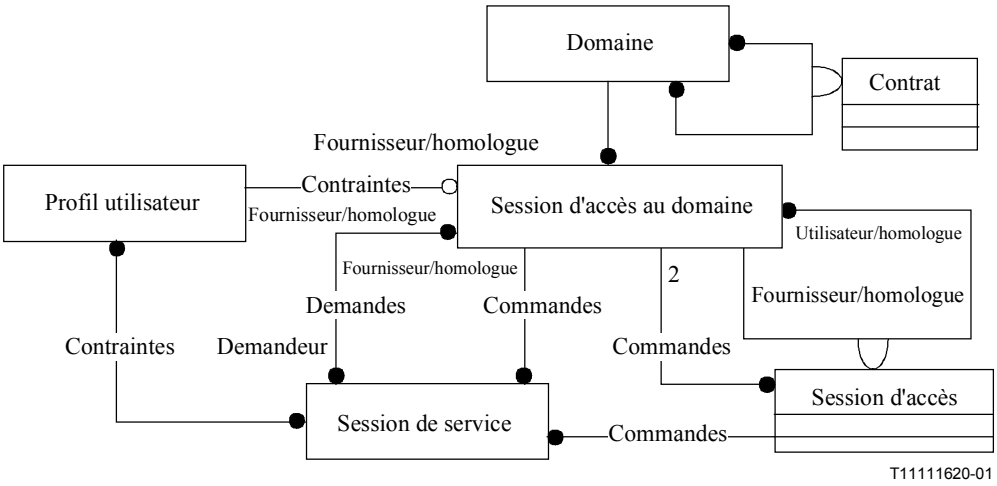


Figure 5-4 – Relations entre les objets liés à l'accès

Un rôle d'accès particulier, à savoir un rôle d'utilisateur ou un rôle de fournisseur, est associé à toute session d'accès au domaine. Le rôle utilisateur reçoit des invitations et effectue des demandes pour la session d'accès au domaine associée. Le rôle fournisseur reçoit des demandes et émet des invitations pour la session d'accès au domaine associée.

La session d'accès est classée conformément aux rôles pris en charge par chaque session d'accès au domaine, qui dépendent des relations entre les domaines. L'utilisation des types de session d'accès suivants peut être nécessaire:

– **session d'accès de type asymétrique**: l'un des domaines joue le rôle d'utilisateur et l'autre celui de fournisseur;
– **session d'accès de type symétrique**: les deux domaines prennent en charge les deux rôles de manière symétrique.

La Figure 5-5 représente les objets informationnels et leurs relations pour le type de session d'accès asymétrique. Cette figure constitue une représentation plus détaillée de la Figure 5-4 pour le type de session d'accès asymétrique. Une telle session est prise en charge par la session d'accès au domaine utilisateur et la session d'accès au domaine fournisseur.

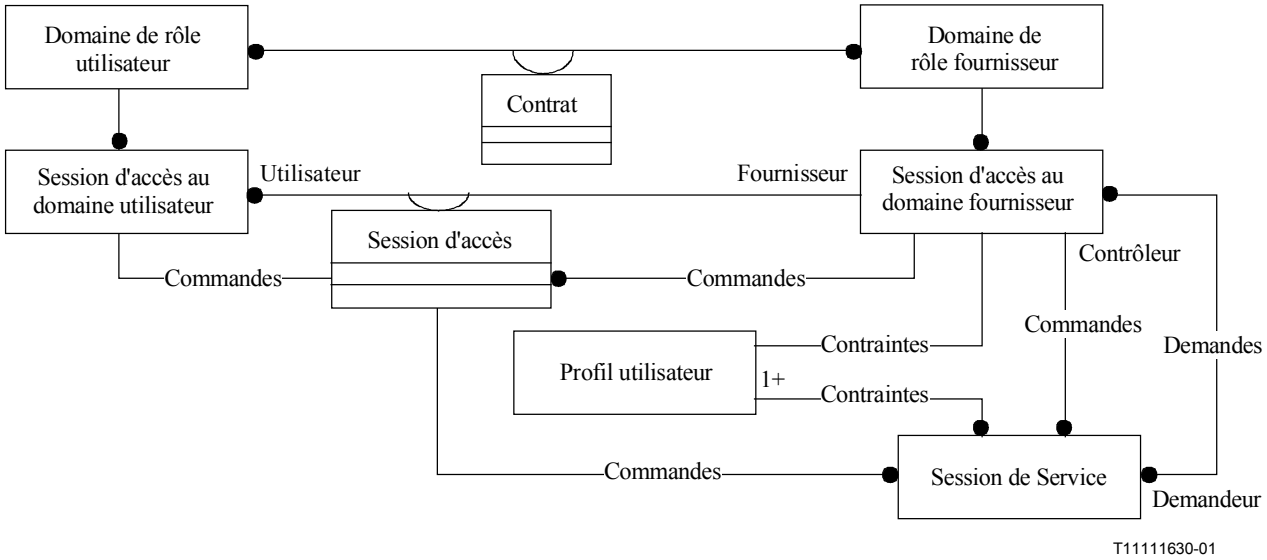


Figure 5-5 – Objets informationnels et leurs relations pour des sessions d'accès de type asymétrique

5.4.1 Session d'accès au domaine (D_AS, *domain access session*)

Cet objet abstrait représente les informations générales nécessaires à l'établissement et à la prise en charge de l'accès entre deux domaines. Chacune des sessions d'accès au domaine est associée à un domaine particulier. Un domaine peut toutefois posséder plusieurs sessions d'accès au domaine. Une session d'accès au domaine est associée en général à une ou plusieurs relations contractuelles avec un autre domaine. Il peut exister plusieurs sessions d'accès au domaine pour chaque relation contractuelle au sein du domaine.

La session D_AS fait l'objet d'une spécialisation pour les types d'objets informationnels "session d'accès au domaine utilisateur" et "session d'accès au domaine fournisseur" décrits dans les sous-paragraphes suivants.

5.4.1.1 Session d'accès au domaine utilisateur (UD_AS, *user domain access session*)

Cet objet géré par l'utilisateur représente l'ensemble de capacités et de configuration mises en œuvre par l'utilisateur pour contacter un fournisseur. La session UD_AS contient des politiques définies par l'utilisateur, telles que les politiques de sécurité et de comptabilisation, qui déterminent les conditions d'interaction avec un fournisseur; elle constitue la base de la négociation d'une session d'accès avec un fournisseur en vue d'établir une session d'accès acceptable par les deux parties. La session UD_AS peut concerner un ou plusieurs terminaux ou noeuds DPE. L'utilisateur et le fournisseur auront connaissance, quelle que soit la configuration de la session UD_AS, du niveau de confiance qui peut être accordé à cette dernière. Ce niveau de confiance (par exemple, la confidentialité, la protection par mot de passe ou les implémentations du chiffrement) sera répercuté dans les restrictions de gestion imposées à la session d'accès (par exemple, le refus de service à valeur élevée). Une session UD_AS prise en charge sur un terminal protégé contre les intrusions et mis à disposition par le fournisseur sera, pas exemple, plus digne de confiance qu'un ordinateur personnel connecté sur un petit réseau local avec des utilisateurs multiples.

5.4.1.2 Session d'accès au domaine fournisseur (PD_AS, *provider domain access session*)

Cet objet géré par le fournisseur doit être créé lorsque l'utilisateur devient une entité reconnue et identifiable possédant des capacités et des données propres au sein du domaine fournisseur. La session se termine lorsque l'utilisateur met fin à toute relation utilisateur avec le fournisseur (ce dernier met alors fin au stockage, dans le profil utilisateur, des informations concernant l'utilisateur). Cet objet a connaissance d'informations permanentes concernant l'utilisateur. Une partie de ces informations spécifie les politiques qui définissent les conditions d'interaction avec l'utilisateur, telles que les politiques de sécurité et de comptabilisation. Ces politiques constituent la base de la négociation d'une session d'accès avec l'utilisateur en vue d'établir une session d'accès acceptable par les deux parties. L'utilisateur et le revendeur auront connaissance de la confiance pouvant être accordée à la session PD_AS. Ce niveau de confiance (par exemple, la confidentialité, la protection par mot de passe ou les implémentations du chiffrement) sera répercuté dans les restrictions de gestion imposées à la session d'accès.

5.4.2 Session d'accès (AS)

Cet objet géré par l'utilisateur ou le fournisseur par le biais de la session D_AS représente l'ensemble des informations concernant un attachement dynamique entre des sessions D_AS, telles que la politique de sécurité, la comptabilisation et la description de session pour cet attachement. Sa durée de vie prend fin lorsque l'utilisateur ou le fournisseur met fin à la relation (attachement dynamique entre sessions D_AS) avec le fournisseur ou l'utilisateur.

5.4.3 Profil utilisateur

Le profil utilisateur contient toutes les informations utilisées directement par la session D_AS pour les décisions d'autorisation, les contraintes et la personnalisation des sessions D_AS, des sessions d'accès (au sein d'attachements de session d'accès) et des sessions de service.

5.5 Classification des sessions de service

La Figure 5-6 représente la hiérarchie de classification des sessions de service du point de vue de la distinction entre les caractéristiques utilisateur et fournisseur.

Cette figure complète la Figure 5-2, la session de service fournisseur (PSS, *provider service session*) et la session de service d'utilisation de domaine (D_USS, *domain usage service session*) correspondant à des sous-classes de la session de domaine et les attachements de session de service d'utilisation de domaine correspondant à des sous-classes de l'attachement de session de domaine.

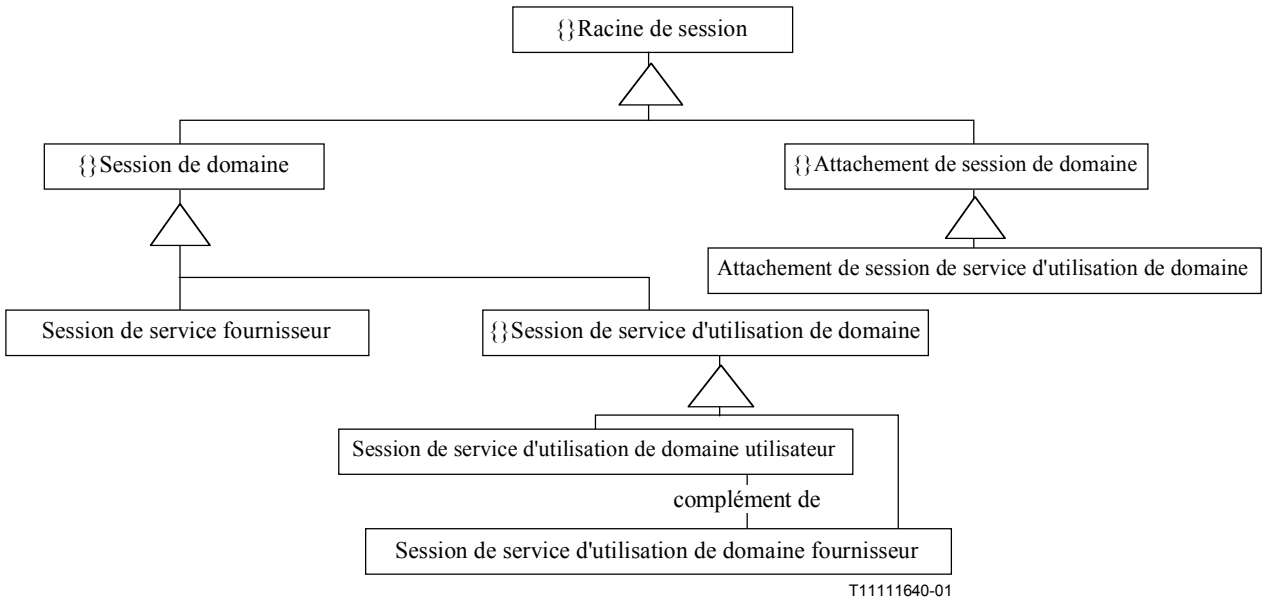


Figure 5-6 – Classification de la session de service

5.6 Session de service

La Figure 5-7 représente les objets liés à la session de service et leurs relations.

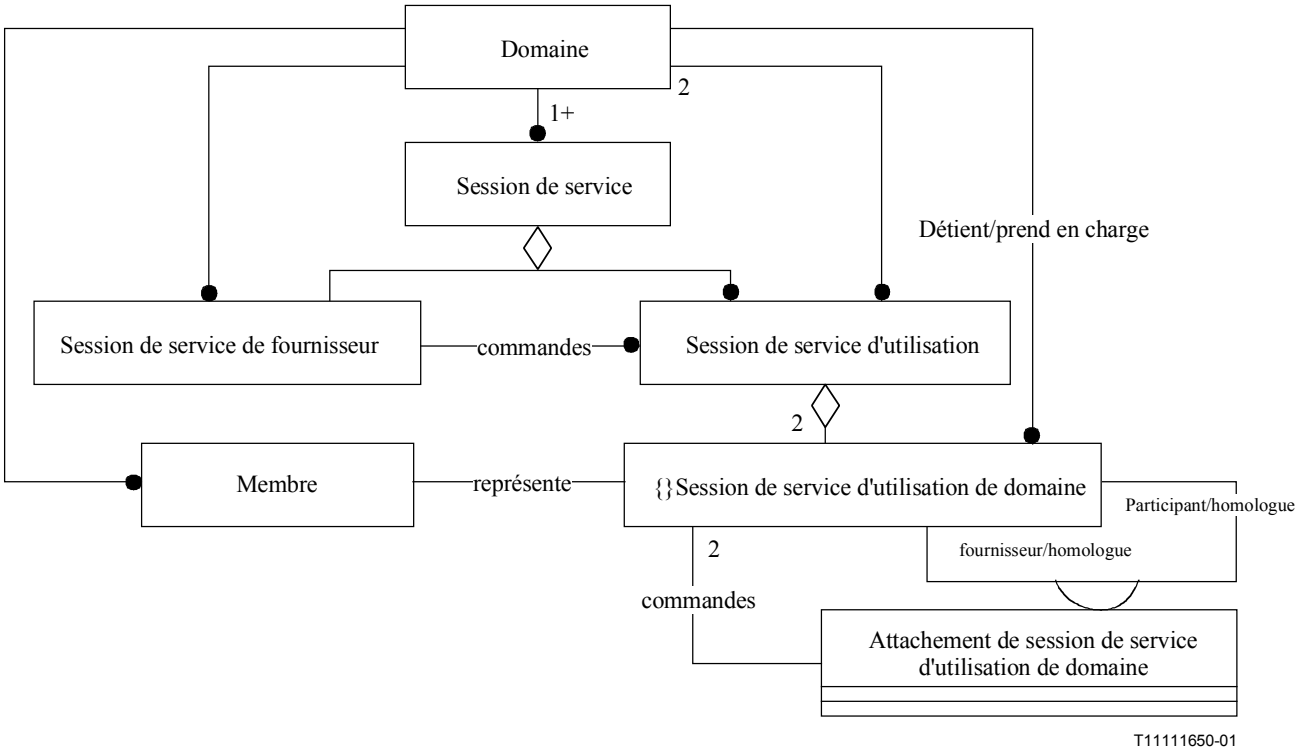


Figure 5-7 – Objets liés à la session de service et leurs relations

La session de service se constitue de sessions de service d'utilisation et de fournisseur. Une session de service d'utilisation peut concerner deux domaines et se compose de deux sessions de service d'utilisation de domaine (D_USS) complémentaires, c'est-à-dire pouvant interagir mutuellement.

Chaque membre d'une session, c'est-à-dire un utilisateur final, une ressource de service ou une session associée, est associé à une session de service d'utilisation.

Le type de session D_USS pris en charge dépend du rôle de ce membre tel qu'il est perçu au sein de la session de service. La Figure 5-6 indique les sessions D_USS possibles. Les rôles d'utilisation génériques suivants sont pris en charge:

– participant d'utilisation: se comporte comme un "utilisateur final" d'un service, c'est-à-dire un rôle actif qui peut effectuer une demande et émettre des notifications de modification de session.

– fournisseur d'utilisation: fournit un service ou agit comme une ressource utilisateur pour une autre entité. Il s'agit d'un rôle passif dans la mesure où il ne peut pas initier d'actions, mais répond à des demandes ou à des notifications de changement.

Plusieurs rôles sont possibles, englobant des rôles propres au service qui sont en dehors du domaine d'application et des rôles de commande et de gestion prenant en charge des relations dans des services composés.

Un membre qui démarre une session de service ou qui s'y joint obtient, de la session d'accès ou de la session d'accès au domaine, les informations du profil utilisateur concerné (par exemple, la description du service). Ceci impose des contraintes à la session de service d'utilisation et, potentiellement, à la session de service fournisseur. Dans le cas d'une session de service avec membres multiples, un profil utilisateur individuel ou une configuration actuelle d'utilisation peut, selon la nature du service et de ses politiques de gestion, affecter la totalité de la session.

Une session de service peut être instanciée par une session d'accès, une session d'accès au domaine ou une autre session de service. L'initiateur d'une session de service associe cette dernière à un ou plusieurs de ses membres. Les membres peuvent avoir diverses responsabilités au sein de la session (par exemple, de gestion ou limitées à l'interaction avec le contenu du service et la commande générale de la session). Si une session de service se trouve sous la responsabilité d'une session d'accès, la première peut rester active tant que cette dernière est active. Lorsqu'une session d'accès se termine, les sessions de service d'utilisation correspondantes doivent être arrêtées, suspendues ou transférées à une autre session d'accès ou session d'accès au domaine.

5.6.1 Session de service fournisseur (PSS)

Cette session constitue une vue centrale de la session de service, englobant tous les membres et toutes les informations fournisseur et logiques nécessaires à l'exécution des demandes de service et au maintien de la session. La prise en charge de ces sessions est de la responsabilité du fournisseur. Une session de service fournisseur représente les capacités de service communes à plusieurs membres. Elle possède en général des objets informationnels liés à la vue de gestion du service (par exemple, la comptabilisation) ou des informations du service liées au système.

5.6.2 Session de service d'utilisation (USS)

Cette session constitue la vue personnalisée du service pour un membre donné (par exemple, un utilisateur final). Elle masque, pour ce membre, la complexité du service et assure que ce dernier prend en charge les préférences et l'environnement du membre. Elle masque également, pour la session de service fournisseur, l'hétérogénéité entre les configurations d'utilisation. Elle sera décomposée en outre en parties d'utilisation de domaine.

5.6.3 Session de service d'utilisation de domaine (D_USS)

Cet objet abstrait représente les informations générales nécessaires à l'établissement et à la prise en charge d'un service entre deux domaines pour un membre de session associé. Chaque session de service d'utilisation de domaine est associée à un domaine particulier et à une session de service particulière. Il s'agit d'un objet abstrait qui ne peut pas être instancié. La session D_USS est spécialisée en quatre types de nouveaux objets informationnels décrits ci-dessous.

5.6.3.1 Session de service d'utilisation de domaine utilisateur (UD_USS)

Cet objet représente des fonctionnalités et des informations présentes au sein d'un domaine d'utilisateur final, par exemple, le domaine d'un consommateur, et nécessaires à la prise en charge de la session de service d'utilisation et à l'interaction de l'utilisateur final avec le service. La session UD_USS joue dans tous les cas le rôle de participant d'utilisation. Le domaine de l'utilisateur final est responsable de la mise en place et de la gestion de cette session. Un accord peut toutefois attribuer au domaine fournisseur certaines des responsabilités de mise en place et de gestion des ressources de la session UD_USS.

5.6.3.2 Session de service d'utilisation de domaine fournisseur (PD_USS, *provider domain usage service session*)

Cet objet représente des fonctionnalités et des informations présentes dans un domaine fournisseur (par exemple, un revendeur ou un fournisseur tiers) permettant la prise en charge de la session de service d'utilisation dans un rôle de fournisseur d'utilisation. Il masque, pour la session PSS, la complexité et les particularités de l'autre domaine et isole les activités propres à un participant des activités générales de session PSS, communes à tous les participants de la session de service. Le domaine qui joue le rôle de fournisseur d'utilisation est responsable de la mise en place et de la gestion de la session de service d'utilisation de domaine fournisseur.

5.6.4 Attachement de session de service d'utilisation de domaine (D_USS_Binding, *domain usage service session binding*)

Cet objet représente les informations liées à l'attachement de deux sessions D_USS. Les sessions D_USS gèrent ces informations. Les informations présentes dans ce cas sont déterminées par le type de session D_USS participant à l'attachement et par le ou les modèles de session.

5.7 Ressource [session de communication (CS)]

Cet objet représente une vue générale du service pour des connexions de flux et une vue indépendante de la technologie réseau pour les ressources de communication nécessaires à l'établissement de connexions de bout en bout. Une ressource (session de communication) peut traiter des connexions multipoint et multimédia multiples.

Une ressource (session de communication) peut déterminer la qualité de service ou procéder à l'établissement, la modification ou la suppression pour des connexions multiples.

L'adoption du concept de "session" pour la commande de capacités de communication a l'avantage de permettre aux services d'instancier de manière dynamique, de conserver, de suspendre et de maintenir une configuration adéquate des ressources de communication correspondant à leurs besoins.

Une ressource (session de communication) est gérée à partir de la session PSS ou de la session PD_USS au moyen d'une session de service unique. Une seule session de service peut être associée à une ressource (session de communication) à un instant donné.

6 Modèle informatique du niveau session et ressource

6.1 Composants liés à la session d'accès

Les composants liés à la session d'accès prennent en charge les sessions liées à l'accès[4]. Ils prennent en charge les fonctionnalités et les opérations de session communes définies pour le concept de

[4] Session d'accès (AS), session d'accès au domaine fournisseur (PD_AS) et session d'accès au domaine utilisateur (UD_AS).

session. Le 6.4 "Relation avec le modèle informationnel" indique la correspondance entre les concepts de session et les composants de service.

Une session d'accès peut être symétrique ou asymétrique. Le type de session d'accès est déterminé par le point de référence entre les domaines. Ce paragraphe examine d'abord les composants nécessaires à la prise en charge d'une session d'accès asymétrique. L'agent fournisseur prend en charge le rôle utilisateur; l'agent initial ou l'agent utilisateur prend en charge le rôle de fournisseur. Une application UAP est également impliquée dans la prise en charge des besoins de l'utilisateur final.

6.1.1 Application utilisateur

La définition de l'application utilisateur (UAP) permet de modéliser un certain nombre d'applications et de programmes au sein du domaine. Une application UAP représente une ou plusieurs de ces applications et programmes. Elle peut être mise en œuvre par des utilisateurs humains, et/ou d'autres applications au sein du domaine utilisateur. Une application UAP peut être un composant lié à une session d'accès ou à une session de service ou aux deux. L'application UAP liée à une session d'accès est définie ci-dessous. Le 6.2 définit l'application UAP liée à une session de service.

En tant que composant lié à une session d'accès, l'application UAP permet à un utilisateur humain ou à une autre application d'utiliser les capacités d'un agent fournisseur (PA) par le biais d'une interface (utilisateur) adéquate. Une application UAP liée à une session d'accès prend en charge une partie de la session d'accès au domaine. L'application UAP fournit les capacités suivantes:

– demande des informations d'authentification de l'utilisateur, nécessaires à l'agent fournisseur pour l'établissement d'une session d'accès avec un agent utilisateur;
– demande de l'utilisateur pour la création de nouvelles sessions de service;
– demande de l'utilisateur pour se joindre à une session de service existante;
– mise en alerte de l'utilisateur lorsque des invitations parviennent à l'agent fournisseur.

Une application UAP liée à une session d'accès peut également prendre en charge les capacités optionnelles suivantes, lorsqu'elles sont prises en charge par l'agent fournisseur:

– permettre à l'utilisateur de rechercher un fournisseur et de s'immatriculer comme utilisateur des services de ce dernier;
– permettre à l'utilisateur de rechercher des services et d'identifier des fournisseurs pour ces services.

Des interfaces de flux peuvent éventuellement être raccordées à une application UAP. Ces interfaces peuvent être raccordées à des interfaces correspondantes dans des systèmes utilisateur ou dans les domaines du fournisseur.

Un domaine utilisateur contient une ou plusieurs applications UAP liées à une session d'accès. Toute application UAP liée à une session d'accès peut demander à un agent fournisseur l'établissement d'une session d'accès. Une ou plusieurs applications UAP interagissent avec un agent fournisseur afin d'utiliser ses capacités liées à la session d'accès au sein d'une telle session.

Une instance d'application UAP peut prendre en charge uniquement des capacités liées à une session d'accès ou uniquement des capacités liées à une session de service ou les deux. Les applications UAP liées à une session d'accès peuvent faire l'objet d'une spécialisation par un domaine en vue d'interagir avec un agent fournisseur spécialisé.

6.1.2 Agent fournisseur

L'agent fournisseur (PA) est un composant indépendant du service qui définit le point d'extrémité de l'utilisateur pour une session d'accès. L'agent fournisseur est pris en charge dans un domaine au sein duquel il joue un rôle d'utilisateur d'accès. L'agent fournisseur prend en charge, par le biais d'une session d'accès, les activités d'un utilisateur pour l'accès à son agent utilisateur et l'utilisation de

services. L'agent fournisseur prend en charge la session d'accès au domaine utilisateur en conjonction avec des applications UAP liées à une session d'accès, ainsi qu'une autre infrastructure de domaine utilisateur.

Un agent fournisseur prend en charge les capacités suivantes:

– établissement d'une relation fiable entre l'utilisateur et le fournisseur (une session d'accès) par interaction avec un agent initial[5], avec obtention d'une référence vers un agent utilisateur[6];

– au sein d'une session d'accès:
  - transport de demandes (d'un utilisateur vers un agent utilisateur) pour la création de nouvelles sessions de service;
  - transport de demandes à se joindre à une session de service existante;
  - réception d'invitations à se joindre à des sessions de service existantes (de la part d'un agent utilisateur) et mise en alerte de l'utilisateur[7];
  - utilisation anonyme des services d'un fournisseur;
  - mise en place de nouveaux composants au sein du domaine de l'utilisateur;
  - prise en charge de l'accès aux informations de configuration de terminal en provenance d'un domaine du fournisseur;
  - immatriculation pour la réception d'invitations émises en l'absence d'une session d'accès.

Les opérations prises en charge par un agent fournisseur sont indépendantes du service.

Le domaine de l'utilisateur contient une instance d'agent fournisseur pour chaque session d'accès simultanée d'un utilisateur avec un fournisseur. Chaque agent fournisseur peut être associé (par une seule session d'accès) avec un même agent utilisateur ou avec des instances distinctes d'agent utilisateur. Un agent fournisseur unique est toujours associé à un agent utilisateur unique au moyen d'une session d'accès (un domaine utilisateur peut prendre en charge un agent fournisseur même s'il n'existe pas de session d'accès; ce domaine utilisateur peut initier une session d'accès et recevoir des invitations s'il est immatriculé).

6.1.3 Agent initial

L'agent initial (IA) est un composant, indépendant de l'utilisateur et du service, qui constitue le point d'accès initial à un domaine. Il est pris en charge par des domaines qui jouent le rôle de fournisseur.

---

[5] L'agent fournisseur peut utiliser un service de localisation ou un autre moyen pour obtenir une référence d'interface pour l'agent initial. L'agent fournisseur fournira le nom du revendeur et éventuellement d'autres informations permettant de préciser le domaine de recherche du service de localisation. L'aptitude du service de localisation de renvoyer cette référence d'interface constitue une partie importante du processus d'activation indépendant de l'emplacement qui est une fonctionnalité importante de la mobilité personnelle. Ceci présuppose que le service de localisation peut effectivement être contacté quel que soit l'emplacement. Ceci peut également impliquer qu'un interfonctionnement est nécessaire entre les services de localisation de divers domaines. La manière dont le service de localisation obtient une référence d'interface pour un agent initial n'est pas définie. Il est probable que le service de localisation devra interagir à cet effet avec un objet appartenant au domaine du fournisseur. Cette interaction n'est pas définie à ce jour.

[6] Cette capacité est un élément important de la prise en charge de la mobilité personnelle, car elle permet à un utilisateur d'accéder à un domaine fournisseur à partir d'emplacements divers.

[7] Par le biais d'une session d'accès liée à une application UAP.

Une référence d'agent initial est renvoyée à un agent fournisseur lorsque ce dernier souhaite contacter le domaine. L'agent initial prend en charge les capacités suivantes[8]:

– authentification du domaine demandeur et établissement d'une relation fiable entre les domaines (une session d'accès) au moyen d'une interaction avec l'agent fournisseur;

– établissement d'une session d'accès en permettant au domaine demandeur de rester anonyme. Ce mode d'accès correspond à un type d'agent utilisateur anonyme.

Les opérations prises en charge par un agent initial sont indépendantes du service.

Un agent initial prend en charge une seule demande issue d'un agent fournisseur à un instant donné. L'agent fournisseur effectue une demande de contact avec le domaine et reçoit la référence d'un agent initial. La référence de l'agent initial perd sa validité une fois que l'agent fournisseur a effectué une interaction avec l'agent initial en vue d'établir une session d'accès[9]. L'agent initial peut ensuite être contacté par un autre agent fournisseur.

6.1.4 Agent utilisateur

L'agent utilisateur (UA) est un composant indépendant du service qui représente un utilisateur au sein du domaine du fournisseur. Il est pris en charge par un domaine qui joue le rôle de fournisseur d'accès. Il constitue le point d'extrémité du domaine du fournisseur d'une session d'accès avec un utilisateur. Il prend en charge la session d'accès au domaine fournisseur. Il est accessible à partir du domaine de l'utilisateur, quel qu'en soit l'emplacement.

Un agent utilisateur fournit les capacités suivantes:

– prise d'une relation fiable entre l'utilisateur et le fournisseur (une session d'accès) en effectuant une référence au fournisseur de l'agent de l'utilisateur;

– au sein d'une session d'accès:
  - rôle de point de contact unique d'un utilisateur pour la commande et la gestion (création/suspension/reprise) du cycle de vie des sessions de service et des sessions de service utilisateur;
  - création d'une nouvelle session de service (en demandant à un atelier de service de créer un gestionnaire USM et un gestionnaire SSM);
  - entrée dans une session de service existante en créant une nouvelle session de service utilisateur (avec création d'un gestionnaire USM par un atelier de service);
  - résolution de l'environnement d'exécution du service pour l'utilisateur, de manière à ce que ce dernier puisse utiliser des services à partir de terminaux de types divers. Cette capacité nécessite de disposer des informations de configuration des ressources du système utilisateur (incluant les terminaux et leurs points d'accès devant être mis en œuvre ou employés ou mis à disposition pour l'utilisateur; l'accès à ces informations peut être limité par l'utilisateur/l'agent fournisseur);
  - fourniture de l'accès à des informations de contrat d'un utilisateur avec le fournisseur;
  - résolution des problèmes d'interaction entre demandes d'utilisation du service.

---

[8] Ces capacités sont disponibles quel que soit l'emplacement de l'agent fournisseur avec lequel interagit l'agent initial; elles constituent de ce fait une partie importante de la prise en charge de la mobilité personnelle, en permettant un accès utilisateur quel que soit l'emplacement.

[9] Ceci signifie que l'agent fournisseur ne doit pas conserver de référence vers un agent IA après avoir établi une session d'application. Une implémentation peut imposer que la référence vers l'agent initial ne soit pas utilisable après l'établissement de la session d'accès.

L'agent utilisateur est défini comme un composant de type abstrait (c'est-à-dire, ne pouvant pas être instancié). Les deux types dérivés suivants peuvent être instanciés:

– agent utilisateur nommé (namedUA);

– agent utilisateur anonyme (anonUA).

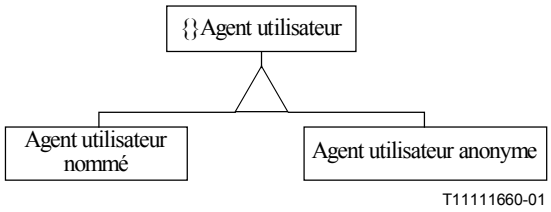


Figure 6-1 – Hiérarchie d'héritage de l'agent utilisateur

Des instances des types dérivés sont créées pour représenter divers types d'utilisateur. Les types dérivés de l'agent utilisateur prennent en charge la totalité des capacités de cet agent.

L'agent utilisateur nommé constitue une spécialisation de l'agent utilisateur pour un utilisateur final ou un abonné du fournisseur.

L'agent utilisateur anonyme constitue une spécialisation de l'agent utilisateur pour un utilisateur qui ne souhaite pas fournir son identité au fournisseur.

Ces types dérivés sont définis dans les paragraphes qui suivent.

6.1.5 Agent utilisateur nommé

L'agent utilisateur nommé (namedUA) est un composant indépendant du service, qui représente un utilisateur au sein du domaine du fournisseur. L'agent utilisateur nommé est une spécialisation de l'agent utilisateur pour un utilisateur final ou un abonné fournisseur. Il est accessible à partir du domaine de l'utilisateur quel que soit l'emplacement de ce domaine.

L'agent utilisateur nommé prend en charge toutes les capacités définies pour l'agent utilisateur. Il prend également en charge les capacités suivantes:

– au sein d'une session d'accès:

  - rôle de point de contact unique d'un utilisateur pour la commande et la gestion (création/suspension/reprise) du cycle de vie des sessions de service et des sessions de service utilisateur, compte tenu des limitations imposées par les abonnés et par l'utilisateur;
  - suspension et reprise de sessions de service utilisateur et de sessions de service existantes. Cette capacité inclut la prise en charge de la mobilité de la session;
  - gestion des préférences de l'utilisateur (choix ou contraintes) concernant l'accès au service et son exécution (cette capacité est prise en charge par le démarrage d'une session de service propre au fournisseur);
  - résolution de l'environnement d'exécution du service pour l'utilisateur, de manière à ce que ce dernier puisse utiliser des services à partir de terminaux de types divers. Cette capacité nécessite de disposer des informations de configuration des ressources du système utilisateur (incluant les terminaux et leurs points d'accès devant être mis en œuvre ou employés ou mis à disposition pour l'utilisateur; l'accès à ces informations peut être limité par l'utilisateur/l'agent fournisseur). Cette capacité inclut la prise en charge de la mobilité personnelle;

- immatriculation de l'utilisateur au niveau d'un terminal pour la réception d'invitations. Cette capacité inclut la prise en charge de la mobilité personnelle;
- permettre à l'utilisateur de définir des politiques utilisateur privées ou publiques (cette capacité est prise en charge par le démarrage d'une session de service propre au fournisseur);
- négociation de modèles de session et d'ensembles de fonctionnalités pris en charge par une session de service, en vue de permettre à cette dernière d'interagir avec une application UAP au sein du domaine de l'utilisateur;

– acceptation d'invitations faite par des utilisateurs pour se joindre à une session de service;

– envoi d'invitations à un terminal immatriculé au préalable par l'utilisateur auprès de l'agent utilisateur nommé. La livraison de ces invitations ne nécessite pas de session d'accès.

L'agent utilisateur nommé peut prendre en charge les capacités optionnelles suivantes:

– effectuer des actions pour le compte d'utilisateurs qui ne sont pas engagés dans une session d'accès avec l'agent utilisateur nommé;

– initier une session d'accès avec un agent fournisseur;

– prise en charge d'une authentification supplémentaire de l'utilisateur. Cette fonctionnalité peut être personnalisée pour l'utilisateur et pour le contexte d'utilisation.

Les opérations prises en charge par un agent utilisateur nommé sont indépendantes du service.

Un agent utilisateur nommé peut prendre en charge une ou plusieurs sessions d'accès simultanées[10]. Chacune de ces sessions d'accès concerne un agent fournisseur unique et distinct.

6.1.6 Agent utilisateur anonyme

L'agent utilisateur anonyme (anonUA) est un composant indépendant du service qui représente utilisateur au sein du domaine du fournisseur. L'agent utilisateur anonyme est une spécialisation de l'agent utilisateur pour des utilisateurs qui ne souhaitent pas révéler leur identité au fournisseur. Il constitue le point d'extrémité du domaine du fournisseur d'une session d'accès avec l'utilisateur anonyme.

L'agent utilisateur anonyme prend en charge toutes les capacités définies pour l'agent utilisateur. Il fournit également les capacités suivantes:

– prise en charge d'une relation fiable entre l'utilisateur et le fournisseur (une session d'accès) par une référence au fournisseur de l'agent utilisateur. Le fournisseur ignore l'identité de l'utilisateur (la "confiance" n'est pas garantie par l'identification de l'utilisateur, comme dans le cas de l'agent utilisateur nommé, mais peut, par exemple, être établie moyennant un paiement préalable);

[10] Les agents utilisateur nommés doivent être en mesure de prendre en charge une session d'accès et, de manière optionnelle, plusieurs sessions d'accès simultanées. L'existence de ces agents se prolonge au-delà de la session d'accès.

– au sein d'une session d'accès:

- suspension et reprise de sessions de service utilisateur et de sessions de service existantes (la reprise de sessions suspendues ne peut pas s'effectuer dans une session d'accès différente[11]);
- gestion des préférences de l'utilisateur (choix ou contraintes) concernant l'accès au service et son exécution (ces capacités doivent être déterminées au cours de la session d'accès et ne peuvent pas être réutilisées dans une autre session d'accès);
- fourniture de l'accès à des informations de contrat d'un utilisateur avec le fournisseur (ce contrat doit être défini lors du démarrage de la session d'accès et prend fin avec cette session);
- définition de politiques utilisateur privées et publiques (cette capacité peut être prise en charge par le démarrage d'une session de service propre au fournisseur; les informations sont conservées uniquement pendant la durée de la session d'accès);
- permettre à l'utilisateur anonyme de s'immatriculer comme un utilisateur du fournisseur (c'est-à-dire, établir un contrat avec le fournisseur pour une durée supérieure à une session d'accès seulement);
- négociation de modèles de session et d'ensembles de fonctionnalités pris en charge par une session de service, en vue de permettre à cette dernière d'interagir avec une application UAP au sein du domaine de l'utilisateur.

L'agent utilisateur anonyme ne prend pas en charge la mobilité personnelle ou de session.

6.2 Composants liés à la session de service

Les composants liés à la session de service définis dans ce paragraphe correspondent aux concepts de session définis dans la référence [6]. Les sessions de service peuvent être prises en charge dans des domaines multiples. Ces composants prennent en charge les sessions liées au service[12].

Les interactions entre domaines dépendent du rôle joué par le domaine dans la session de service. Un domaine jouant le rôle de fournisseur d'utilisation prendra en charge un gestionnaire de session de service. Il prendra également en charge des gestionnaires de session de service utilisateur (USM) pour chacun des domaines jouant le rôle de participant d'utilisation. Le domaine participant prend en charge une application UAP liée à une session de service à des fins d'interaction avec un gestionnaire USM.

Un domaine peut être perçu par le domaine fournisseur comme un domaine participant, alors qu'il associe en fait cette session de service avec la sienne.

Les modèles de session fournissent une définition générale du mode d'interaction des composants de session de service au sein de chaque domaine. Ils permettent, à des fins de prise en charge de la session de service, l'interaction de composants qui ont été conçus et implémentés de manière distincte.

Le modèle de session permet aux composants d'effectuer des demandes concernant, par exemple, la fin et la suspension de la session, les participants impliqués ou l'établissement et la modification d'attachements de flux entre participants.

---

[11] On fait l'hypothèse que le fournisseur met fin aux sessions suspendues lorsque la session d'accès se termine.

[12] Session de service (SS), session de service fournisseur (PSS), session de service d'utilisation (USS), session de service d'utilisation de domaine fournisseur (PD_USS) et session de service d'utilisation de domaine utilisateur (UD_USS).

Les composants liés à la session de service peuvent prendre en charge un ou plusieurs modèles de session, qui peuvent être définis par un certain nombre d'organismes. Toute session peut prendre en charge un ou plusieurs modèles de session. Les services peuvent décider de ne pas prendre en charge le modèle de session proposé, compte tenu du fait que la procédure d'accès permet de négocier des variantes d'interface d'utilisation.

6.2.1 Application utilisateur

L'application utilisateur (UAP) est définie à des fins de modélisation d'un certain nombre d'applications et de programmes au sein du domaine utilisateur. Une application UAP représente une ou plusieurs de ces applications et programmes du modèle informatique. Une application UAP peut être mise en œuvre par des utilisateurs humains ou d'autres applications appartenant au domaine utilisateur. Une application UAP est un composant qui peut être lié à une session d'accès ou à une session de service ou aux deux. L'application UAP liée à une session d'accès a été définie dans le 6.1.1. L'application UAP liée à une session de service est définie ci-dessous.

En tant que composant lié à une session de service, l'application UAP permet à un utilisateur de mettre en œuvre les capacités d'une session de service par le biais d'une interface utilisateur adéquate. Elle joue le rôle de point d'extrémité d'une session de service qui prend en charge la session de service d'utilisation de domaine utilisateur (UD_USS). Les capacités fournies par une application UAP donnée sont propres à cette dernière et à toute session de service dont elle fait partie. Les applications UAP peuvent fournir à l'utilisateur certaines capacités générales, telles que les suivantes:

- démarrage et fin de la session;
- invitation d'autres utilisateurs à se joindre à la session;
- entrée dans une session existante;
- ajout, suppression ou modification d'attachements de flux et de la participation des utilisateurs concernés;
- mise en place de relations de session de commande et autres modifications de la session de service;
- suspension de la participation d'un utilisateur dans la session ou de la totalité de la session;
- reprise de la participation d'un utilisateur dans la session ou de la totalité de la session.

Des interfaces de flux peuvent éventuellement être attachées à une application UAP[13]. Ces interfaces peuvent être raccordées à d'autres interfaces de flux situées dans d'autres systèmes utilisateur, dans le domaine fournisseur ou les deux.

Le domaine de l'utilisateur contient une ou plusieurs applications UAP liées à une session de service. Une application UAP liée à une session de service peut être impliquée dans une ou plusieurs sessions de service. Elle interagit, dans chaque session de service où elle est impliquée, avec un gestionnaire d'utilisateur de session de service ou avec un gestionnaire de session de service[14].

L'application UAP peut également prendre en charge un modèle de session particulier, tel que le modèle de session proposé. Un gestionnaire USM ou SSM utilise ces interfaces pour partager avec l'application UAP des informations concernant les participants et les ressources de service dans la session de service.

---

[13] Certains services ne nécessitent pas d'interfaces de flux pour l'application UAP.

[14] Les services qui prennent en charge un seul utilisateur dans une session de service fournissent uniquement un gestionnaire SSM (sans gestionnaire USM) et permettent l'interaction directe de l'application UAP avec le gestionnaire SSM. Ces services seront limités à un seul utilisateur pour toute session de service et ne doivent pas avoir la capacité d'inviter d'autres utilisateurs à se joindre à la session, du fait qu'il ne sera pas possible de disposer d'un gestionnaire USM pour l'utilisateur invité.

Une instance d'application UAP peut prendre en charge des capacités liées à une session d'accès, des capacités liées à une session de service ou les deux. Les applications UAP liées à une session de service seront spécialisées pour la ou les sessions de service au sein desquelles elles interagissent. Pour qu'une session de service puisse être utilisée, il est nécessaire qu'une application UAP spécialisée pour le type de service soit présente au sein du domaine utilisateur (ou mise en place, par exemple par téléchargement).

6.2.2 Atelier de service

Un atelier de service (SF) est un composant propre au service qui créée les composants de session de service pour un type de service.

Une demande de création d'une session de service pour un type de service donné conduira à la création d'une ou de plusieurs instances de composant lié au service[15]. L'atelier de service procédera à la création et à l'initialisation des instances conformément aux règles imposées par leur implémentation. L'atelier de service renverra au client une ou plusieurs références d'interface vers ces composants (l'atelier de service est utilisé pour créer des instances de tous les composants liés à la session de service définis dans le présent document, à savoir les gestionnaires USM et SSM).

Les demandes sont faites en général par des agents utilisateur. D'autres clients peuvent également être en mesure de demander la création d'une session de service. Le client doit posséder une référence de l'interface vers l'atelier de service et émettre une demande adéquate. Un atelier de service qui prend en charge plusieurs types de service fournira en général une interface distincte pour chaque type de service.

Un atelier de service prend en charge la capacité suivante:

– création, à la demande, de composants liés au service pour un ou plusieurs types de service (cette capacité inclut le choix des modèles pris en charge par la session de service, ce choix pouvant toutefois également être déterminé par le type de service).

Un atelier de service peut prendre en charge les capacités optionnelles suivantes:

– création d'un composant lié au service (en général, un gestionnaire USM) qui doit être utilisé en conjonction avec d'autres composants liés au service (en général, des gestionnaires SSM et USM) créés par une autre instance d'atelier de service;
– poursuite de la gestion des composants créés. Il peut fournir une liste des sessions qu'il gère et "nettoyer" certaines sessions à la demande;
– mécanismes éventuels d'ordonnancement de l'activation d'une session pour une date et une heure données (un tel mécanisme inclut la réservation des ressources);
– prise en charge de la suspension et de la reprise d'une session de service.

L'atelier de service assemble les ressources nécessaires à la création d'un composant. Il représente de ce fait un domaine d'allocation de ressources correspondant à l'ensemble des ressources dont il dispose. Un atelier de service peut prendre en charge une interface permettant à des clients de limiter ce domaine.

6.2.3 Gestionnaire de session de service

Le gestionnaire de session de service (SSM) est un composant qui englobe les parties spécifiques et générales de la commande de session de service fournisseur (PSS). Un gestionnaire SSM prend en charge des capacités de service partagées entre les membres (participants, ressources de service, etc.) au sein d'une session de service. Les informations liées à un membre donné de la session de service sont encapsulées dans des gestionnaires de session de service d'utilisation de membre (MUSM,

[15] En général les gestionnaires USM et SSM.

member usage service session manager). Les gestionnaires SSM prennent en charge tout ou partie des capacités suivantes:

– gestion des diverses ressources partagées entre des utilisateurs multiples au sein d'une session de service. Ceci peut se limiter à l'utilisation de références vers d'autres objets (tels qu'un gestionnaire CSM) qui gèrent effectivement le contexte d'utilisation d'un type de ressource donné;

– gestion de l'état de la session de service avec prise en charge de sa suspension et de son rétablissement;

– prise en charge de l'ajout, de l'invitation et de la suppression d'utilisateurs pour la session de service au moyen d'une interaction avec les agents utilisateur correspondants;

– prise en charge de l'ajout, de la suppression et de la modification d'attachements de flux et de la participation des utilisateurs qu'ils concernent;

– prise en charge des capacités de négociation entre utilisateurs au moyen d'une interaction avec les gestionnaires USM. Le gestionnaire SSM servira de centre de commande pour l'obtention d'un consensus (par exemple, par le biais de procédures de vote);

– prise en charge de capacités de gestion associées à la session de service (par exemple, la comptabilisation).

Des interfaces de flux peuvent éventuellement être attachées à un gestionnaire SSM[16]. Ces interfaces peuvent être raccordées à d'autres interfaces de flux situées dans ce domaine ou dans un autre domaine.

Un gestionnaire SSM est créé par un atelier de service, à raison d'une occurrence pour chaque demande de type de service correspondant. Il est supprimé lorsque tous les utilisateurs quittent la session de service ou lorsqu'il est abandonné par un utilisateur ou par un atelier de service. La durée de vie d'un gestionnaire SSM est identique à celle de la session de service fournisseur correspondante.

6.2.4 Gestionnaire de session de service d'utilisation de membre

Le gestionnaire de session de service d'utilisation de membre (MUSM) est un composant abstrait qui englobe les parties spécifiques et génériques de la commande de session de service d'utilisation de domaine (D_USS) qui interagit avec la session PSS. Il est spécialisé conformément au rôle de membre de session pris en charge par la session D_USS:

– le gestionnaire de session de service utilisateur (USM) représente la session UD_USS.

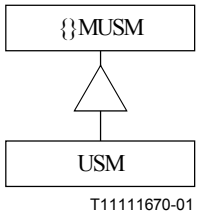


Figure 6-2 – Hiérarchie d'héritage du gestionnaire de session d'utilisation de membre

[16] Les interfaces de flux sont fournies en général par les composants de prise en charge de service (SSC) associés à un moniteur SSM.

Le gestionnaire MUSM représente et mémorise le contexte d'un membre (participant, ressource de service ou fournisseur) au sein d'une session de service. Il possède les caractéristiques principales suivantes:

– il contient les informations et les capacités de service locales du membre. Une opération impliquant des activités du membre purement locales peut être commandée et gérée directement par le gestionnaire MUSM. Dans le cas contraire, le gestionnaire MUSM interagit avec le gestionnaire SSM pour effectuer l'opération. Le gestionnaire SSM peut interagir avec le gestionnaire MUSM en réponse à des opérations d'autres membres (ou résultant de la logique de service) qui affectent ce membre. De telles interactions sont fonction du rôle du membre;
– il conserve la trace et effectue la commande des ressources (non partagées) utilisées de manière exclusive par le membre durant une session de service. Ceci peut se faire simplement par référence à d'autres objets (par exemple, un gestionnaire de session de communication) qui gèrent effectivement le contexte d'un membre pour un type de ressource donné;
– il peut être configuré en fonction des préférences du membre, soit au moment de l'initialisation, soit de manière dynamique en cours de session.

Le gestionnaire MUSM est un composant de service abstrait qui ne donne pas lieu à la création d'une instance. Des instances de composant spécialisé adéquates sont créées pour représenter des membres de session particuliers.

6.2.5 Gestionnaire de session de service utilisateur

Le gestionnaire de session de service utilisateur (USM) est un composant qui englobe les parties spécifiques et générales de la commande de session de service utilisateur de domaine fournisseur (PD_USS). Il s'agit d'une spécialisation du gestionnaire MUSM qui représente et mémorise le contexte d'un participant[17] ou d'une ressource au sein d'une session de service. Il possède les mêmes caractéristiques que le gestionnaire MUSM (le membre étant remplacé selon le cas par le participant ou la ressource de service):

– il mémorise l'état la session PD_USS et prend en charge la suspension et la reprise d'un participant au sein de la session de service;
– il prend en charge les divers rôles du participant au sein du service. Le rôle d'un participant dépend du service (il peut, par exemple, être le président d'une conférence).

Des interfaces de flux peuvent éventuellement être raccordées à un gestionnaire USM[18]. Ces interfaces peuvent être raccordées à d'autres interfaces de flux situées dans ce domaine ou dans un autre domaine.

Un gestionnaire USM est créé par un atelier de service, à raison d'un par demande pour le type de service correspondant (par session PD_USS). Il est supprimé lorsque le participant quitte la session du service. La durée de vie d'un gestionnaire USM est identique à celle de la session PD_USS correspondante.

---

[17] Un participant peut être soit un utilisateur final, soit une session de service jouant le rôle de participant d'utilisation.

[18] Les interfaces de flux sont fournies en général par les composants de prise en charge de service (SSC) associés à un moniteur USM.

6.3 Composants liés à la ressource (session de communication)

Il convient de noter que les services qui n'utilisent pas d'attachement de flux ne possèdent pas de ressources (sessions de communication) et n'ont pas besoin de ces composants. Il est toutefois probable qu'un gestionnaire CSM ou TCSM sera présent dans la plupart des domaines, du fait que les composants ne dépendent pas du service.

6.3.1 Gestionnaire de session de communication

Le gestionnaire de session de communication (CSM, *communication session manager*) est un composant indépendant du service, qui gère, au niveau application, des attachements de bout en bout entre des interfaces de flux (connexion d'écoulement de flux). Un écoulement de flux est la représentation abstraite d'une connexion. Le gestionnaire CSM fournit des interfaces qui permettent à des gestionnaires USM ou SSM d'établir, de modifier ou de supprimer des écoulements de flux. Le gestionnaire CSM décompose la demande de connexion en deux parties: la partie nodale et la partie de transport. Il demande au gestionnaire TCSM de traiter la partie nodale et à d'autres objets de gestion de connexion de traiter la partie transport.

Le gestionnaire CSM fournit la capacité suivante:

– création et commande de connexion d'écoulement de flux (SFC, *stream flow connection*) de bout en bout.

6.3.2 Gestionnaire de session de communication terminale

Le gestionnaire de session de communication terminale (TCSM, *terminal communication session manager*) est un composant indépendant du service, qui gère des connexions de flux terminales (TFC) (connexions de flux entre noeuds) au sein du domaine de l'utilisateur. Le gestionnaire TCSM fournit au gestionnaire CSM une interface qui permet à ce dernier de lui demander l'établissement, la modification et la suppression de connexions au sein du domaine de l'utilisateur.

Le gestionnaire TCSM fournit la capacité suivante:

– création et commande de connexion de flux (TFC) au sein du domaine utilisateur.

6.4 Relations avec le modèle informationnel

Les Tableaux 6-1 et 6-2 indiquent la correspondance entre les concepts et les objets de session du modèle informationnel d'une part, et les composants du modèle informatique d'autre part.

La mise en correspondance des concepts de session avec un composant signifie que ce dernier prend en charge les fonctionnalités et l'état de la session et gère les ressources de service qui font partie de la session. Si un concept de session est mis en correspondance avec plusieurs composants, chacun de ces derniers prend alors en charge une partie de la fonctionnalité et de l'état et gère une partie des ressources de service de la session.

La mise en correspondance d'objets informationnels avec un composant signifie que les informations figurant dans l'objet informationnel sont contenues dans le composant et que le composant est en mesure de fournir à d'autres composants ou à d'autres objets l'accès aux informations en question.

Tableau 6-1 – Mise en correspondance de composants liés à la session d'accès

Concept de session/Objets informationnels	Composants
Session d'accès (AS) avec rôles utilisateur-fournisseur	Agent fournisseur et agent utilisateur
Session d'accès au domaine utilisateur (UD_AS)	Agent fournisseur
Session d'accès au domaine fournisseur (PD_AS)	Agent utilisateur
Profil utilisateur avec rôles utilisateur-fournisseur	Agent utilisateur
Contrat avec rôles utilisateur-fournisseur	Agent fournisseur et agent utilisateur

Tableau 6-2 – Mise en correspondance de composants liés à la session de service

Concept de session/Objets informationnels	Composants
Session de service (SS)	Application UAP liée à une session de service, à un gestionnaire USM et à un gestionnaire SSM
Session de service utilisateur (USS)	Application UAP liée à une session de service et à un gestionnaire USM
Session de service d'utilisation de domaine utilisateur (UD_USS)	Application UAP liée à une session de service
Session de service d'utilisation de domaine fournisseur (PD_USS)	Gestionnaire USM
Session de service fournisseur (PSS)	Gestionnaire SSM

6.5 Exemples

Les sous-paragraphes qui suivent décrivent des exemples illustrant l'interaction possible entre composants dans les cas suivants:

– prise de contact avec un fournisseur,
– procédure d'entrée en relation avec un fournisseur en tant qu'utilisateur connu,
– démarrage d'une nouvelle session de service,
– invitation d'un utilisateur à se joindre à une session de service existante,
– entrée dans une session de service existante,
– création d'un attachement de flux au sein d'une session de service existante.

Il convient de noter que les scénarios sont des exemples qui représentent uniquement, pour plus de simplicité, les cas de réussite des opérations (en l'absence d'erreurs, de fautes et de rejets).

6.5.1 Prise de contact avec un fournisseur

Cet exemple traite le cas de la prise de contact d'un utilisateur A avec un fournisseur. Ce scénario prend en charge la mobilité de l'utilisateur en lui permettant de contacter un fournisseur donné à partir d'un terminal quelconque.

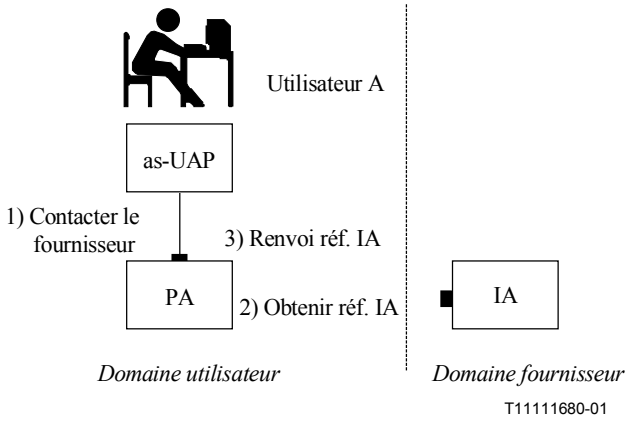


Figure 6-3 – Prise de contact avec un fournisseur

Conditions préalables

Un agent fournisseur doit être présent au sein du domaine de l'utilisateur.

Scénario

1) L'utilisateur démarre une application UAP liée à une session d'accès. Il fournit le nom du revendeur qu'il souhaite contacter. L'application UAP demande à l'agent fournisseur de prendre contact avec le fournisseur en indiquant le nom du revendeur.
2) L'agent fournisseur obtient la référence de l'interface d'un agent initial du fournisseur[19].
3) L'agent fournisseur renvoie à l'application UAP une indication de réussite.

Conditions postérieures

L'agent fournisseur possède la référence d'une interface vers l'agent initial. L'utilisateur n'a pas établi de session d'accès entre l'agent fournisseur et l'agent initial. Ce dernier ne prend pas en charge l'accès à un service de l'utilisateur, mais fournit uniquement des opérations permettant d'établir une session d'accès pour un utilisateur connu (voir 6.5.2) ou vers un utilisateur anonyme.

L'agent initial n'a connaissance d'aucune interface de l'agent fournisseur ou appartenant au domaine de l'utilisateur.

Le fournisseur peut décharger, pour le domaine de l'utilisateur, un agent fournisseur propre au fournisseur une fois qu'il a obtenu une référence d'interface vers l'agent initial. Cette fonctionnalité facilite la prise en charge de la mobilité de l'utilisateur. Aucun scénario ne décrit à l'heure actuelle comment atteindre ce but.

6.5.2 Mise en relation avec un fournisseur en tant qu'utilisateur connu

Cet exemple traite le cas d'un utilisateur A qui établit une session d'accès avec son agent utilisateur nommé au sein du fournisseur (voir Figure 6.4). L'utilisateur souhaite mettre en œuvre des services d'un fournisseur auprès duquel il a souscrit un abonnement préalable (voir 6.5.3). Ce scénario prend en charge la mobilité de l'utilisateur en lui permettant de contacter un fournisseur donné à partir d'un terminal quelconque.

[19] L'agent fournisseur peut utiliser un service de localisation ou d'autres moyens pour obtenir une référence d'interface pour l'agent initial.

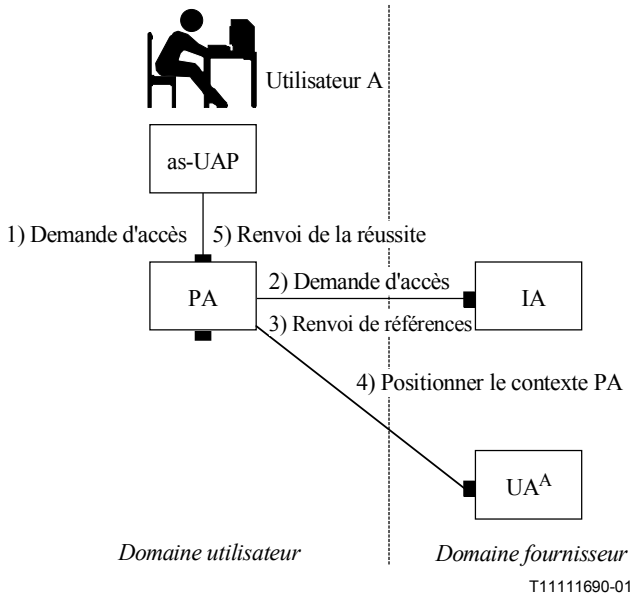


Figure 6-4 – Mise en relation avec un fournisseur en tant qu'utilisateur connu

Conditions préalables

L'utilisateur a contacté le fournisseur (comme au 6.5.1) et l'agent fournisseur possède une référence d'interface vers un agent initial du fournisseur.

Scénario

1) L'utilisateur A met en œuvre une application UAP liée à une session d'accès pour se mettre en relation avec le fournisseur en tant qu'utilisateur connu[20]. Il demande ensuite à l'agent fournisseur de se mettre en relation avec le fournisseur en tant qu'utilisateur connu. L'application UAP fournit les informations de sécurité à l'agent fournisseur.

2) L'agent fournisseur demande l'établissement d'une session d'accès avec l'agent utilisateur nommé relatif à l'utilisateur. L'agent fournisseur indique à l'agent initial le nom de l'utilisateur. L'agent fournisseur a transféré les informations de sécurité de l'utilisateur vers les services de sécurité pris en charge par l'environnement DPE. Ces services de sécurité effectuent une interaction avec le domaine fournisseur en vue d'authentifier l'utilisateur[21].

3) L'agent initial a authentifié l'utilisateur par le biais des services de sécurité de l'environnement DPE et établi une session d'accès. Il renvoie la référence de l'interface de l'agent utilisateur.

4) L'agent fournisseur envoie à l'agent utilisateur des informations concernant le domaine utilisateur. Ces informations constituent le contexte de l'agent fournisseur, incluant des références aux interfaces de l'agent fournisseur et éventuellement des informations de terminal.

5) L'agent fournisseur renvoie à l'application UAP une indication de réussite.

---

[20] L'application UAP peut demander à l'utilisateur son nom d'utilisateur et d'autres informations liées à la sécurité, par exemple un mot de passe.

[21] L'agent fournisseur doit commencer par envoyer à l'agent initial des informations d'authentification si l'environnement DPE ne prend pas en charge les services de sécurité.

Conditions postérieures

L'utilisateur a établi une session d'accès entre l'agent fournisseur et un agent utilisateur nommé. Ce dernier est personnalisé pour l'utilisateur et connaît les interfaces de l'agent fournisseur.

Toute référence d'interface de l'agent initial détenue par l'agent fournisseur devient non valide.

Il convient de noter que cette succession d'événements prend en charge la mobilité personnelle en fournissant les capacités suivantes:

a) capacité de contacter le service de dénomination quel que soit l'emplacement et de renvoyer une référence à l'agent initial;

b) capacité de l'agent initial pour l'établissement d'une relation fiable avec l'utilisateur, indépendamment de l'emplacement physique de ce dernier;

c) capacité de l'agent initial pour le renvoi vers l'utilisateur d'une référence de son propre agent utilisateur nommé (agent fournisseur agissant pour le compte de l'utilisateur).

Une fois qu'une session d'accès a été établie, le fournisseur a la possibilité de décharger, pour le domaine utilisateur, un agent fournisseur propre au fournisseur. Cette capacité facilite la prise en charge de la mobilité de l'utilisateur. Aucun scénario ne décrit à l'heure actuelle comment atteindre ce but.

6.5.3 Démarrage d'une nouvelle session de service

Cet exemple traite le cas d'une nouvelle session de service par un utilisateur (il s'agit, dans cet exemple, d'un service de visioconférence, mais les interactions sont les mêmes pour tout autre type de service). On fait l'hypothèse que l'utilisateur est engagé dans une session d'accès avec le fournisseur et dispose d'un abonnement valide pour le service (le type de service est "visioconférence 234"). On fait également l'hypothèse que l'application UAP liée à une session de service est présente au niveau du terminal de l'utilisateur (voir Figure 6-5).

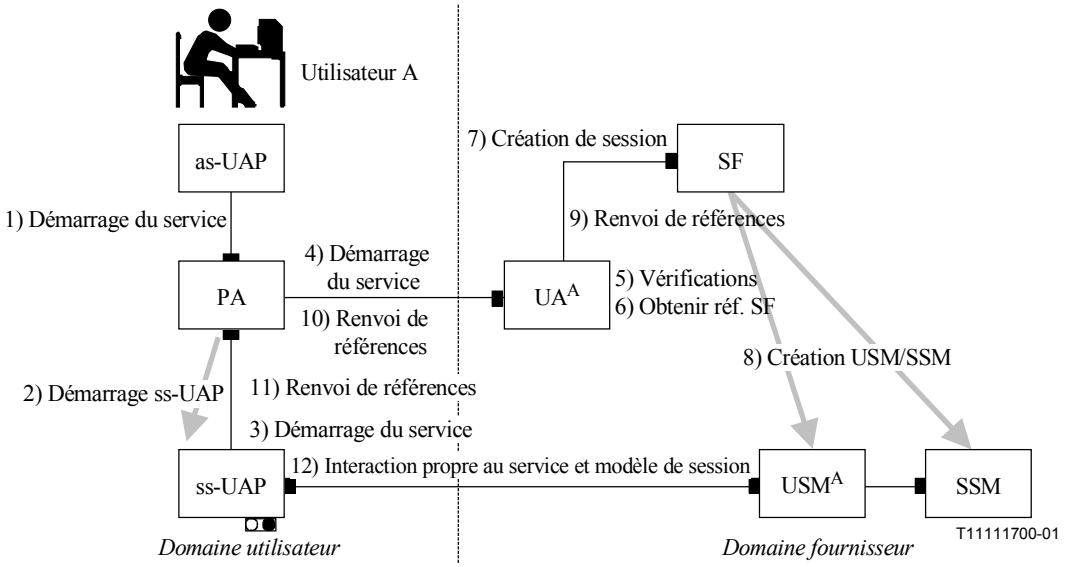


Figure 6-5 – Démarrage d'une nouvelle session de service

Conditions préalables

Une session d'accès existe entre l'agent fournisseur (utilisateur A) et un agent utilisateur (au sein du domaine fournisseur). Une application UAP liée à une session d'accès indique à l'utilisateur les services qu'il peut démarrer.

Scénario

1) L'application UAP liée à une session d'accès demande à l'agent fournisseur une liste des services auxquels l'utilisateur est abonné. L'agent fournisseur répercute cette demande vers l'agent utilisateur qui lui renvoie la liste. L'application UAP affiche cette liste à destination de l'utilisateur qui choisit le service devant être démarré[22]. L'application UAP demande ensuite à l'agent fournisseur de démarrer le service.

2) L'agent fournisseur démarre l'application UAP liée à cette session de service[23] et lui indique qu'elle doit démarrer le type de service "visioconférence 234".

3) L'application UAP liée à une session de service demande à l'agent fournisseur une nouvelle session de service avec le type de service "visioconférence 234" (l'application UAP peut transmettre elle-même à l'agent fournisseur les informations qui la concernent, incluant les modèles de session et les ensembles de fonctionnalités pris en charge, ainsi que les références de ses interfaces opérationnelles et de flux).

4) L'agent fournisseur demande à l'agent utilisateur (de l'utilisateur A) de démarrer une nouvelle session de service avec le type de service "visioconférence 234" (il peut également transmettre les informations concernant l'application UAP).

5) L'agent utilisateur peut effectuer certaines autres actions avant de passer à la suite. Il peut, par exemple, effectuer un contrôle de la demande de nouvelle session par rapport au profil d'abonnement de l'utilisateur[24], afin de vérifier que ce dernier est abonné à ce service et que ce service est utilisable avec la configuration de terminal de l'utilisateur. Il est également possible de prendre d'autres décisions. L'agent utilisateur peut activer une exception à l'intention de l'agent fournisseur s'il refuse de démarrer la session de service.

6) L'agent utilisateur reçoit la référence d'un atelier de service qui est en mesure de créer des composants de session de service pour le type de service "visioconférence 234"[25].

7) L'agent utilisateur demande à l'atelier de service la création d'une nouvelle session avec le type de service "visioconférence 234".

8) L'atelier de service crée et initialise un gestionnaire SSM et un gestionnaire USM[26].

9) L'atelier de service renvoie à l'agent utilisateur les références d'interface du gestionnaire USM et du gestionnaire SSM.

10) L'agent utilisateur renvoie à l'agent fournisseur les références d'interface du gestionnaire USM et du gestionnaire SSM.

11) L'agent fournisseur renvoie à l'application UAP liée à une session de service les références d'interface du gestionnaire USM et du gestionnaire SSM.

---

[22] Les interactions précédentes ne sont pas représentées dans la figure.

[23] Si l'application UAP liée à une session de service n'est pas disponible dans le domaine de l'utilisateur, l'agent fournisseur peut alors tenter de la décharger et de passer à la suite.

[24] Le profil d'abonnement de l'utilisateur peut définir des préférences et des contraintes pour l'invocation d'un service. Ces préférences ou contraintes peuvent être fonction de l'emplacement de l'utilisateur. Cette fonctionnalité fournit la prise en charge de la mobilité personnelle.

[25] L'agent utilisateur peut mettre en oeuvre un service de localisation ou un autre moyen pour trouver un atelier de service adéquat. L'agent utilisateur peut également fournir d'autres informations, telles que les informations de configuration de terminal, permettant de préciser le domaine de recherche de l'atelier de service. Les autres moyens englobent les informations d'abonnement qui peuvent éventuellement contenir une référence d'interface vers l'atelier de service devant être utilisé.

[26] L'atelier de service crée les objets informatiques qui constituent la session de service. Ces objets peuvent comprendre, entre autres, les gestionnaires USM et SSM.

12) L'application UAP liée à une session de service et le gestionnaire USM (et SSM) peuvent interagir par le biais d'interfaces propres au service ou d'interfaces définies par des modèles de session, y compris le modèle de session proposé. Il se peut que certaines interactions soient nécessaires entre ces composants avant que l'utilisateur puisse employer le service.

A cet instant, l'utilisateur A est le seul impliqué dans la session de service. Certains services peuvent être accessibles par un seul utilisateur ou être mis en œuvre par un utilisateur unique. Dans le cas de cet exemple qui traite d'un service de visioconférence, l'utilisateur A souhaite probablement inviter d'autres utilisateurs à se joindre à la session.

6.5.4 Invitation d'un utilisateur à se joindre à une session de service existante

Cet exemple traite le cas d'un utilisateur A qui invite un autre utilisateur (B) à se joindre à la session de service (voir Figure 6-6). L'exemple se termine au moment où l'invitation a été remise à l'utilisateur B. Le 6.5.5 traite l'exemple de l'entrée effective de l'utilisateur B dans la session.

Cet exemple fait l'hypothèse que l'utilisateur B invité est un utilisateur nommé, représenté par un agent utilisateur nommé au sein du fournisseur. Des utilisateurs anonymes représentés par un agent utilisateur anonyme ne peuvent pas être invités à se joindre à une session parce qu'il est impossible de localiser l'utilisateur concerné, du fait qu'un agent utilisateur anonyme ne publie pas l'identité de son utilisateur (et peut éventuellement ne pas en avoir connaissance).

Ce scénario prend en charge la mobilité de l'utilisateur en lui permettant d'être invité à se joindre à une session quel que soit son emplacement (ce qui ne signifie pas nécessairement qu'il sera capable de se joindre à la session).

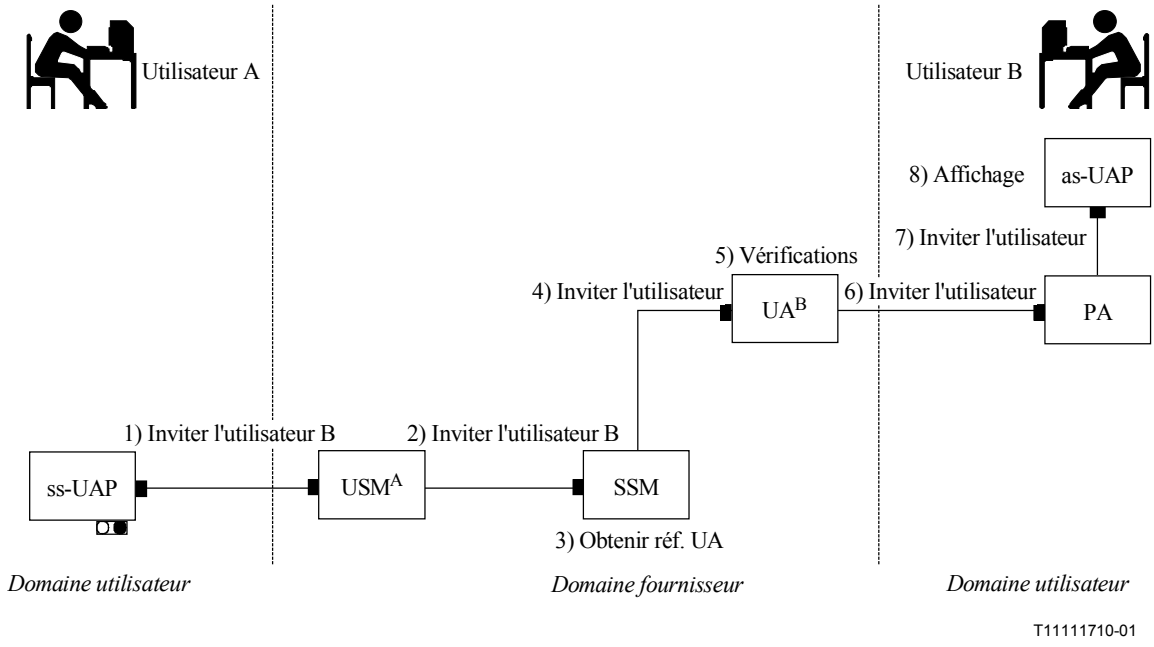


Figure 6-6 – Invitation d'un utilisateur à se joindre à une session de service existante

Conditions préalables

Une session d'accès existe entre l'agent fournisseur et un agent utilisateur de l'utilisateur qui émet l'invitation (utilisateur A). Il n'existe pas nécessairement une session d'accès entre un agent fournisseur et l'agent utilisateur de l'utilisateur qui reçoit l'invitation (utilisateur B), mais l'exemple fait l'hypothèse qu'il existe effectivement une session d'accès pour l'utilisateur B.

L'utilisateur A met en œuvre une application UAP liée à une session de service et dispose d'une session de service établie avec un gestionnaire USM et un gestionnaire SSM. L'utilisateur A souhaite inviter l'utilisateur B à se joindre à cette session de service. L'utilisateur A est "actif" dans la session de service, c'est-à-dire qu'il n'a pas suspendu sa participation.

Scénario

1) L'utilisateur A met en œuvre une application UAP pour inviter un autre utilisateur (invité) à se joindre à une session (l'utilisateur A fournit le nom de l'invité ou un nom d'alias d'utilisateur défini qui peut être résolu par l'agent utilisateur de l'invitant). L'application UAP demande à un gestionnaire USM d'inviter l'utilisateur B à se joindre à la session.
2) Le gestionnaire USM demande au gestionnaire SSM d'inviter un utilisateur à se joindre à la session (un gestionnaire USM et un gestionnaire SSM peuvent vérifier que l'utilisateur A est autorisé à inviter l'utilisateur B; ces vérifications ne sont pas définies dans le présent document).
3) Le gestionnaire SSM reçoit une référence pour une interface d'invitation de l'agent utilisateur de l'utilisateur B[27].
4) Le gestionnaire SSM envoie une invitation par le biais de l'interface d'invitation de l'agent utilisateur de l'utilisateur B.
5) L'agent utilisateur de l'invité peut effectuer certaines autres actions. Il peut par exemple vérifier le profil utilisateur par rapport à une politique d'invitation. Cette politique détermine alors les actions de l'agent utilisateur et les interactions avec d'autres objets. L'agent utilisateur peut activer une exception à l'intention du gestionnaire SSM s'il refuse de remettre l'invitation.
6) Une session d'accès existe, dans cet exemple, entre l'agent utilisateur de l'utilisateur B et l'agent fournisseur sur le terminal de l'utilisateur B. L'invitation est remise à l'agent fournisseur par le biais d'une interface d'invitation de l'agent fournisseur.
7) L'agent fournisseur émet l'invitation à destination de l'application UAP liée à une session d'accès.
8) L'application UAP affiche l'invitation à destination de l'utilisateur B.

L'invitation à se joindre à la session de service a été remise à l'agent utilisateur de l'utilisateur B et à l'agent fournisseur; elle est affichée par l'application UAP. L'invitation contient des informations suffisantes permettant à l'agent utilisateur de localiser la session de service et à l'utilisateur de se joindre à cette dernière (comme décrit au 6.5.5). Une partie seulement de ces informations sera retransmise vers l'agent fournisseur et l'application UAP.

Une session d'accès existe déjà, dans l'exemple ci-dessus, entre l'agent fournisseur de l'utilisateur B et l'agent utilisateur. Les variantes suivantes sont possibles pour la suite du scénario si l'utilisateur B n'est pas engagé actuellement dans une session d'accès avec l'agent utilisateur:

- l'agent utilisateur stocke l'invitation dans l'attente de l'établissement d'une session d'accès par l'utilisateur invité, après quoi l'invitation est remise comme ci-dessus;
- l'agent utilisateur remet l'invitation à un terminal immatriculé (qui a été choisi par l'utilisateur pour recevoir des invitations en l'absence d'une session d'accès)[28];
- l'agent utilisateur renvoie au gestionnaire SSM l'adresse d'un terminal immatriculé;

---

[27] L'agent utilisateur peut mettre en oeuvre un service de localisation ou d'autres moyens pour déterminer l'emplacement de l'agent utilisateur de l'utilisateur B.

[28] Ce cas est nécessaire pour la prise en charge de la mobilité personnelle.

- l'agent utilisateur transmet l'invitation à un autre agent utilisateur (qui a été choisi par l'utilisateur pour recevoir des invitations en l'absence d'une session d'accès; cet agent utilisateur peut se trouver dans un autre domaine du fournisseur);
- l'agent utilisateur renvoie l'adresse d'un autre agent utilisateur;
- l'agent utilisateur démarre une session de service d'un type spécifié (l'invitation peut être transmise à la session de service au moment de sa configuration ou à un instant ultérieur).

6.5.5 Entrée dans une session de service existante

Cet exemple traite le cas d'un utilisateur B qui se joint à une session existante dans laquelle il a été invité (voir Figure 6-7).

On fait l'hypothèse que l'utilisateur B se trouve dans une session d'accès avec le fournisseur et possède un abonnement valide pour le service (le type de service est "visioconférence 234"). On fait également l'hypothèse que l'application UAP liée à une session de service est présente sur le terminal de l'utilisateur B.

L'utilisateur B peut se joindre à cette session à partir de tout terminal pour lequel il a établi une session d'accès. Une fois que la session d'accès est établie, l'agent fournisseur demande une liste des invitations reçues par l'agent utilisateur. L'agent fournisseur peut ensuite demander à se joindre à l'une quelconque des sessions correspondantes. Nous supposerons toutefois dans cet exemple que les invitations ont été remises à l'agent fournisseur de l'utilisateur B et à l'application UAP liée à une session d'accès, comme décrit au 6.5.4.

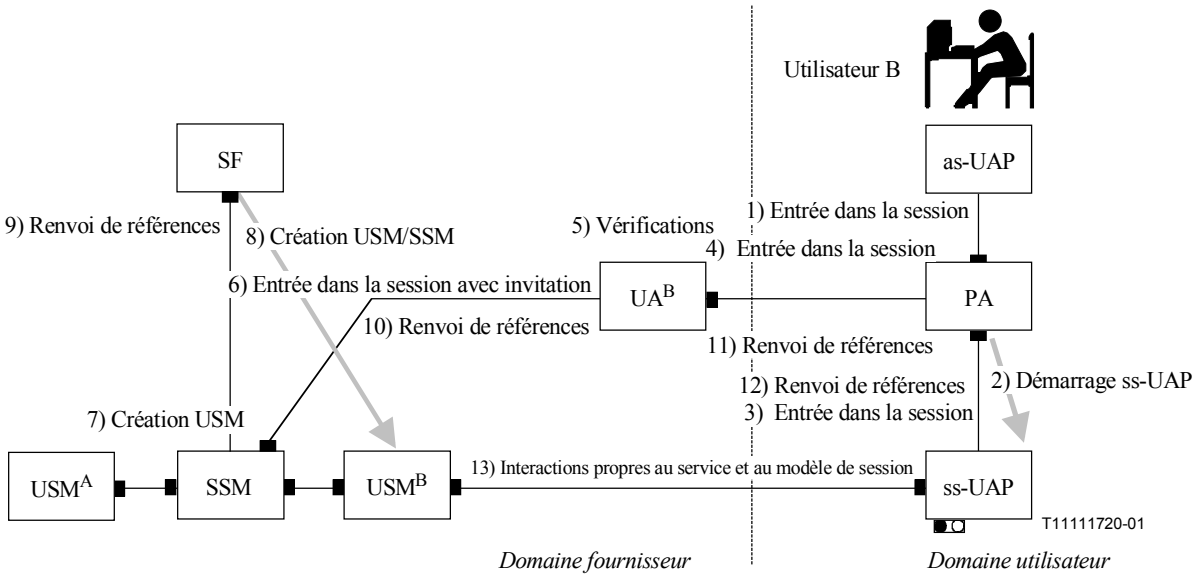


Figure 6-7 – Entrée dans une session de service existante

Conditions préalables

Une session d'accès existe entre l'agent fournisseur (utilisateur B) et l'agent utilisateur (au sein du domaine fournisseur). L'agent utilisateur de l'utilisateur B et l'agent fournisseur ont reçu une invitation à se joindre à une session de service et une application UAP liée à une session d'accès indique à l'utilisateur l'invitation qui a été reçue.

Scénario

1) L'application UAP liée à une session d'accès affiche une liste d'invitations à se joindre à des sessions de service. L'utilisateur choisit l'une de ces invitations. L'application UAP demande à l'agent fournisseur de se joindre à la session de service en lui indiquant un identificateur d'invitation.
2) L'agent fournisseur démarre une application UAP liée à une session de service[29] qui est associée à ce type de session de service et lui indique l'identificateur d'invitation qu'elle doit utiliser pour sa demande.
3) L'application UAP liée à une session de service demande à se joindre à la session de service de l'agent fournisseur en indiquant l'identificateur d'invitation (l'application UAP peut transférer vers l'agent fournisseur des informations qui la concernent, incluant les modèles de session pris en charge et des références vers ses interfaces opérationnelles et de flux).
4) L'agent fournisseur demande à se joindre à la session de service de l'agent utilisateur (de l'utilisateur B) en indiquant l'identificateur d'invitation (il peut également transférer des indications concernant l'application UAP).
5) L'agent utilisateur peut effectuer certaines autres actions avant de passer à la suite. Il peut, par exemple, contrôler l'identificateur d'invitation par rapport aux invitations existantes de cet utilisateur et à son profil d'abonnement[30] pour vérifier qu'il est abonné à ce service et peut l'utiliser avec la configuration de terminal de l'utilisateur, etc. L'agent utilisateur peut également refuser de se joindre à la session.
6) L'agent utilisateur reçoit la référence du gestionnaire SSM[31] et demande à se joindre à la session (il peut transférer certaines informations figurant dans l'invitation pour confirmer que le gestionnaire SSM a invité cet utilisateur à se joindre à cette session).
7) Le gestionnaire SSM demande à son atelier de service de créer un gestionnaire USM pour l'utilisateur B.
8) L'atelier de service crée et initialise un gestionnaire USM.
9) L'atelier de service renvoie au gestionnaire SSM les références d'interface d'un gestionnaire USM.
10) Le gestionnaire SSM renvoie à l'agent utilisateur les références d'un gestionnaire USM et ses références propres.
11) L'agent utilisateur renvoie à l'agent fournisseur les références des gestionnaires USM et SSM.
12) L'agent fournisseur renvoie à l'application UAP liée à une session de service les références des gestionnaires USM et SSM.
13) L'application UAP liée à une session de service et le gestionnaire USM (et SSM) peuvent interagir par le biais d'interfaces propres au service ou définies par les modèles de session, y compris le modèle de session proposé. Il se peut que certaines interactions soient nécessaires entre ces composants avant que l'utilisateur puisse employer le service.

---

[29] Si l'application UAP liée à une session de service n'est pas disponible dans le domaine de l'utilisateur, l'agent fournisseur peut alors tenter de l'obtenir par déchargement.

[30] Le profil d'abonnement de l'utilisateur peut définir des préférences et des contraintes pour l'invocation d'un service. Ces préférences ou contraintes peuvent être fonction de l'emplacement de l'utilisateur. Cette fonctionnalité fournit la prise en charge de la mobilité personnelle.

[31] L'invitation peut contenir une référence vers une interface du gestionnaire SSM devant être utilisée pour se joindre à la session. L'agent utilisateur peut également mettre en oeuvre, pour trouver le gestionnaire SSM, les informations fournies par l'invitation, ainsi qu'un service de localisation ou d'autres moyens. L'agent utilisateur peut également fournir au gestionnaire SSM d'autres informations telles que des informations de configuration de terminal ou d'application.

L'utilisateur A et l'utilisateur B sont impliqués dans la session de service à partir de cet instant. Dans le cas de cet exemple qui traite d'un service de visioconférence, l'un ou l'autre des utilisateurs peut inviter d'autres utilisateurs à se joindre à la session. Une politique propre au service peut éventuellement déterminer si un utilisateur donné est autorisé à en inviter d'autres.

6.5.6 Demande et établissement d'un attachement de flux

Cet exemple traite le cas de l'établissement d'un attachement de flux entre des applications UAP de consommateurs situés dans les domaines des consommateurs 1 et 2 (voir Figure 6-8). Le consommateur 3 demande l'établissement d'un attachement de flux auquel participeront le consommateur 1 et le consommateur 2, mais sans y participer lui-même[32].

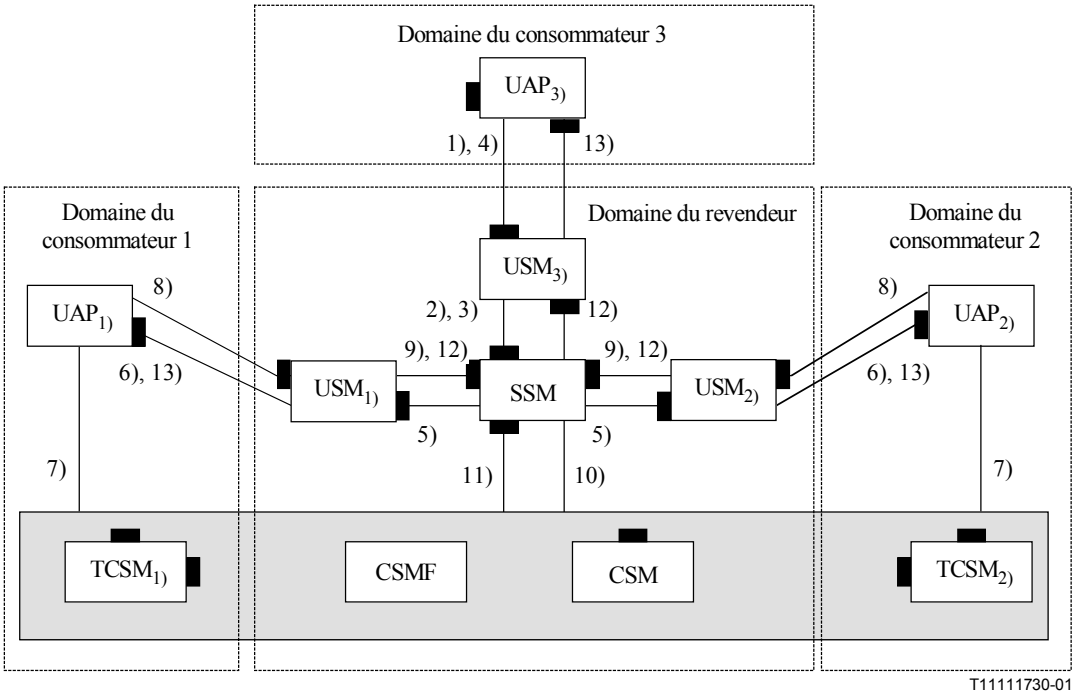


Figure 6-8 – Demande et établissement d'un attachement de flux [la session de service possède trois participants et une ressource (session de communication) impliquant deux d'entre eux]

Conditions préalables

Les trois consommateurs ont déjà été invités dans la session de service et sont des participants de cette session de service.

Conditions postérieures

Un attachement de flux est établi entre les participants 1 et 2; le participant 3 possède une référence de l'attachement de flux en vue de sa commande ultérieure.

---

[32] Cet exemple illustre la distinction entre une session de service et une session de communication. D'autres exemples sont possibles, tels que celui d'un consommateur 1 qui agit comme l'initiateur de la session de service et de la session de communication (cas proche de la téléphonie classique). La distinction entre accès et utilisation et entre session de service et session de communication conserve sa validité et fournit une valeur ajoutée (par exemple, pour la prise en charge des caractéristiques de mobilité).

Scénario

Le scénario représente la réussite d'un établissement, mais il est toutefois clair que des décisions différentes peuvent être prises au niveau de certains points.

1) L'application UAP3 demande l'établissement d'un attachement de flux entre les participants 1 et 2. Le moniteur USM3 peut vérifier, de manière optionnelle, que le consommateur 3 possède l'autorisation d'établir l'attachement de flux concerné.

2) Le gestionnaire USM3 retransmet la demande vers le gestionnaire SSM de la session de service.

   Le moniteur SSM peut vérifier, de manière optionnelle, la permission d'établissement de cet attachement de flux; il peut, si nécessaire, négocier cette permission avec les autres membres de cette session (ce cas implique une procédure de vote, non représentée).

3) Le gestionnaire SSM renvoie au gestionnaire USM3, si la permission est obtenue, un identificateur d'attachement de flux ainsi qu'un identificateur de demande à des fins de confirmation ultérieure.

4) Le gestionnaire USM renvoie à l'application UAP3 un identificateur d'attachement de flux ainsi qu'un identificateur de demande à des fins de confirmation ultérieure.

   (Les étapes suivantes peuvent être réalisées en parallèle pour les participant 1 et 2; elles sont décrites en utilisant le suffixe "*i*".)

5) Le gestionnaire SSM demande au gestionnaire USMi de se joindre à l'attachement de flux.

   Le gestionnaire USMi peut prendre certaines décisions optionnelles, concernant par exemple l'absence de participation pour le compte du consommateur *i*.

6) Le gestionnaire USMi retransmet la demande vers l'application UAPi.

7) L'application UAPi démarre un scénario d'établissement d'application pour obtenir la mémoire tampon NFEP liée à l'utilisateur de l'interface de flux pour le consommateur *i* pour ce service. Ceci peut déjà avoir été fait[33], ou doit être réalisé à cet instant pour permettre au consommateur de participer à cet attachement de flux.

8) L'application UAPi accepte la demande et renvoie au moniteur USMi une indication de cette acceptation, ainsi qu'une description des conditions de participation du consommateur *i* dans l'attachement de flux et un descripteur d'interface de flux.

9) Le gestionnaire USMi retransmet ces informations au gestionnaire SSM.

   Ce dernier peut, en fonction des réponses des participants (et de la logique propre à ce service), choisir d'abandonner l'établissement de l'attachement de flux ou, comme indiqué, effectuer une demande de communication pour l'attachement de flux.

10) Le gestionnaire SSM demande l'établissement d'une ressource (session de communication), si elle n'existe pas encore, puis demande l'établissement des écoulements de flux associés à l'attachement de flux.

11) Renvoi des notifications finales au gestionnaire SSM.

---

[33] Plusieurs cas se présentent selon le type de service et les capacités du terminal. Si l'application est la seule qui utilise des flux, la partie nodale (terminale) de l'attachement de flux peut alors être câblée. Il se peut également que, lorsque l'invitation est reçue, le consommateur 1 sache déjà (ou présume) qu'il sera sollicité pour participer à un attachement de flux et commencera la préparation des actions internes du terminal nécessaires pour disposer d'une interface de flux (par exemple, demander à une autre application de libérer une interface de flux ou de supprimer certaines applications afin d'améliorer les performances). Cette fonctionnalité constitue une spécialisation du comportement de l'application UAP.

12) Renvoi des notifications finales aux gestionnaires USM pour les consommateurs 1, 2 et 3 (ou à certains d'entre eux, compte tenu des réponses fournies par chaque consommateur *i* lors de l'étape 9).

13) Renvoi des notifications finales aux consommateurs 1, 2 et 3 (ou à certains d'entre eux, compte tenu des réponses fournies par l'application UAP lors de l'étape 9 ou du comportement des gestionnaires USMi[34]).

7 Aperçu général de la spécification de point de référence Ret

Ce paragraphe constitue une introduction de la spécification du point de référence Ret; il fournit une indication des fonctionnalités globales et du domaine d'application ainsi qu'une définition sommaire de ce point de référence.

Le paragraphe 8 décrit la manière dont un consommateur accède à un revendeur pour l'utilisation des services dont l'accès est fourni par ce revendeur. Il traite de l'établissement et de l'utilisation d'une association sécurisée entre les domaines sous la forme d'une session d'accès (définie et décrite de manière plus détaillée au paragraphe 8). Il traite également de la commande des services, de la commande des sessions de service et de la gestion d'abonnement au sein de la session d'accès. Il présente un ensemble d'interfaces opérationnelles offertes par les rôles commerciaux de consommateur et de revendeur. La définition de ces interfaces est faite en langage usuel et sous la forme de diagrammes, ainsi que d'une manière semi-formalisée sous la forme de spécifications en langage IDL; les comportements sont décrits en langage usuel. Une interface dédiée aux abonnements est également décrite de manière détaillée. Le paragraphe 8 contient les spécifications détaillées des interfaces IDL.

Le paragraphe 9 contient les spécifications complètes des interfaces IDL du point Ret-RP, y compris pour les interfaces de la partie relative à l'abonnement.

Les spécifications sont fournies de manière non formalisée en langage usuel, sous forme de diagrammes et de manière semi-formalisée par des spécifications en langage de définition d'interface (IDL).

Le présent supplément a pour objet de fournir des spécifications prêtes à l'emploi pour une implémentation par des fournisseurs multiples avec des interfaces informatiques compatibles entre le domaine revendeur et le domaine consommateur.

7.1 Fonctionnalités générales et domaine d'application des points de référence

Un point de référence de l'environnement SPFEE sert à la description des interactions entre acteurs (par le biais de leurs interfaces mutuelles). Le présent supplément traite de la fonctionnalité suivante:

– points de référence de revendeur (Ret-RP, *retailer reference point*): entre le consommateur et le revendeur.

Il convient de noter que la présente spécification généralise le rôle d'utilisateur final sous la forme d'un rôle de consommateur. Le rôle de consommateur modélise deux acteurs, à savoir l'abonné et l'utilisateur final. L'abonné correspond à l'entité qui est engagée dans une relation commerciale avec le revendeur, alors que l'utilisateur final correspond à la personne qui utilise effectivement les capacités du fournisseur de service par l'intermédiaire du revendeur.

Le point Ret-RP prend en charge l'accès d'un consommateur à un revendeur pour l'utilisation de services que ce dernier met à disposition pour le compte d'un ou de plusieurs fournisseurs de service.

---

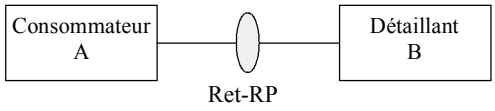
[34] Il est possible que le moniteur USM reçoive la notification mais ne la transmette pas vers l'application UAP; ce cas est comparable à l'étape 5, dans laquelle le moniteur USM prend des décisions (par exemple, de filtrage) pour le compte de l'utilisateur ou de l'application UAP.

Il gère l'établissement et l'utilisation d'une association sécurisée entre les domaines, appelée "session d'accès". Il fournit, au sein de la session d'accès, la gestion du cycle de vie pour l'utilisation des services du fournisseur de service. Il correspond aux fonctionnalités, interfaces et objets liés à la session d'accès. Il définit des interfaces permettant de prendre en charge l'utilisation des fonctionnalités suivantes:

– initiation du dialogue entre les domaines consommateur (abonné et utilisateur final) et revendeur;
– identification mutuelle des domaines (chacun des domaines peut rester anonyme, selon l'interaction demandée);
– établissement d'une association sécurisée entre les domaines (session d'accès);
– établissement du contexte par défaut pour la commande et la gestion des fonctionnalités d'utilisation;
– détection des offres de service[35];
– liste des sessions d'accès, des sessions de service et des services qui ont fait l'objet d'un abonnement;
– initiation des procédures d'utilisation entre les domaines de l'utilisateur final et du fournisseur de service;
– commande et gestion de sessions (par exemple, fin, suspension, reprise, entrée, notification de modifications, etc.).

Le présent supplément utilise les principes suivants:

– la mobilité personnelle et de session fournit la description des modalités de transfert et de gestion des environnements personnels au niveau de points d'accès de l'utilisateur final au sein d'une session;
– la gestion fournit les procédures permettant de traiter les informations administratives (par exemple, les abonnés) et les fonctionnalités FCAPS (par exemple, la gestion des fautes pour un service).



Utilisation du point Ret-RP en fonction des rôles commerciaux TINA

T11111740-01

7.1.1 Cycle de vie du rôle commercial du point Ret-RP

Le point Ret-RP prend en charge la totalité du cycle de vie de la relation entre un consommateur et un revendeur, qui est définie par l'expression "cycle de vie de l'abonné de l'utilisateur final".

Le cycle de vie de l'abonné décrit le processus permettant à un abonné d'établir une relation avec un revendeur, puis de modifier et de mettre fin à la relation. La relation englobe l'abonnement, la personnalisation et l'association entre l'abonné et l'utilisateur final.

Le cycle de vie de l'utilisateur décrit le processus par lequel des utilisateurs finaux peuvent accéder à des services et les utiliser. Ces fonctionnalités englobent l'établissement du système de l'utilisateur final, le contact avec le revendeur et la personnalisation du service.

---

[35] Il peut s'agir de services primaires [par exemple, de vidéo à la demande (VoD, *video on demand*)], de services auxiliaires d'un service primaire (par exemple, gestion de configuration VoD) ou de services administratifs (par exemple, gestion d'abonné VoD).

7.2 Principales hypothèses

Le présent supplément fait les deux hypothèses principales suivantes:

– environnement DPE permanent et capable d'interfonctionnement pour la fourniture des services de sécurité;

– existence d'un cadre général d'adressage et de résolution de nom.

7.3 Définition du point de référence Ret

La définition du point de référence Ret fournit une spécification semi-formalisée de la relation entre les rôles commerciaux du consommateur et du revendeur. Conformément à la référence [6] "Modèle commercial et points de référence SPFEE", le point Ret-RP se subdivise en une partie "accès" et une partie "utilisation". La partie "accès" du point Ret-RP décrit la manière dont un rôle commercial consommateur accède à un rôle commercial revendeur pour l'utilisation des services d'un fournisseur de service; la partie "utilisation" est en dehors du domaine d'application du présent document. Ces deux parties peuvent être utilisées séparément, du fait qu'elles sont traitées de manière indépendante dans les spécifications SPFEE. La suite du présent document décrit uniquement la partie "accès".

7.3.1 Rôles commerciaux et rôles de session

Comme indiqué dans la référence [6], un rôle commercial peut jouer divers rôles de session. Deux rôles de session de base ont été définis, à savoir les rôles "utilisateur" et "fournisseur". Les interactions liées à l'accès utilisent des spécialisations de ces rôles qui deviennent ainsi un utilisateur d'accès et un fournisseur d'accès. Les rôles de session et les conventions de dénomination des domaines commerciaux correspondent aux dénominations de la structure de module des spécifications IDL (paragraphe 9).

Bien que les spécifications du point Ret-RP fassent référence aux rôles commerciaux (conformément à la référence [6]), leur application peut s'étendre à des relations impliquant des rôles de session identiques, quels que soient les rôles commerciaux concernés. La partie "accès" des spécifications du point Ret-RP peut s'appliquer, par exemple, lorsqu'il est possible de définir un utilisateur d'accès et un fournisseur d'accès. Les procédés permettant d'étendre les spécifications de point Ret-RP à des contextes autres que la relation entre un consommateur et un revendeur (par exemple pour une fédération entre revendeurs) sont en dehors du domaine d'application du présent supplément.

7.3.2 Conformité aux spécifications de point de référence Ret

Le présent supplément fournit des directives pour la compréhension de la conformité aux spécifications de point Ret-RP. Cette conformité ne nécessite pas la prise en charge de toutes les fonctionnalités, mais implique la prise en charge de toutes les fonctionnalités obligatoires[36] conformément aux spécifications SPFEE, ainsi que la conformité à ces spécifications pour la prise en charge de fonctionnalités optionnelles.

On distingue des interfaces et des opérations obligatoires et optionnelles au niveau du point Ret-RP.

La conformité du point Ret-RP est déclarée de manière distincte pour le côté consommateur et le côté revendeur.

Il s'ensuit que le niveau minimal de conformité du point Ret-RP pour un système SPFEE prend en charge au minimum les interfaces et opérations obligatoires au niveau de l'un des deux côtés (consommateur ou revendeur).

Pour que deux systèmes SPFEE puissent interagir au niveau du point Ret-RP, il est nécessaire que l'un des systèmes soit conforme du côté consommateur et l'autre du côté revendeur.

---

[36] Une fonctionnalité correspond ici à une interface ou à une opération.

8 Spécification du point Ret-RP

Le point Ret-RP fournit les capacités suivantes:

– initiation du dialogue entre les domaines consommateur et revendeur;

– identification mutuelle des domaines (chacun des domaines peut rester anonyme en fonction de l'interaction demandée);

– établissement d'une association sécurisée (session d'accès) entre les domaines;

– établissement du contexte par défaut pour la commande et la gestion des fonctionnalités d'utilisation (sessions de service);

– détection des offres de service[37];

– liste des sessions d'accès, des sessions de service et des services qui ont fait l'objet d'un abonnement;

– initiation des procédures d'utilisation entre les domaines de l'utilisateur final et du fournisseur de service;

– commande et gestion de sessions (par exemple, fin, suspension, reprise, entrée, notification de modifications, etc.).

Il convient de noter que le point Ret-RP concerne deux types de fonctionnalités d'accès:

– fonctionnalités dédiées à la session d'accès entre le consommateur et le revendeur;

– fonctionnalités liées à l'accès aux services, pour lesquelles le revendeur doit invoquer un ou plusieurs fournisseurs de service qui prennent effectivement en charge les services (le revendeur joue un rôle de revendeur dédié uniquement à la fonctionnalité d'accès).

  L'une des fonctionnalités du revendeur lui permet d'invoquer, si nécessaire, plusieurs fournisseurs de service pour répondre à la demande du consommateur. L'opération est totalement transparente pour ce dernier. Le revendeur effectue une ou plusieurs invocations pour un ou plusieurs fournisseurs de service et renvoie un résultat global. L'opération est également transparente vis-à-vis des fournisseurs de service qui ne doivent pas avoir connaissance du fait que le revendeur effectue éventuellement des invocations similaires pour d'autres fournisseurs de service. Il convient de noter que les spécifications pour les points Ret-RP et Ret-SP-RP présentent une souplesse et un caractère général permettant le multiplexage du rôle de revendeur. Il en résulte qu'une spécification supplémentaire à ce sujet est en dehors du domaine d'application de la spécification des points de référence Ret et Ret-SP.

Les fonctions dédiées à l'accès aux services sont les suivantes:

– détection des offres de service[38];

– liste des sessions d'accès, des sessions de service et des services qui ont fait l'objet d'un abonnement;

– initiation des procédures d'utilisation entre les domaines (démarrage d'une session de service);

– commande et gestion de sessions (par exemple, fin, suspension, reprise, entrée, notification de modifications, etc.).

---

[37] Il peut s'agir de services primaires [par exemple, de vidéo à la demande (VoD)], de services auxiliaires d'un service primaire (par exemple, gestion de configuration VoD) ou de services administratifs (par exemple, gestion d'abonné VoD).

[38] Il peut s'agir de services primaires [par exemple, de vidéo à la demande (VoD)], de services auxiliaires d'un service primaire (par exemple, gestion de configuration VoD) ou de services administratifs (par exemple, gestion d'abonné VoD).

La spécification du point Ret-RP traite en grande partie de l'établissement et de l'utilisation d'une association sécurisée entre les domaines, c'est-à-dire une session d'accès.

Les sous-paragraphes qui suivent décrivent la session d'accès du point Ret-RP. Une grande partie des interfaces et des opérations définies au niveau du point Ret-RP s'appliqueront pour les parties d'accès d'autres points de référence entre domaines (tels que les points Ret-SP et revendeur-revendeur). Un ensemble d'interfaces réutilisables a été défini pour d'autres points de référence. Ces interfaces sont désignées par les préfixes i_User et i_Provider correspondant respectivement aux types d'interfaces des rôles d'utilisateur d'accès et de fournisseur d'accès.

Les interfaces au niveau du point Ret-RP sont caractérisées par les préfixes i_Consumer et i_Retailer correspondant respectivement aux domaines administratifs commerciaux de consommateur et de revendeur pour le modèle commercial SPFEE [6]. Dans le cas du point Ret-RP, ces domaines jouent les rôles d'utilisateur et de fournisseur d'accès. Toutes les interfaces i_Consumer et i_Retailer sont héritées des interfaces i_User et i_Provider correspondantes. Toutes les spécialisations du point Ret-RP sont définies pour les interfaces i_Consumer ou i_Retailer. Aucune spécialisation n'est toutefois définie à l'heure actuelle.

En résumé, les interfaces du point Ret-RP sont héritées des interfaces génériques fournies par l'utilisateur et qui peuvent être réutilisées pour un grand nombre d'autres points de référence.

NOTE – Le corps du présent supplément décrit uniquement les interfaces et leurs opérations. Le paragraphe 9 fournit une liste complète de descriptions en langage IDL, ainsi que le regroupement des interfaces sous la forme de modules.

La suite de la description de la session d'accès de point Ret-RP est structurée comme suit:

Le sous-paragraphe 8.1 "Présentation générale des interfaces d'accès au niveau du point Ret-RP" décrit les interfaces d'accès au point Ret-RP et fournit une brève explication pour chaque opération. Il traite uniquement des interfaces qui sont exportées par le point Ret-RP.

Le sous-paragraphe 8.2 "Interfaces utilisateur-fournisseur" décrit les interfaces génériques utilisateur-fournisseur qui ne sont pas exportées par le point Ret-RP. Il indique la hiérarchie d'héritage pour les interfaces exportées par le point Ret-RP. Il décrit également des interfaces génériques pouvant être réutilisées pour d'autres points de référence entre domaines.

Le sous-paragraphe 8.3 "Vue des informations d'accès" fournit une vue informationnelle du point Ret-RP qui décrit les types d'informations transférées au niveau de ce point Ret-RP.

Les sous-paragraphes 8.5 "Définitions d'interface d'accès: interfaces de domaine consommateur" et 8.6 "Définitions d'interface d'accès: interfaces de domaine revendeur" décrivent le détail des opérations des interfaces au niveau du point Ret-RP prises en charge par les domaines consommateur et revendeur, y compris les paramètres et l'aspect dynamique.

Le paragraphe 9 fournit les définitions IDL de chacune des interfaces.

8.1 Présentation générale des interfaces d'accès au niveau du point Ret-RP

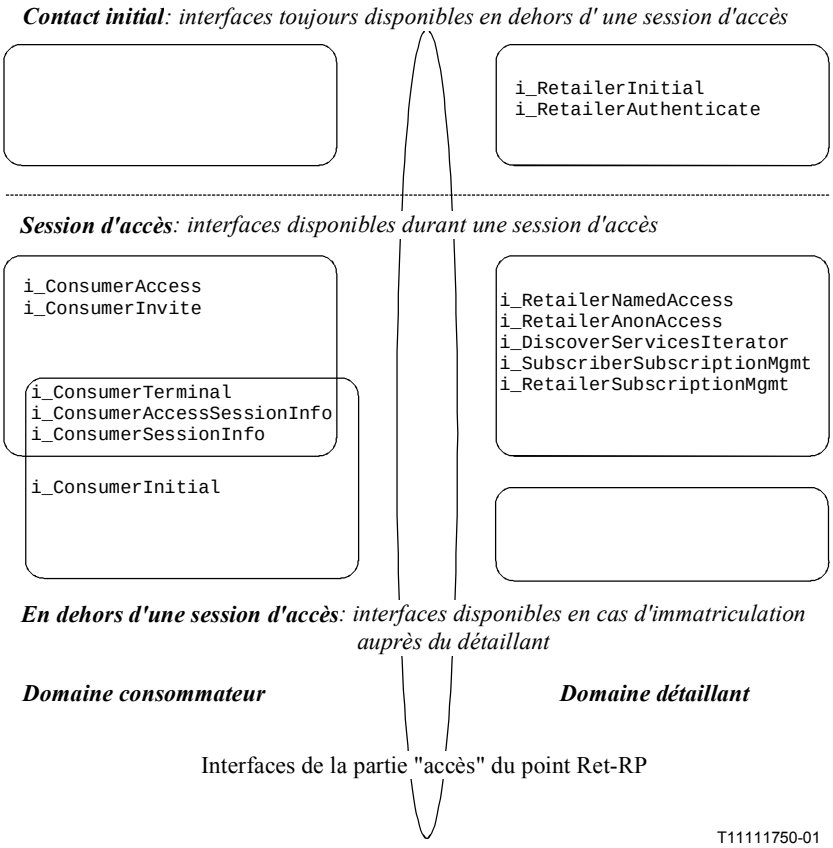
La partie "accès" du point Ret-RP est définie par un ensemble d'interfaces fournies au niveau du point de référence. Toutes les interfaces de la partie "accès" sont classées selon le côté consommateur ou revendeur du point de référence qui fournit l'interface.

Les interfaces sont également classées en fonction de leur modalités d'accès: durant une session d'accès, de manière inconditionnelle en dehors d'une session d'accès ou en dehors d'une session d'accès moyennant une immatriculation préalable.

L'immatriculation d'interfaces peut être faite par le consommateur dans le domaine revendeur durant une session d'accès. La durée de vie de l'immatriculation dépend de l'immatriculation effectuée par le consommateur, à savoir pour la durée de la session d'accès ou de manière permanente.

Le diagramme qui suit indique les noms des interfaces définies dans la partie "accès" selon les classements précédents.

Les interfaces "toujours disponibles en dehors d'une session d'accès" sont prises en charge par le revendeur afin de permettre à un consommateur de demander l'établissement d'une session d'accès. Elles constituent le point de contact initial pour le consommateur et lui permettent de s'authentifier et d'authentifier le revendeur, d'établir la session d'accès et d'obtenir la référence d'une interface i_RetailerNamedAccess ou i_RetailerAnonAccess.



Les interfaces "disponibles durant une session d'accès" permettent au consommateur et au revendeur d'interagir durant une session d'accès. Les interfaces du revendeur permettent au consommateur de détecter des services, d'initier l'utilisation de ces services, de commander et de gérer ces services (par exemple fin, suspension, reprise, etc.) ainsi que d'enregistrer le contexte du consommateur et les interfaces vers le revendeur. Les interfaces prises en charge par le consommateur permettent au revendeur de détecter les interfaces et la configuration de terminal du consommateur, de notifier des modifications dans les sessions d'accès et de service et d'émettre des invitations à se joindre à des sessions de service. Ces capacités sont disponibles uniquement durant la session d'accès.

Les interfaces "disponibles en cas d'immatriculation auprès du détaillant" sont prises en charge par le consommateur. Elle doivent faire l'objet d'une immatriculation auprès du revendeur de manière à être disponibles en dehors de la session d'accès. Lorsque l'interface adéquate a fait l'objet d'une immatriculation, le revendeur est en mesure d'effectuer toutes les opérations "disponibles durant une session d'accès" et peut également demander au consommateur d'initier une session d'accès.

Les sous-paragraphes qui suivent donnent une description globale des interfaces et de leurs opérations. Les sous-paragraphes 8.5 et 8.6 donnent les détails des opérations, des listes de paramètres et des caractéristiques dynamiques.

Les interfaces dédiées à l'abonnement (i_SubscriberSubscriptionMgmt et i_RetailerSubscriptionMgmt) sont décrites au 8.7 "Gestion de l'abonnement".

8.1.1 Exemple de scénario de partie "accès" du point Ret-RP

Le préssent sous-paragraphe fournit un exemple d'utilisation des interfaces d'accès au niveau du point Ret-RP. Il décrit un consommateur qui utilise des interfaces de revendeur pour établir une session d'accès, utiliser des fonctionnalités de revendeur et s'immatriculer pour la réception d'invitations en dehors d'une session d'accès.

1) Un domaine consommateur contacte un revendeur en obtenant une référence d'une interface i_RetailerInitial[39].

2) Le domaine consommateur invoque l'opération requestNamedAccess() sur l'interface i_RetailerInitial lorsqu'il souhaite établir une session d'accès avec le revendeur en tant qu'utilisateur nommé (le consommateur peut utiliser l'opération requestAnonymousAccess() sur cette interface s'il souhaite rester anonyme).

   2a) Si les services de sécurité CORBA ont été utilisés, les justificatifs des deux domaines et d'autres informations d'authentification ont alors été échangés; le consommateur et le revendeur se sont authentifiés mutuellement. L'invocation de l'opération requestNamedAccess() renvoie la référence d'une interface i_RetailerNamedAccess (une session d'accès a été établie entre les domaines consommateur et revendeur).

   2b) Si les services de sécurité CORBA n'ont pas été utilisés, l'invocation de l'opération requestNamedAccess() échoue alors et l'exception e_AuthenticationError est activée. Cette exception contient la référence d'une interface i_RetailerAuthenticate pouvant être utilisée par le consommateur pour s'authentifier. Le consommateur invoque ensuite l'opération requestNamedAccess() sur l'interface i_RetailerInitial pour obtenir une référence vers l'interface i_RetailerNamedAccess.

3) Une session d'accès a été établie à cet instant et le domaine consommateur possède une référence vers l'interface i_RetailerNamedAccess.

4) Le domaine consommateur informe le domaine revendeur au sujet de ses interfaces et de la configuration de terminal par le biais de l'opération **setUserCtxt**() sur l'interface i_RetailerNamedAccess. Le revendeur obtient les références des interfaces i_ConsumerAccess, i_ConsumerInvite, i_ConsumerTerminal et i_ConsumerSessionInfo utilisables durant cette session d'accès.

5) Le consommateur peut alors invoquer les opérations appropriées sur l'interface i_RetailerNamedAccess aux fins suivantes:

   – détecter les services offerts par le revendeur (discoverServices());
   – s'abonner à ces services (en démarrant un service d'abonnement);
   – établir la liste des sessions d'accès et des sessions de service qui sont actuellement activées (listAccessSessions(), listServiceSessions());
   – démarrer une nouvelle session de service (startService());
   – suspendre, reprendre, se joindre à et mettre fin à des sessions existantes (suspendSession(), resumeSession(), endSession());
   – obtenir des références vers des interfaces propres au revendeur (getInterfaces());
   – immatriculer des interfaces pour une utilisation en dehors d'une session d'accès (registerInterfaceOutsideAccessSession());
   – et davantage ...

[39] La manière dont le consommateur obtient cette interface n'est pas imposée par le point Ret-RP.

6) Le revendeur peut:

- obtenir des références vers des interfaces propres au revendeur (par le biais de l'interface i_ConsumerAccess);
- inviter le consommateur à se joindre à une session (par le biais de l'interface i_ConsumerInvite);
- détecter la configuration de terminal (par le biais de l'interface i_ConsumerTerminal);
- informer le consommateur au sujet de ses accès et sessions de service (par le biais des interfaces i_UserAccessSessionInfo et i_UserSessionInfo).

7) Le consommateur immatricule l'interface i_ConsumerInitial pour une utilisation en dehors d'une session d'accès (par le biais de l'opération registerInterfaceOutsideAccessSession() sur l'interface i_RetailerNamedAccess). Il met ensuite fin à la session d'accès, après quoi il ne peut plus effectuer de demande pour le revendeur.

8) Le revendeur conserve la capacité d'inviter le consommateur à se joindre à une session de service par le biais de l'opération inviteUserOutsideAccessSession() sur l'interface i_ConsumerInitial.

9) Si le consommateur souhaite se joindre à la session dans laquelle il a été invité, il devra alors établir une nouvelle session d'accès (comme dans l'étape 1).

8.1.2 Interfaces toujours disponibles en dehors d'une session d'accès

Seules les interfaces revendeur sont toujours disponibles en dehors d'une session d'accès.

Les interfaces suivantes fournies par le revendeur permettent au consommateur et/ou au revendeur de s'authentifier et d'établir une session d'accès.

- i_RetailerInitial – Cette interface constitue le point de contact initial du consommateur pour le revendeur. Elle peut être utilisée pour demander l'établissement d'une session d'accès. La session d'accès fournit au consommateur un accès aux services auxquels il est abonné, etc. par le biais d'une interface i_RetailerNamedAccess, ou i_RetailerAnonAccess si le consommateur est authentifié respectivement en tant qu'utilisateur nommé ou anonyme. Si le consommateur n'est pas authentifié, il renvoie alors la référence d'une interface i_RetailerAuthenticate permettant d'effectuer cette authentification.
- i_RetailerAuthenticate – Cette interface est utilisée par le consommateur pour s'authentifier et authentifier le revendeur ainsi que pour transmettre un justificatif pouvant être utilisé pour l'établissement de la session d'accès.

8.1.2.1 Interface i_RetailerInitial

L'interface i_RetailerInitial permet au consommateur de demander l'établissement d'une session d'accès.

- L'opération requestNamedAccess() permet au consommateur de s'identifier vis-à-vis du revendeur et d'établir une session d'accès. Il se peut qu'un contexte sécurisé ait déjà été établi entre le consommateur et le revendeur par le biais des services de sécurité CORBA, auquel cas cette opération renvoie une référence vers une interface i_RetailerNamedAccess. L'opération active une exception e_AuthenticationError si le consommateur n'a pas encore été authentifié. Elle contient une référence vers une interface i_RetailerAuthenticate pouvant être utilisée pour authentifier et établir le contexte sécurisé. Cette opération est alors invoquée pour extraire la référence de l'interface i_RetailerNamedAccess.
- L'opération requestAnonymousAccess() permet au consommateur d'établir une session d'accès avec le revendeur sans révéler son identité. La session d'accès fournit l'accès à certains services, mais il se peut que le consommateur ait besoin de négocier avec le revendeur les services disponibles (ces services ne sont évidemment pas spécialisés pour le

consommateur). Le consommateur interagit avec le revendeur par le biais d'une interface i_RetailerAnonAccess. Cette opération est par ailleurs identique à l'opération requestNamedAccess().

8.1.2.2 Interface i_RetailerAuthenticate

L'interface i_RetailerAuthenticate permet l'authentification du consommateur et/ou du revendeur et l'acquisition de justificatifs, ainsi que l'établissement d'un contexte sécurisé. Cette interface fournit un procédé général d'authentification pouvant être utilisé pour la prise en charge de divers protocoles d'authentification.

Cette interface a pour objet principal de vérifier que le consommateur et le revendeur interagissent avec le domaine pour lequel ils ont été autorisés. Elle n'est pas nécessairement prévue pour identifier le consommateur (L'opération requestNamedAccess() est utilisée pour ce faire et fournir l'accès à ses services).

– L'opération getAuthenticationMethods() fournit la liste des méthodes d'authentification prises en charge par le revendeur.

– L'opération authenticate() permet au consommateur de choisir une méthode d'authentification, de transmettre des données d'authentification et de demander des justificatifs spécifiques pouvant être utilisés pour gérer un contexte sécurisé. Le revendeur renvoie alors (si nécessaire) ses données d'authentification, les données de mise à l'épreuve permettant au consommateur de répondre (si nécessaire) par le biais de l'opération continueAuthentication() et (si possible) les justificatifs demandés. Si un autre protocole d'authentification est nécessaire avant le renvoi de justificatifs, ces derniers peuvent être renvoyés par l'opération continueAuthentication().

– L'opération continueAuthentication() peut être invoquée une ou plusieurs fois après l'opération authenticate(). Elle permet au consommateur de fournir une réponse aux données de mise à l'épreuve renvoyées par l'opération authenticate() ou une invocation antérieure de l'opération continueAuthentication(). Les justificatifs demandés par le consommateur peuvent être renvoyés lors de la première invocation de l'opération continueAuthentication() ou des invocations suivantes, en fonction des prescriptions du protocole.

8.1.2.3 Interfaces disponibles durant une session d'accès

Les interfaces consommateur et revendeur sont disponibles durant une session d'accès.

Le consommateur prend en charge les interfaces suivantes du revendeur pouvant être utilisées durant la session d'accès:

– i_ConsumerAccess – Le revendeur peut utiliser cette interface pour obtenir des informations concernant les interfaces disponibles au sein du domaine consommateur; cette interface lui fournit les références d'interfaces vers d'autres interfaces du domaine consommateur.

– i_ConsumerInvite – Cette interface est utilisée par le revendeur pour notifier au consommateur des invitations à se joindre à des sessions de service. Le consommateur peut immatriculer une interface i_ConsumerInitial pour être en mesure de recevoir des invitations en dehors d'une session d'accès.

– i_ConsumerTerminal – Cette interface est utilisée par le revendeur au sein d'une session d'accès pour accéder à des informations de configuration de terminal, par exemple les applications installées, la configuration matérielle, les points NAP, etc.

– i_ConsumerAccessSessionInfo – Cette interface est utilisée par le revendeur pour fournir au consommateur des informations de changement d'état d'autres sessions d'accès dans lesquelles ce consommateur est engagé avec ce revendeur.

– i_ConsumerSessionInfo – Cette interface est utilisée par le revendeur pour fournir au consommateur des informations de changement d'état d'autres sessions d'accès dans

lesquelles ce consommateur est engagé avec ce revendeur. Les informations sont émises à destination de toutes les sessions de service utilisées par toutes les sessions d'accès avec ce revendeur.

Toutes [les interfaces des] consommateurs prises en charge peuvent être immatriculées auprès du revendeur à des fins d'utilisation pendant la durée ou en dehors d'une session d'accès en utilisant les opérations de l'interface i_RetailerNamedAccess. Il est également possible d'immatriculer d'autres interfaces propres au consommateur qui ne sont pas définies au niveau du point Ret-RP. L'immatriculation des trois premières interfaces indiquées plus haut est obligatoire; elle utilise l'opération setUserCtxt() de l'interface i_RetailerNamedAccess. La durée de vie de cette immatriculation est identique à celle de la session d'accès.

Le revendeur prend en charge deux interfaces utilisables durant des sessions d'accès. Le consommateur obtiendra une référence pour une seule de ces interfaces. S'il s'est authentifié en tant qu'utilisateur nommé en invoquant l'opération requestNamedAccess(), il obtiendra alors une référence pour l'interface i_RetailerNamedAccess; s'il s'est authentifié en tant qu'utilisateur anonyme en invoquant l'opération requestAnonymousAccess(), il obtiendra alors une référence pour l'interface i_RetailerAnonAccess:

– i_RetailerNamedAccess – Cette interface permet à un consommateur connu d'accéder aux services auxquels il est abonné ainsi que de démarrer et de gérer des sessions de service, etc.

– i_RetailerAnonAccess – Cette interface est utilisée par le revendeur pour notifier au consommateur des invitations à se joindre à des sessions de service. Le consommateur peut immatriculer une interface i_ConsumerInitial pour recevoir des invitations en dehors d'une session d'accès.

Un consommateur aura accès à l'une de ces interfaces durant une session d'accès, selon qu'il s'est authentifié en tant qu'utilisateur nommé ou anonyme. La définition actuelle du point Ret-RP ne permet pas à un utilisateur anonyme de devenir un utilisateur nommé au cours d'une même session d'accès.

Le revendeur prend également en charge l'interface suivante:

– i_DiscoverServicesIterator – Une référence à cette interface est renvoyée au consommateur après l'invocation de l'opération discoverServices() sur l'une des interfaces précédentes. Elle est utilisée pour extraire les descriptions de services autres que celles qui ont été renvoyées directement par l'opération discoverServices().

8.1.2.4 Interface i_ConsumerAccess

L'interface i_ConsumerAccess permet au revendeur d'accéder au domaine consommateur durant une session d'accès. Elle permet au revendeur de demander des références vers les interfaces prises en charge par ce domaine. Ces interfaces englobent celles qui sont définies au niveau du point Ret-RP ainsi que d'autres interfaces propres au revendeur.

– cancelAccessSession() – permet au revendeur de mettre fin à cette session d'accès. Une fois que cette opération a été invoquée, ni le consommateur ni le revendeur utiliseront les autres interfaces (il reste possible d'utiliser les interfaces immatriculées pour une utilisation en dehors de la session d'accès, ou les interfaces vers une autre session d'accès).

Cette interface hérite de l'interface i_UserAccess les opérations suivantes permettant au revendeur d'obtenir des références vers d'autres interfaces prises en charge par le consommateur:

– getInterfaceTypes() – permet au revendeur de détecter tous les types pris en charge par le domaine consommateur.

– getInterface() – permet au revendeur d'extraire une référence d'interface indiquant le type et les propriétés de l'interface.

– getInterfaces() – permet au revendeur d'extraire une liste de toutes les interfaces prises en charge par le consommateur.

Cette interface est immatriculée auprès du revendeur par le biais de l'opération setUserCtxt(); elle est disponible durant la session d'accès en cours.

8.1.2.5 Interface i_ConsumerInvite

L'interface i_ConsumerInvite permet au revendeur d'émettre des invitations à se joindre à une session de service durant une session d'accès. Elle est uniquement disponible pour la réception d'invitations durant une session d'accès. S'il souhaite recevoir des invitations en dehors d'une session d'accès, le consommateur doit alors immatriculer l'interface i_ConsumerInitial pour une utilisation en dehors d'une session d'accès.

– inviteUser() – permet au revendeur d'inviter le consommateur à se joindre à une session de service. La liste de paramètres contient une description du service et des informations suffisantes pour se joindre à la session. L'entrée dans la session n'est possible qu'au moyen de l'opération joinSessionWithInvitation() sur l'interface i_RetailerNamedAccess.

– cancelInviteUser() – permet au revendeur d'indiquer au consommateur l'annulation d'une invitation qui lui a été envoyée précédemment.

Cette interface est immatriculée auprès du revendeur par le biais de l'opération setUserCtxt(); elle est utilisable durant la session d'accès en cours.

8.1.2.6 Interface i_ConsumerTerminal

L'interface i_ConsumerTerminal interface permet au revendeur d'obtenir des informations concernant la configuration du terminal du domaine du consommateur ainsi que les applications.

– getTerminalInfo() – Cette opération permet au revendeur d'extraire des informations concernant le terminal du domaine du consommateur, telles que l'identité du terminal, son type, les points d'accès réseau et les applications utilisateur.

Cette interface est immatriculée auprès du revendeur par le biais de l'opération setUserCtxt(); elle est utilisable durant la session d'accès en cours.

8.1.2.7 Interface i_ConsumerAccessSessionInfo

L'interface i_ConsumerAccessSessionInfo permet au revendeur de fournir au consommateur des informations de changement d'état d'autres sessions d'accès dans lesquelles ce dernier est impliqué (par exemple, la création ou la suppression de telles sessions). Le consommateur reçoit uniquement des informations concernant des sessions d'accès dans lesquelles il est impliqué.

– newAccessSessionInfo() – Cette opération (unilatérale) est utilisée par le revendeur pour informer le consommateur au sujet d'une nouvelle session d'accès dans laquelle ce dernier est impliqué.

– endAccessSessionInfo() – Cette opération (unilatérale) est utilisée par le revendeur pour informer le consommateur de la fin d'une autre session d'accès.

– cancelAccessSessionInfo() – Cette opération (unilatérale) est utilisée par le revendeur pour informer le consommateur de l'annulation d'une session d'accès par le revendeur.

– newSubscribedServicesInfo() – Cette opération (unilatérale) est utilisée par le revendeur pour informer le consommateur qu'il a été abonné à de nouveaux services.

Cette interface n'est pas immatriculée auprès du revendeur par le biais de l'opération setUserCtxt(). Le consommateur doit, pour ce faire, immatriculer cette interface par le biais de l'interface i_RetailerNamedAccess. L'immatriculation peut être faite pour une utilisation pendant la durée ou en dehors d'une session d'accès.

8.1.2.8 Interface i_ConsumerSessionInfo

L'interface i_ConsumerSessionInfo permet au revendeur de fournir au consommateur des informations concernant des changements d'état de sessions de service dans lesquelles ce consommateur est impliqué. Les opérations d'information sont invoquées chaque fois qu'un changement d'état de la session de service affecte le consommateur (par exemple, lorsque la session est suspendue), mais non lorsque ce changement n'affecte pas le consommateur (par exemple, lorsqu'un autre participant quitte la session). Cette interface est informée des changements dans toutes les sessions de service impliquant le consommateur, y compris celles qui ne sont pas associées à cette session d'accès.

Le consommateur peut recevoir des informations de la part des opérations suivantes:

– newSessionInfo() – une nouvelle session de service a été démarrée;

– endSessionInfo() – une session de service s'est terminée;

– endMyParticipationInfo() – la participation du consommateur dans la session s'est terminée;

– suspendSessionInfo() – une session de service existante a fait l'objet d'une suspension;

– suspendMyParticipationInfo() – la participation du consommateur dans la session de service a fait l'objet d'une suspension;

– resumeSessionInfo() – une session suspendue a fait l'objet d'une reprise;

– resumeMyParticipationInfo() – le consommateur a repris sa participation dans la session;

– joinSessionInfo() – le consommateur s'est joint à une session de service.

Cette interface peut être immatriculée auprès du revendeur par le biais de l'opération setUserCtxt(), auquel cas elle est disponible uniquement durant la session d'accès en cours.

Elle peut également être immatriculée auprès du revendeur à tout autre instant au moyen des opérations d'immatriculation sur l'interface i_RetailerAccess, pour une utilisation pendant la durée ou en dehors d'une session d'accès.

8.1.2.9 Interface i_RetailerNamedAccess

L'interface i_RetailerNamedAccess permet à un consommateur connu d'accéder aux services auxquels il est abonné. Le consommateur utilise cette interface pour toutes les opérations au cours d'une session d'accès avec le revendeur. Une référence vers cette interface est renvoyée par l'invocation de l'opération requestNamedAccess() sur l'interface i_RetailerInitial une fois que le consommateur a été authentifié par le revendeur et qu'une session d'accès a été établie.

Cette interface fournit les opérations suivantes héritées de l'interface i_ProviderNamedAccess:

– **setUserCtxt**() – permet au consommateur de fournir au revendeur des informations concernant les interfaces du domaine consommateur ainsi que d'autres informations concernant ce domaine (par exemple, les applications utilisateur disponibles au sein du domaine consommateur, le système d'exploitation utilisé, etc.). Cette opération doit être invoquée immédiatement après la réception de la référence de cette interface faute de quoi des opérations suivantes risquent d'activer une exception.

– **listAccessSessions**() – permet au consommateur d'une session d'accès de prendre connaissance des autres sessions d'accès dans lesquelles il est engagé avec ce revendeur (il se trouve, par exemple, sur son lieu de travail, mais une session d'accès est établie à partir de son domicile et exécute une session de sécurité active).

– **endAccessSession**() – permet au consommateur de mettre fin à une session d'accès donnée, qui peut être soit la session en cours, soit une autre session identifiée par l'opération listAccessSessions(). Le consommateur peut également indiquer certaines actions devant être exécutées si des sessions de service sont actives.

– **getUserInfo**() – fournit le nom d'utilisateur du consommateur ainsi que d'autres propriétés.

– **listSubscribedServices**() – fournit la liste des services auxquels le consommateur est abonné. Il est possible d'utiliser une liste de propriétés pour préciser l'étendue des services qui ont fait l'objet d'un abonnement. L'opération renvoie des informations suffisantes pour permettre au consommateur de démarrer un service donné (auquel il est abonné).

– **discoverServices**() – extrait la liste de tous les services offerts par le revendeur. Le consommateur peut préciser l'étendue de la liste en fournissant certaines propriétés souhaitées pour le service et une taille maximale de la liste. Une référence à une interface i_DiscoverServicesIterator peut être utilisée pour extraire les services restants.

– **getServiceInfo**() – renvoie les informations concernant un service donné (identifié par l'invocation de son identificateur de service). Des informations similaires (t_ServiceProperties) peuvent être obtenues par le biais des opérations listSubscribedServices() ou discoverServices(), mais l'opération getServiceInfo() est une version simplifiée spécialisée pour un seul service et indépendante du statut d'abonnement.

– **listRequiredServiceComponents**() – extrait des informations concernant le déchargement du logiciel d'application dans le cas d'appelettes Java. Les informations de terminal sont fournies sous la forme d'un paramètre en entrée de manière à éviter une invocation explicite de l'opération getTerminalInfo(). Par exemple, dans le cas du déchargement d'une appelette Java, la liste de propriétés contiendra un élément avec un couple nom-valeur qui décrit le localisateur URL de l'appelette Java; le nom sera égal à "URL" et la valeur sera une chaîne contenant la valeur du localisateur URL.

– **listServiceSessions**() – fournit la liste des sessions de service du consommateur. La demande peut préciser l'étendue de la liste en indiquant une session d'accès et des propriétés de la session (par exemple active, suspendue, type de service, etc.).

– **getSession**(Models/InterfaceTypes/Interface/Interfaces)() – ces opérations effectuent l'extraction des informations concernant une session donnée.

– **listSessionInvitations**() – fournit la liste des invitations à se joindre à une session de service qui ont été envoyées au consommateur.

– **listSessionAnnouncements**() – fournit la liste des sessions de service qui ont fait l'objet d'une annonce. L'étendue de la liste peut être restreinte en fournissant des propriétés d'annonce.

– **startService**() – permet au consommateur de démarrer une session de service.

– **endSession**() – permet au consommateur de mettre fin à une session de service.

– **endMyParticipation**() – permet au consommateur de mettre fin à sa participation dans une session de service.

– **suspendSession**() – permet au consommateur de suspendre une session de service.

– **suspendMyParticipation**() – permet au consommateur de suspendre sa participation dans une session de service.

– **resumeSession**() – permet au consommateur de réactiver une session de service.

– **resumeMyParticipation**() – permet au consommateur de réactiver sa participation dans une session de service.

– **joinSessionWithInvitation**() – permet au consommateur de se joindre à une session de service dans laquelle il a été invité.

– **joinSessionWithAnnouncement**() – permet au consommateur de se joindre à une session de service qui a fait l'objet d'une annonce.

– **replyToInvitation**() – permet au consommateur de répondre à une invitation. Cette opération peut être utilisée pour informer la session de service invitante si le consommateur se joindra ou non à la session ou pour renvoyer l'invitation à un tiers (cette opération ne permet pas au consommateur de se joindre effectivement à la session).

L'interface prend également en charge les opérations suivantes héritées de l'interface i_ProviderAccessInterfaces, qui sont utilisées pour l'accès à des interfaces propres au revendeur:

– **getInterfaceTypes**() – permet au consommateur de détecter tous les types des interfaces prises en charge par le domaine revendeur.

– **getInterface**() – permet au consommateur d'extraire une référence d'interface qui fournit le type de l'interface et ses propriétés.

– **getInterfaces**() – permet au consommateur d'extraire une liste de toutes les interfaces prises en charge par le revendeur.

– **registerInterface**() – permet d'immatriculer une interface consommateur pour une utilisation dans la session d'accès en cours. Les immatriculations prennent fin avec la session d'accès ou lorsque l'opération unregisterInterface() est invoquée. Un index de l'interface est renvoyé pour permettre de mettre fin à l'immatriculation.

– **registerInterfaceOutsideAccessSession**() – permet à un consommateur d'immatriculer une interface pour une utilisation en dehors d'une session d'accès (l'interface immatriculée doit rester disponible en dehors d'une session d'accès entre le consommateur et le revendeur).

– **listRegisteredInterfaces**() – permet au consommateur d'obtenir la liste des interfaces qu'il a immatriculées auprès du revendeur. La liste précise quelles sont les interfaces immatriculées pour une utilisation durant une session d'accès ou en dehors d'une telle session.

– **unregisterInterface**() – permet au consommateur de mettre fin à l'immatriculation d'une interface, de sorte que le revendeur ne tente pas de l'utiliser (durant une session d'accès ou en dehors d'une telle session).

8.1.2.10 Interface i_RetailerAnonAccess

L'interface i_RetailerAnonAccess permet l'accès aux services du revendeur par un consommateur anonyme. Ce dernier utilise cette interface pour toutes les opérations au cours d'une session d'accès avec le revendeur. Cette interface est renvoyée lorsque le consommateur invoque l'opération requestAnonymousAccess() sur l'interface i_RetailerInitial.

Aucune opération n'est définie pour cette interface. Elle prendra en charge des opérations comparables à celles de l'interface i_RetailerNamedAccess.

8.1.2.11 Interface i_DiscoverServicesIterator

L'interface i_DiscoverServicesIterator est renvoyée par des invocations de l'opération discoverServices(). Cette opération est utilisée pour extraire une liste des services pris en charge par le revendeur et correspondant à un ensemble de propriété. Il se peut que la liste générée par cette opération soit trop volumineuse pour être renvoyée sous la forme d'un paramètre de retour. Cette interface permet d'extraire la liste sous la forme de morceaux d'une taille pouvant être "digérée" par le consommateur. Chaque invocation de l'opération discoverServices() renvoie une nouvelle instance de cette interface.

– **maxLeft**() – Le consommateur peut déterminer le nombre de services restants qui n'ont pas été fournis dans la liste.

– **nextN**() – Le consommateur peut indiquer qu'il souhaite recevoir des informations concernant les *n* services suivants.

– **destroy**() – Le consommateur indique au revendeur qu'il n'a plus besoin de cette interface.

8.1.3 Interfaces disponibles en dehors d'une session d'accès si elles sont immatriculées

Le consommateur peut immatriculer certaines de ses interfaces pour une utilisation par le revendeur en dehors de la session d'accès en cours. Une interface peut être immatriculée par le biais de l'opération registerInterfaceOutsideAccessSession() sur l'interface i_RetailerNamedAccess. Le revendeur obtient, en cas d'immatriculation, une référence vers l'interface lorsque le consommateur

ou lui-même met fin à la session d'accès en cours. Le revendeur peut invoquer des opérations sur cette interface en l'absence d'une session d'accès.

Le revendeur n'utilisera pas l'interface immatriculée durant la session d'accès au cours de laquelle l'immatriculation a été effectuée. Il peut continuer par la suite à utiliser l'interface jusqu'au moment où l'immatriculation se termine. Lorsqu'une nouvelle session d'accès est établie, le revendeur continue à invoquer des opérations sur l'interface immatriculée en plus des nouvelles interfaces fournies par cette session d'accès.

– i_ConsumerInitial. Cette interface permet au revendeur d'initier une session d'accès avec le consommateur. Elle lui permet également d'envoyer des invitations au consommateur en dehors d'une session d'accès.
– i_ConsumerTerminal. Le revendeur utilisera cette interface pour accéder, si nécessaire, à des informations de configuration de terminal. Prière de se référer à la description précédente.
– i_ConsumerAccessSessionInfo. Le revendeur utilisera cette interface pour fournir au consommateur des informations concernant des modifications de l'une quelconque de leurs sessions d'accès. Prière de se référer à la description précédente.
– i_ConsumerSessionInfo. Le revendeur utilisera cette interface pour fournir au consommateur des informations concernant des modifications de l'une quelconque de leurs sessions de service. Voir description précédente.

8.1.3.1 Interface i_ConsumerInitial

L'interface i_ConsumerInitial permet au revendeur de prendre contact avec le consommateur en dehors d'une session d'accès. Elle peut être utilisée pour demander au consommateur d'établir une session d'accès avec le revendeur et pour inviter un utilisateur à se joindre à une session de service.

Cette interface est uniquement disponible pour le revendeur si le consommateur l'a immatriculée durant une session d'accès (par le biais de l'opération registerInterfaceUntilUnregistered() opération sur l'interface i_RetailerNamedAccess). Elle n'est pas disponible par le biais d'un courtier, comme dans le cas de l'interface i_RetailerInitial.

Les opérations suivantes sont disponibles:

– requestAccess() – permet au revendeur de demander au consommateur l'établissement d'une session d'accès.
– inviteUserWithoutAccessSession() – permet au revendeur d'envoyer une invitation au consommateur alors que ce dernier n'est pas impliqué dans une session d'accès avec le revendeur.
– cancelInviteUserWithoutAccessSession() – permet au revendeur d'annuler une invitation envoyée au consommateur.

8.2 Interfaces utilisateur-fournisseur

Les interfaces définies ci-dessous sont utilisables au niveau du point Ret-RP. Les noms des interfaces font appel aux termes "*consumer*" et "*retailer*" (consommateur et revendeur) pour indiquer que leur utilisation est prévue au niveau du point Ret-RP. Les descriptions qui suivent englobent toutes les opérations utilisées au niveau du point Ret-RP.

D'autres points de référence peuvent avoir besoin d'utiliser des interfaces similaires pour des activités liées à l'accès (par exemple pour établir une session d'accès, démarrer des services, etc.). Un ensemble d'interfaces d'accès génériques a été défini pour permettre la réutilisation des interfaces et des opérations par d'autres points d'accès. Ces interfaces prennent en charge les rôles de session d'accès définis au 5.4 du présent supplément traitant de l'architecture de service. Les rôles pris en charge sont l'utilisateur d'accès et le fournisseur d'accès. Ils sont caractérisés par les préfixes i_User et i_Provider.

Les interfaces destinées à une utilisation au niveau du point Ret-RP ont été définies précédemment, y compris toutes les opérations héritées. Toutes les interfaces désignées par le suffixe i_Consumer ou i_Retailer sont héritées des interfaces i_User et i_Provider correspondantes. Toutes les spécialisations du point Ret-RP sont définies dans les interfaces i_Consumer ou i_Retailer. Aucune spécialisation n'est toutefois définie à l'heure actuelle.

Les figures qui suivent définissent la hiérarchie d'héritage pour les deux interfaces de point Ret-RP ainsi que les interfaces génériques utilisateur-fournisseur.

8.2.1 Interfaces utilisateur

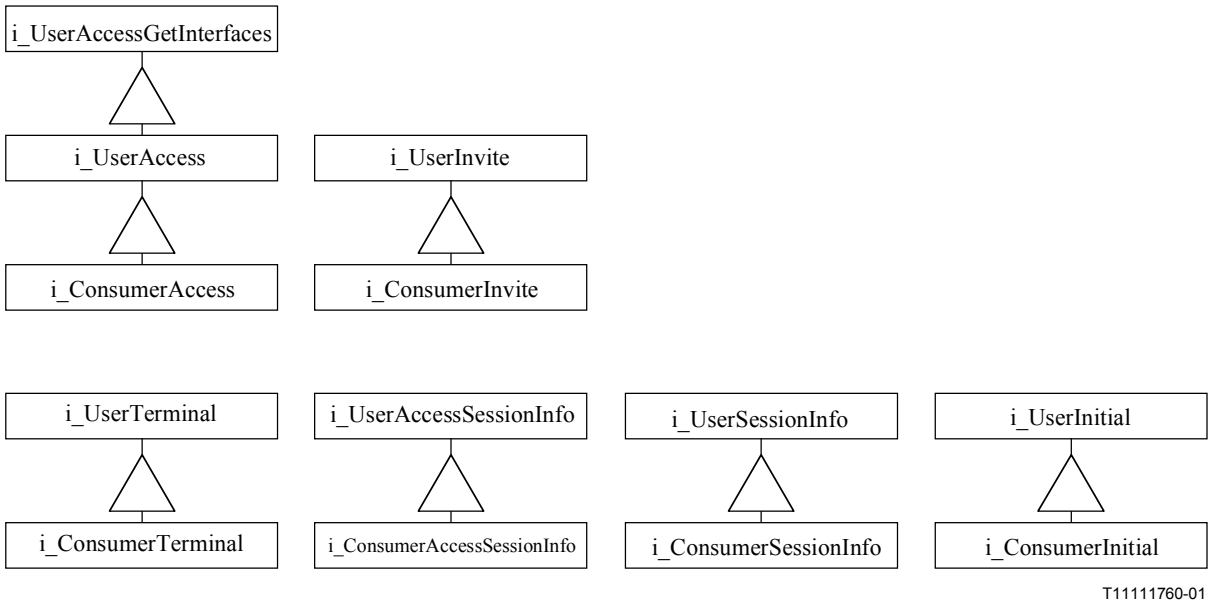


Figure 8-1 – Interfaces consommateur héritées des interfaces utilisateur

La Figure 8-1 représente les interfaces consommateur ainsi que les interfaces utilisateur dont elles héritent. Les interfaces utilisateur et consommateur sont de simples mises en correspondance (toutes les interfaces consommateur héritent d'une interface "utilisateur nommé" correspondante). Toutes les interfaces utilisateur définissent les opérations qui sont décrites au 8.1 pour les interfaces consommateur. La seule exception concerne l'interface i_UserAccess qui hérite toutes les opérations de l'interface i_UserAccessGetInterfaces pour permettre leur réutilisation par d'autres interfaces en vue de l'extraction de nouvelles interfaces.

8.2.1.1 Interface i_UserAccess

Cette interface hérite de l'interface abstraite i_UserAccessGetInterfaces et définit l'opération suivante:

– cancelAccessSession()

8.2.1.2 Interface i_UserInvite

Cette interface définit les opérations suivantes:

– inviteUser()

– cancelInviteUser()

8.2.1.3 Interface i_UserTerminal

Cette interface définit l'opération suivante:

– getTerminalInfo()

8.2.1.4 Interface i_UserAccessSessionInfo

Cette interface définit les opérations suivantes:

– newAccessSessionInfo()
– endAccessSessionInfo()
– cancelAccessSessionInfo()
– newSubscribedServicesInfo()

8.2.1.5 Interface i_UserSessionInfo

Cette interface définit les opérations suivantes:

– newSessionInfo()
– endSessionInfo()
– endMyParticipationInfo()
– suspendSessionInfo()
– suspendMyParticipationInfo()
– resumeSessionInfo()
– resumeMyParticipationInfo()
– joinSessionInfo()

8.2.1.6 Interface i_UserInitial

Cette interface définit les opérations suivantes:

– requestAccess()
– inviteUserOutsideAccessSession()
– cancelInviteUserOutsideAccessSession()

8.2.2 Interfaces fournisseur

La Figure 8-2 représente les interfaces revendeur ainsi que les interfaces fournisseur dont elles héritent.

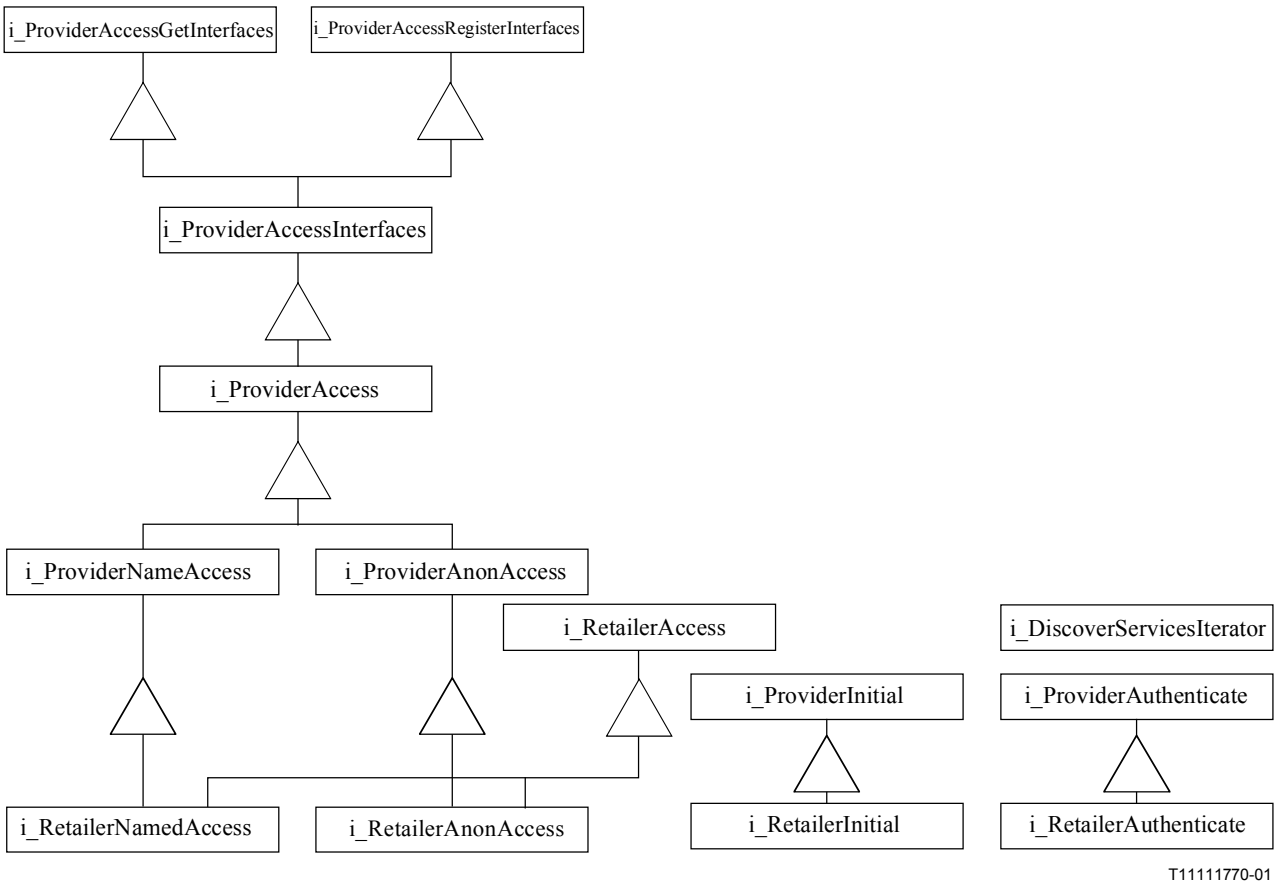


Figure 8-2 – Interfaces revendeur héritées des interfaces fournisseur

Les interfaces i_RetailerNamedAccess et i_RetailerAnonAccess possèdent une hiérarchie d'héritage multiple. Elles héritent toutes deux de l'interface i_RetailerAccess qui définit des opérations propres au point Ret-RP communes aux deux interfaces (aucune opération n'est définie à l'heure actuelle).

L'interface i_RetailerNamedAccess hérite également de l'interface i_ProviderNamedAccess qui définit les opérations disponibles pour le rôle de fournisseur d'accès générique lorsque le domaine utilisateur prend en charge un utilisateur connu. L'interface i_ProviderNamedAccess définit toutes les opérations fournies par l'interface i_RetailerNamedAccess.

L'interface i_RetailerAnonAccess hérite de l'interface i_ProviderAnonAccess qui définit les opérations disponibles pour le rôle de fournisseur d'accès générique lorsque le domaine utilisateur prend en charge un utilisateur anonyme. L'interface i_ProviderAnonAccess hérite uniquement à l'heure actuelle des opérations de l'interface i_ProviderAccess.

L'interface i_ProviderAccess définit le rôle de fournisseur d'accès générique à des fins de réutilisation pour d'autres points de référence. Elle est héritée par les deux interfaces i_ProviderNamedAccess et i_ProviderAnonAccess. Aucune opération n'est définie à l'heure actuelle pour cette interface. Dans le futur, certaines des opérations définies pour l'interface i_ProviderNamedAccess seront déplacées ici, du fait qu'elles sont communes aux deux interfaces et peuvent être employées pour des utilisateurs connus ou anonymes.

8.2.2.1 Interface i_ProviderNamedAccess

Cette interface définit les opérations suivantes et hérite d'autres opérations de l'interface i_ProviderAccess:

– setUserCtxt()

– listAccessSessions()

– endAccessSession()
– getUserInfo()
– listSubscribedServices()
– discoverServices()
– getServiceInfo()
– listRequiredServiceComponents()
– listServiceSessions()
– getSession(Models/InterfaceTypes/Interface/Interfaces)()
– listSessionInvitations()
– listSessionAnnouncements()
– startService()
– endSession()
– endMyParticipation()
– suspendSession()
– suspendMyParticipation()
– resumeSession()
– resumeMyParticipation()
– joinSessionWithInvitation()
– joinSessionWithAnnouncement()
– replyToInvitation()

8.2.2.2 Interface i_ProviderAnonAccess

Cette interface ne définit aucune opération. Elle hérite de l'interface i_ProviderAccess.

8.2.2.3 Interface i_ProviderAccess

Cette interface ne définit aucune opération. Elle hérite de l'interface i_ProviderAccessInterfaces.

8.2.3 Interfaces abstraites

Le présent sous-paragraphe décrit les interfaces abstraites qui sont héritées par certaines interfaces revendeur et consommateur. Elles ne sont pas exportées par le point de référence Ret. Leur objet principal est de fournir un procédé général d'immatriculation et d'extraction d'interfaces pour un domaine donné.

– i_UserAccessGetInterfaces – permet au fournisseur d'extraire, dans la session d'accès en cours, toutes les interfaces, ou uniquement celles qui possèdent certaines propriétés ou certains types d'interface.
– i_ProviderAccessGetInterfaces – permet à l'utilisateur d'extraire, dans la session d'accès en cours, toutes les interfaces, ou uniquement celles qui possèdent certaines propriétés ou certains types d'interface.
– i_ProviderAccessRegisterInterfaces – permet à l'utilisateur d'immatriculer des interfaces pour la durée de vie d'une session d'accès ou de manière permanente. Elle fournit également une opération permettant de mettre fin à l'immatriculation des interfaces.
– i_ProviderAccessInterfaces – hérité des deux interfaces précédentes sans fournir de fonctionnalités supplémentaires.

8.2.3.1 Interface i_UserAccessGetInterfaces

Cette interface définit les opérations suivantes:

– getInterfaceTypes()
– getInterface()
– getInterfaces()

8.2.3.2 Interface i_ProviderAccessGetInterfaces

Cette interface définit les opérations suivantes:

– getInterfaceTypes()
– getInterface()
– getInterfaces()

8.2.3.3 Interface i_ProviderAccessRegisterInterfaces

Cette interface définit les opérations suivantes:

– registerInterface()
– registerInterfaces()
– registerInterfaceOutsideAccessSession()
– registerInterfacesOutsideAccessSession()
– listRegisteredInterfaces()
– unregisterInterface()
– unregisterInterfaces()

8.2.3.4 Interface i_ProviderAccessInterfaces

Cette interface hérite des interfaces i_ProviderAccessGetInterfaces et i_ProviderAccessRegisterInterfaces sans fournir d'opération supplémentaire.

8.3 Vue informationnelle commune

Le présent sous-paragraphe décrit des types d'informations communs qui ont une probabilité élevée de réutilisation (par d'autres points de référence).

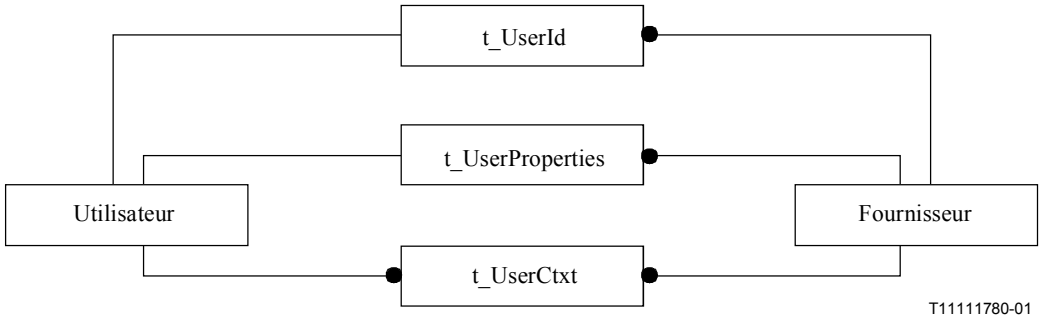


Figure 8-3 – Relations et cardinalités des types communs entre utilisateur et fournisseur

8.3.1 Propriétés et listes de propriétés

Une propriété est un attribut ou une qualité d'un objet. Dans le cas du point Ret-RP, les propriétés sont utilisées pour assigner une qualité à un objet ou pour rechercher des objets qui possèdent une certaine qualité.

Les objets concernés par le point Ret-RP peuvent être des utilisateurs, des services, des sessions, des interfaces, etc. Chacun de ces objets possédera diverses propriétés et chacune de ces propriétés possédera un domaine de valeurs et de structures (certaines qualités de ces objets sont suffisamment connues à l'heure actuelle, alors que d'autres propriétés seront propres au revendeur).

La structure t_Property a été définie en conséquence pour représenter une propriété. Sa définition IDL correspond à celle du service d'échange d'objets CORBA; elle est utilisée dans le module SPFEECommonTypes.

```
// module SPFEECommonTypes
typedef string t_PropertyName;
typedef sequence<t_PropertyName> t_PropertyNameList;
typedef any t_PropertyValue;
struct t_Property {
       t_PropertyName name;
       t_PropertyValue value;
};
typedef sequence<t_Property> t_PropertyList;

enum t_HowManyProps {none, some, all};
union t_SpecifiedProps switch (t_HowManyProps) {
       case some: t_PropertyNameList prop_names;
       case none:
       case all: octet dummy;
};

typedef string Istring;
```

Comme indiqué ci-dessus, la propriété t_Property est une structure constituée d'un nom et d'une valeur. La nom est une chaîne internationale et la valeur est du type any. Ce format permet au destinataire de la propriété de lire la chaîne et de rechercher une correspondance avec des propriétés dont il a connaissance. S'il s'agit d'une propriété connue, il connaîtra alors également le format de la valeur. La valeur ne doit pas être lue s'il s'agit d'une propriété non connue (une valeur du type any contient un code de type qui peut être recherché dans le référentiel d'interface pour déterminer le type de la valeur, mais ceci ne sera pas nécessaire dans la plupart des cas).

Les types t_Property et t_PropertyList sont utilisés comme qualités d'attribut pour des entités, lorsqu'on ne souhaite pas définir totalement quelles sont les qualités présentes (certaines de ces qualités peuvent également être propres au revendeur, de sorte que ces types sont utilisables pour des extensions du point Ret-RP).

Certaines caractéristiques du terminal peuvent, par exemple, être envoyées au revendeur après l'établissement d'une session d'accès. Le type t_TerminalProperties définit alors une liste de propriétés permettant d'envoyer ces caractéristiques au revendeur. Les caractéristiques devant être envoyées à un revendeur ne sont pas nécessairement connues pour tous les types de terminal et peuvent varier selon le cas.

Le point Ret-RP définit une propriété particulière de la liste t_TerminalProperties; cette propriété appelée "TERMINAL INFO" possède une valeur du type t_TerminalInfo. Le type t_TerminalInfo est une structure qui contient certaines informations concernant les caractéristiques du terminal. Lorsque le revendeur accède à la liste t_TerminalProperties et trouve une structure t_Property contenant le nom "TERMINAL INFO", il peut alors examiner la valeur pour déterminer les caractéristiques du terminal. Cette valeur sera toujours du type any, mais elle est formatée avec des informations contenues dans la structure t_TerminalInfo.

La structure t_TerminalInfo peut toutefois être incomplète ou non significative pour tous les types de terminaux. Si elle ne contient pas d'informations suffisantes, une version future de point Ret-RP ou un revendeur peut définir une autre propriété, appelée par exemple "ADDITIONAL TERMINAL INFO", qui contient les caractéristiques supplémentaires dans un format de valeur approprié. Le revendeur recevra dans ce cas les deux propriétés dans la liste t_TerminalProperties.

Si la structure t_TerminalInfo contient des informations non significatives, un revendeur peut alors définir une propriété complète qui sera émise par le consommateur à la place de la propriété "TERMINAL INFO".

Le point Ret-RP définit, lorsque cela est possible, des noms et des valeurs de propriété. Pour certaines listes de propriété, telles que t_InterfaceProperties, la détermination des propriétés qui leur sont associées est de la responsabilité du consommateur ou du revendeur.

```
// module SPFEECommonTypes
enum t_WhichProperties {
       NoProperties,
       SomeProperties,
       SomePropertiesNamesOnly,
       AllProperties,
       AllPropertiesNamesOnly
};

struct t_MatchProperties {
       t_WhichProperties whichProperties;
       t_PropertyList properties;
};
```

La structure t_MatchProperties est utilisée pour définir le type des valeurs renvoyées par certaines opérations. Ces opérations renvoient des listes d'objets. La structure t_MatchProperties est utilisée pour identifier les objets renvoyés en fonction de certaines propriétés (par exemple, pour l'opération listSubscribedServices, l'objet en question sera un service auquel un consommateur est abonné. Le paramètre t_MatchProperties définit les propriétés qui doivent être renvoyées dans la liste).

La structure t_MatchProperties contient une liste t_PropertyList et un type énuméré t_WhichProperties. La liste t_PropertyList contient les propriétés pour lesquelles une correspondance doit être recherchée. Le type t_WhichProperties indique quelles sont les propriétés (toutes, certaines ou aucune) qui doivent être en correspondance, ainsi que si cette correspondance porte sur le nom et la valeur ou uniquement sur le nom de la propriété.

Pour l'opération listSubcribedServices, si par exemple, le type t_WhichProperties est égal à:

NoProperties, alors les services ayant fait l'objet d'un abonnement n'ont pas l'obligation de correspondre à l'une des propriétés, de sorte que tous ces services seront renvoyés dans la liste.

SomeProperties, alors les services ayant fait l'objet d'un abonnement doivent correspondre à l'une au moins des propriétés de la liste t_PropertyList pour figurer dans la liste renvoyée (la correspondance portant sur le nom et la valeur).

SomePropertiesNamesOnly, alors les services ayant fait l'objet d'un abonnement doivent correspondre à l'un au moins des noms de propriété de la liste t_PropertyList pour figurer dans la liste renvoyée. Les valeurs des propriétés de la liste t_PropertyList peuvent ne pas avoir de signification et ne doivent pas être utilisées.

AllProperties, alors les services ayant fait l'objet d'un abonnement doivent correspondre à toutes les propriétés de la liste t_PropertyList pour figurer dans la liste renvoyée (la correspondance portant sur le nom et la valeur).

AllPropertiesNamesOnly, alors les services ayant fait l'objet d'un abonnement doivent correspondre à tous les noms de propriété de la liste t_PropertyList pour figurer dans la liste renvoyée. Les valeurs des propriétés de la liste t_PropertyList peuvent ne pas avoir de signification et ne doivent pas être utilisées.

8.3.2 Informations utilisateur

```
// module SPFEECommonTypes
typedef Istring t_UserId;
typedef Istring t_UserName;
typedef t_PropertyList t_UserProperties;
```

Le type t_UserId identifie l'utilisateur vis-à-vis du revendeur. Il est non ambigu pour cet utilisateur au sein du domaine de ce revendeur. Il est utilisé dans l'opération requestNamedAccess() et renvoyé par l'opération getUserInfo(). Le type t_UserId ne contient pas le nom du revendeur et ne peut donc pas être utilisé pour prendre contact avec ce dernier. Il peut être émis à destination d'un courtier ou d'un service de dénomination lors d'une tentative de contact avec un revendeur en utilisant le nom de ce dernier.

Le type t_UserProperties est une séquence de types t_UserProperty. Il contient des informations concernant l'utilisateur que ce dernier souhaite communiquer au revendeur. Les noms de propriété suivants ont été définis pour le type t_UserProperty. D'autres noms de propriété, spécifiques du fournisseur, sont autorisés.

```
// Property Names defined for t_UserProperties:
// name:   "PASSWORD"
// value:  string
// use:    user password, as a string.

// name:   "SecurityContext"
// value:  opaque
// use:    to carry a retailer specific security context
//     e.g. could be used for an encoded user password.
```

8.3.2.1 Exception e_UserDetailsError

```
// module SPFEECommonTypes
enum t_UserDetailsErrorCode {
       InvalidUserName,
       InvalidUserProperty
};

exception e_UserDetailsError {
       t_UserDetailsErrorCode errorCode;
       t_UserName name;
       t_PropertyErrorStruct propertyError;
};
```

L'exception e_UserDetailsError est définie pour des opérations nécessitant un paramètre t_UserDetails (par exemple, l'opération inviteUserReq() pour la partie "utilisation" du point Ret-RP). L'exception est activée si le paramètre t_UserName ou t_UserProperties n'est pas valide.

Les codes d'erreur suivants peuvent être utilisés pour définir le problème rencontré:

– InvalidUserName:

  Le paramètre t_UserName ne contient pas un identificateur de participant valide (ceci peut provenir du fait qu'il n'a pas un format correct ou parce qu'il ne fait pas référence à un utilisateur connu).

  Le nom de la variable t_UserName de l'exception contient la valeur du paramètre t_UserName figurant dans l'invocation de l'opération.

– InvalidUserProperty:

Le paramètre t_UserProperties est erroné. Le paramètre propertyError de l'exception décrit le type d'erreur dans la propriété utilisateur. Si l'élément propertyError de l'exception contient la valeur InvalidPropertyName, il s'agit alors d'un nom de propriété qui n'est pas valide pour cette opération. S'il contient la valeur InvalidPropertyValue, il s'agit alors d'une valeur qui n'est pas valide pour le nom de propriété. Si l'élément propertyError contient la valeur UnknownPropertyName, il s'agit alors d'un nom de propriété non reconnu par la session. Certaines sessions peuvent ignorer les types t_PropertyName non reconnus. Elles ne doivent pas traiter la valeur t_PropertyValue associée au nom t_PropertyName mais peuvent traiter les autres structures t_Property figurant dans la liste t_UserProperties. De telles sessions n'ont pas l'obligation d'activer une exception avec ce code d'erreur.

8.3.3 Informations de contexte utilisateur

```
// module SPFEECommonTypes
typedef Istring t_UserCtxtName;

// module SPFEEProviderAccess
struct t_UserCtxt {
   SPFEECommonTypes::t_UserCtxtName                    ctxtName;
   SPFEEAccessCommonTypes::t_AccessSessionId          asId;
   Object accessIR;                                // type: i_UserAccess
   Object terminalIR;                              // type: i_UserTerminal
   Object inviteIR;                                // type: i_UserInvite
   Object sessionInfoIR;                           // type: i_UserSessionInfo
   SPFEEAccessCommonTypes::t_TerminalConfig          terminalConfig;
};
```

La structure t_UserCtxt fournit au revendeur des informations concernant le domaine consommateur, incluant le nom du contexte, les interfaces disponibles durant cette session d'accès et des informations de configuration du terminal. La structure t_UserCtxt est utilisée uniquement dans la partie "accès" du point Ret-RP, mais elle figure ici pour faciliter la compréhension du type t_UserCtxtName. Le sous-paragraphe 8.4 en donne une description complète.

Le type t_UserCtxtName correspond au nom donné à ce contexte consommateur. Ce nom est généré par le domaine consommateur. Il est utilisé pour faire la distinction entre des sessions d'accès pour divers domaines consommateur ou terminaux. Le nom t_UserCtxtName est renvoyé dans une liste de session d'accès (avec l'identificateur t_AccessSessionId) car il fournit sous une forme plus conviviale le nom du "terminal" auquel la session d'accès est connectée.

8.3.4 Types liés à l'utilisation

8.3.4.1 Type t_SessionId

```
// module SPFEECommonTypes
typedef unsigned long t_SessionId;
```

Toutes les opérations sur les interfaces de domaine participant (incluant toutes les opérations exécutables et toutes les opérations d'information) contiennent un paramètre t_SessionId. Ce dernier permet au domaine participant d'identifier la session de service qui émet chaque demande d'opération. Il s'agit d'un entier long de 32 bits. L'identificateur t_SessionId est identique à l'identificateur sessionId fourni pour cette session par l'opération startService() ou joinSession() (ceci signifie que l'identificateur qui figure dans la liste de sessions de la partie "accès" correspondant à ce paramètre t_SessionId fera référence à la même session). Le domaine utilisateur peut activer le code PD_InvalidSessionId dans l'exception e_PartyDomainError s'il ne reconnaît pas l'identificateur t_SessionId.

8.3.4.2 Type t_ParticipantSecretId

```
// module SPFEECommonTypes
typedef sequence<octet, 16> t_ParticipantSecretId;
```

Toutes les opérations sur les interfaces de domaine fournisseur de la session de service (y compris toutes les demandes) contiennent un paramètre t_ParticipantSecretId. Ce type est également renvoyé par les demandes pour le démarrage d'une session de service ou pour se joindre à une telle session.

Ceci permet à une session de service d'identifier l'émetteur de chaque demande d'opération. Il s'agit d'une clé d'une longueur de 128 bits. Le format de la clé n'est pas défini, à part le fait qu'une clé contenant des bits tous à "0" indique que le participant ne connaît pas ou n'a pas besoin de clé. La session peut activer un code d'erreur InvalidParticipantSecretId dans l'exception e_UsageError si la demande nécessite une clé.

Le paramètre t_ParticipantSecretId est fourni pour permettre l'implémentation de sessions en utilisant une seule interface vers tous les participants. La session peut avoir une certitude raisonnable que les demandes sont effectivement émises par l'utilisateur identifié.

Il n'est pas prévu d'utiliser le paramètre t_ParticipantSecretId comme mécanisme principal de sécurité. La sécurité CORBA ou d'autres contextes de sécurité doivent être utilisés comme mécanisme sous-jacent pour des interactions entre domaines participant et utilisateur.

8.3.5 Invitations et annonces

Les invitations permettent de demander à un utilisateur final de se joindre à une session en cours. Elles sont remises au domaine consommateur pour l'utilisateur final si une session d'accès existe. En l'absence d'une session d'accès avec le domaine consommateur, l'invitation peut être remise à une interface immatriculée au préalable ou être stockée jusqu'au moment de l'établissement d'une session d'accès. Une invitation contient les informations suffisantes permettant à l'utilisateur d'identifier l'utilisateur, qui a demandé l'émission de l'invitation, d'identifier et de se joindre à la session ou de décliner l'invitation (toutes ces opérations sont définies au niveau de la partie "accès" du point Ret-RP et le revendeur est toujours impliqué dans la procédure permettant au consommateur de détecter la session et de s'y joindre).

```
// module SPFEEAccessCommonTypes
typedef unsigned long t_InvitationId;
typedef SPFEECommonTypes::Istring t_InvitationReason;

struct t_InvitationOrigin {
      SPFEECommonTypes::t_UserId              userId;
      SPFEECommonTypes::t_SessionId           sessionId;
};

struct t_SessionInvitation {
      t_InvitationId                          id;
      SPFEECommonTypes::t_UserId              inviteeId;
      t_SessionPurpose                        purpose;
      t_ServiceInfo                           serviceInfo;
      t_InvitationReason                      reason;
      t_InvitationOrigin                      origin;
      SPFEECommonTypes::t_PropertyList        invProperties;
};

typedef sequence<t_SessionInvitation> t_InvitationList;

// module SPFEECommonTypes
enum t_InvitationReplyCodes {
      SUCCESS, UNSUCCESSFUL, DECLINE, UNKNOWN, ERROR,
      FORBIDDEN, RINGING, TRYING, STORED, REDIRECT, NEGOTIATE,
      BUSY, TIMEOUT
};
```

```
typedef t_PropertyList t_InvitationReplyProperties;

struct t_InvitationReply {
      t_InvitationReplyCodes reply;
      t_InvitationReplyProperties properties;
};
```

La structure t_SessionInvitation décrit la session de service dans laquelle le consommateur a été invité et fournit un élément t_InvitationId lui permettant d'identifier cette invitation au moment où il se joint à la session (aucune référence n'est fournie au sujet des interfaces vers la session, ni aucune information qui permettrait au consommateur de se joindre à la session sans établir au préalable une session d'accès avec ce revendeur). La structure fournit également un élément t_UserId qui contient l'identificateur de l'utilisateur invité. Le domaine consommateur peut vérifier que l'invitation est destinée à un utilisateur final connu du domaine.

L'élément t_SessionPurpose est une chaîne décrivant l'objet de la session. L'objet de la session peut être décrit au moment du démarrage de la session (par le biais de la propriété t_StartServiceSSProperties) ou au cours de la session.

L'élément t_ServiceInfo est le service, ayant fait l'objet d'un abonnement, que le consommateur peut utiliser pour se joindre à la session. Sa description est donnée au 8.4.

L'élément t_InvitationReason est une chaîne indiquant pour quel motif cette invitation a été remise à l'utilisateur invité. Il peut être défini par le participant qui a demandé l'invitation ou par la session.

La structure t_InvitationOrigin définit à quel endroit l'invitation a été générée. Elle contient l'identification de l'utilisateur qui a démarré la session et son identificateur pour cette session.

Le consommateur renvoie une réponse t_InvitationReply lui permettant de fournir au revendeur des informations concernant l'action qu'il va effectuer pour l'invitation. Les codes de réponse suivants sont définis:

– SUCCESS – le consommateur accepte l'invitation. Il devra établir une session d'accès avant de pouvoir se joindre à la session de service, cette session d'accès ne devant pas être établie à partir du terminal qui a reçu l'invitation. Le consommateur utilisera ensuite l'opération joinSessionWithInvitation() sur l'interface i_RetailerNamedAccess pour se joindre à la session. Le consommateur peut utiliser l'opération replyToInvitation() pour "changer d'avis" et ne pas se joindre à la session, mais il doit déjà avoir répondu dans ce cas au moyen du code RINGING ou d'un code de réponse autre que SUCCESS.
– UNSUCCESSFUL – le consommateur n'a pas pu être contacté par le biais de cette opération (il ne se joindra pas à la session à la suite de cette invitation; si toutefois la même invitation a été envoyée à plusieurs interfaces, une réponse d'une autre interface peut indiquer que le consommateur va se joindre à la session).
– DECLINE – le consommateur refuse de se joindre à la session.
– UNKNOWN – le consommateur auquel a été envoyée l'invitation n'est pas connu de cette interface (l'invitation t_SessionInvitation contient un identificateur t_UserId permettant au domaine consommateur de vérifier que l'invitation est destinée à un utilisateur connu de ce domaine).
– FAILED – le consommateur n'est pas en mesure de se joindre à la session de service (aucun motif n'est fourni; l'invitation peut avoir un format incorrect ou le consommateur peut ne pas être en mesure de se joindre à une session).
– FORBIDDEN – le domaine consommateur n'a pas l'autorisation d'accepter la demande.
– RINGING – le consommateur est connu de ce domaine et la prise de contact est en cours. Le revendeur ne doit pas faire l'hypothèse que le consommateur va se joindre à la session (si tel était le cas, il pourrait alors procéder comme décrit ci-dessus pour le code SUCCESS; il peut

utiliser l'opération replyToInvitation() sur l'interface i_RetailerNamedAccess s'il souhaite fournir au revendeur des informations de statut concernant cette invitation).

– TRYING – le consommateur est connu de ce domaine mais ne peut pas être contacté directement. Le domaine consommateur effectue certaines actions pour tenter de contacter le consommateur. Le revendeur peut traiter ce cas comme une réponse RINGING.
– STORED – le consommateur est connu de ce domaine mais n'est pas contacté pour l'instant. L'invitation a été stockée par le consommateur pour une extraction ultérieure (le revendeur peut traiter ce cas comme une réponse RINGING, mais il peut se passer un certain temps avant la réponse du consommateur).
– REDIRECT – le consommateur est connu de ce domaine mais n'est pas disponible au niveau de cette interface. Le revendeur doit utiliser l'adresse fournie dans la liste t_InvitationReplyProperties pour contacter le consommateur.
– NEGOTIATE – le consommateur est connu de ce domaine mais n'a pas encore été contacté pour le moment. La liste t_InvitationReplyProperties contient un ensemble de variantes que le revendeur peut utiliser pour contacter le consommateur (ces variantes ne sont pas définies pour le point Ret-RP, de sorte qu'il s'agit pour l'instant d'une fonctionnalité propre au revendeur).
– BUSY – le consommateur ne peut pas être contacté parce qu'il est occupé. Ce code doit être traité comme une réponse UNSUCCESSFUL.
– TIMEOUT – le consommateur ne peut pas être contacté car un débordement de temporisation s'est manifesté au sein du domaine consommateur pendant la tentative de prise de contact; ceci signifie que le domaine consommateur utilise une temporisation pour la méthode de prise de contact (par exemple, l'ouverture d'une fenêtre ou une sonnerie téléphonique) et que cette temporisation a débordé. Ce code doit être traité comme une réponse UNSUCCESSFUL.

Les codes de réponse à une invitation sont définis conformément au projet de Norme "Protocole d'initiation de session" du sous-groupe MMUSIC (commande de session multimédia avec participants multiples (*multimedia multiparty session control*) du groupe de travail d'ingénierie Internet (*IETF*)).

Les annonces permettent à une session de se faire connaître d'un "groupe" d'utilisateurs finaux. Les annonces ne sont pas destinées à un utilisateur donné, ni "livrées" à l'utilisateur final, mais sont stockées par le domaine revendeur jusqu'au moment où le domaine consommateur en demande la liste. Les annonces sont renvoyées au consommateur en fonction des "groupes" auxquels il appartient (ces groupes sont définis par des propriétés utilisateur, mais aucun procédé spécifique n'a été spécifié au niveau du point Ret-RP pour leur définition). Les annonces contiennent des informations suffisantes pour que l'utilisateur puisse se joindre à la session (cette opération est définie au niveau de la partie "accès" du point Ret-RP; le revendeur est toujours impliqué dans le processus d'autorisation de l'utilisateur pour détecter et se joindre à la session).

Projet de définition: La structure des annonces est à l'état de projet.

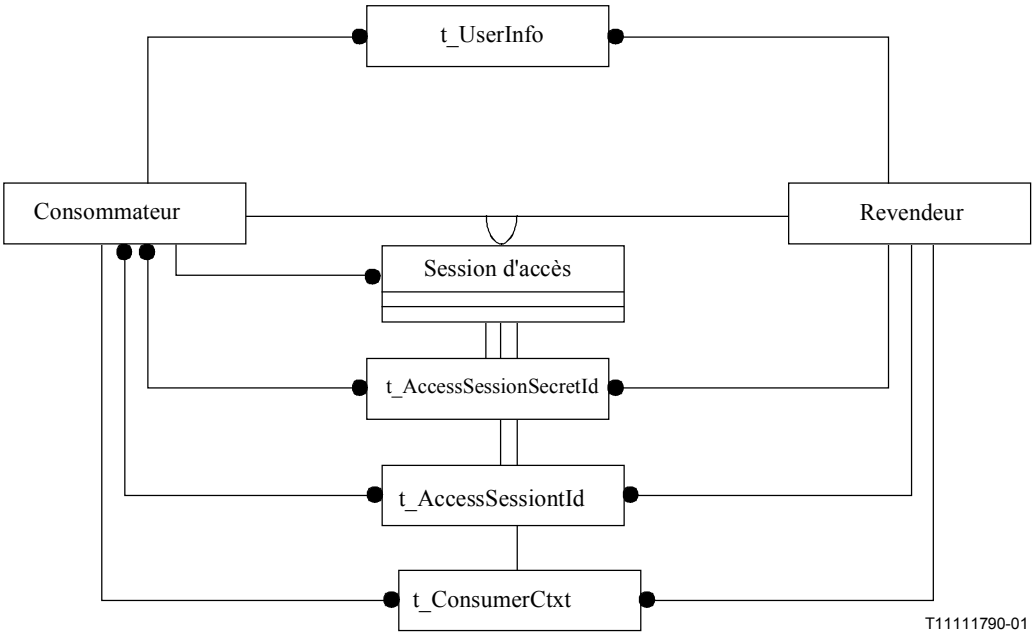
```
// module SPFEECommonTypes
typedef t_PropertyList t_AnnouncementProperties;

struct t_SessionAnnouncement {
       t_AnnouncementId announcementId;
       t_SessionPurpose sessionPurpose;
       t_ServiceInfo serviceInfo;
       t_AnnouncementProperties properties;
};

typedef sequence<t_SessionAnnouncement> t_AnnouncementList;

// module SPFEEAccessCommonTypes
typedef unsigned long t_AnnouncementId;
```

La structure t_SessionAnnouncement décrit la session qui fait l'objet d'une annonce et le "groupe" d'utilisateurs auxquels l'annonce est diffusée. Il s'agit d'une structure contenant l'identificateur de l'annonce, le but de la session, les informations de service et une liste des propriétés de l'annonce. Aucune valeur ni nom de propriété ne sont définis au niveau du point Ret-RP pour des annonces. Les propriétés de l'annonce permettent aux revendeurs de définir leurs propres types d'annonce qui peuvent être transmis par le biais des opérations d'annonce définies au niveau du point Ret-RP.

L'élément t_AnnouncementId identifie une annonce au sein du domaine consommateur. Le domaine consommateur peut demander une liste des annonces qui sont associées à un utilisateur final. L'élément t_AnnouncementId est utilisé par le domaine consommateur pour faire la distinction entre les annonces qu'il reçoit. Les identités de chaque annonce peuvent uniquement être employées par cet utilisateur. Elles n'identifient pas de manière non ambiguë l'annonce pour tous les consommateurs d'un revendeur.

8.4 Vue des informations d'accès

Le présent sous-paragraphe décrit les types d'informations transmis au niveau du point Ret-RP. Les types sont transférés par les opérations définies pour les interfaces d'accès.



8.4.1 Informations de la session d'accès

```
// module SPFEEAccessCommonTypes
typedef unsigned long t_AccessSessionId;
typedef sequence<octet, 16> t_AccessSessionSecretId;
```

Le type t_AccessSessionId est utilisé pour identifier une session d'accès. L'identificateur t_AccessSessionId de la session d'accès actuelle du consommateur est renvoyé par l'opération requestNamedAccess() ou requestAnonAccess(). L'identificateur t_AccessSessionId pour d'autres sessions d'accès peut être déterminé par le biais de l'opération listAccessSessions() sur l'interface i_RetailerNamedAccess (les utilisateurs anonymes ne peuvent avoir qu'une seule session d'accès et, en conséquence, un seul identificateur t_AccessSessionId). Le domaine de validité de l'identificateur t_AccessSessionId est relatif à un consommateur, c'est-à-dire que tous les identificateurs t_AccessSessionId d'un même consommateur (t_UserId) sont non ambigus.

Le type t_AccessSessionSecretId est utilisé pour identifier la session d'accès qui fait l'objet d'une demande pour l'opération requestNamedAccess(). Chaque session d'accès d'un consommateur

possède un identificateur t_AccessSessionSecretId non ambigu qui est renvoyé par l'opération requestNamedAccess().

Toutes les opérations requestNamedAccess() ont comme premier paramètre un identificateur t_AccessSessionSecretId. Ce paramètre peut être examiné par le revendeur pour déterminer quelle est la session d'accès du consommateur qui a effectué la demande. Ceci est utile lorsque le comportement de cette demande dépend du contexte consommateur (par exemple, l'opération startService() vérifie le contexte consommateur pour déterminer si ce service peut être utilisé).

L'identificateur t_AccessSessionSecretId est connu uniquement au sein de la session d'accès où il a été créé. Il n'est pas connu des autres sessions d'accès du même consommateur et ne peut pas être obtenu par l'invocation de l'opération listAccessSessions(). Ceci est nécessaire parce que cet identificateur est utilisé pour indiquer la session d'accès au sein de laquelle la demande est faite. Si une autre session d'accès prend connaissance de l'identificateur t_AccessSessionSecretId de cette session d'accès, elle serait en mesure de l'utiliser pour affirmer à tort que la demande provient de cette autre session d'accès. L'identificateur ne doit donc pas être divulgué à un consommateur humain ou à d'autres applications au sein du domaine consommateur. L'identificateur t_AccessSessionSecretId ne constitue pas en lui-même un mécanisme de sécurité car la sécurité CORBA reste nécessaire pour l'établissement de contextes sécurisés entre les domaines consommateur et revendeur. Il permet toutefois au revendeur de retrouver aisément l'émetteur d'une demande donnée.

8.4.2 Informations utilisateur

La plus grande partie des informations liées à l'utilisateur est décrite au 8.3 qui traite des types communs.

```
// module SPFEEAccessCommonTypes
struct t_UserInfo {
       SPFEECommonTypes::t_UserId userId;
       SPFEECommonTypes::t_UserName name;
       SPFEECommonTypes::t_UserProperties userProperties;
};
```

La structure t_UserInfo décrit l'utilisateur. Elle contient l'identificateur t_UserId, le nom de l'utilisateur et la liste t_UserProperties. Elle est renvoyée par l'opération getUserInfo() sur l'interface i_ProviderAccess.

8.4.3 Informations de contexte utilisateur

```
// module SPFEECommonTypes
typedef Istring t_UserCtxtName;

// module SPFEEProviderAccess
struct t_UserCtxt {
       SPFEECommonTypes::t_UserCtxtName ctxtName;
       SPFEEAccessCommonTypes::t_AccessSessionId asId;
       Object accessIR;                    // type: i_UserAccess[40]
       Object terminalIR;                  // type: i_UserTerminal
       Object inviteIR;                    // type: i_UserInvite
       Object sessionInfoIR;               // type: i_UserSessionInfo
       SPFEEAccessCommonTypes::t_TerminalConfig terminalConfig;
};
```

[40] Le type utilisé dans la description IDL de ce paramètre est le type de la classe de base. Le type effectif dépend du point de référence utilisé. Le revendeur peut, dans ce cas, s'attendre au type i_Consumer. Prière de se référer également à la remarque concernant l'opération requestNamedAccess() pour l'objet namedAccessIR.

La structure t_UserCtxt fournit au revendeur des informations concernant l'utilisateur et le domaine consommateur, incluant le nom du contexte, les interfaces disponibles durant cette session d'accès et des informations de configuration de terminal.

L'élément t_UserCtxtName contient le nom du contexte consommateur. Il est généré par le domaine consommateur et utilisé pour faire la distinction entre les sessions d'accès vers divers domaines consommateur ou terminaux. L'élément UserCtxtName est renvoyé dans les listes de session d'accès (ainsi que l'élément t_AccessSessionId) car il fournit une forme plus conviviale du nom du terminal auquel la session d'accès est connectée.

L'élément accessIR contient une référence vers l'interface i_UserAccess prise en charge par le domaine consommateur pour une utilisation dans cette session d'accès.

L'élément terminalIR contient une référence vers l'interface i_UserTerminal prise en charge par le domaine consommateur pour une utilisation dans cette session d'accès.

L'élément inviteIR contient une référence vers l'interface i_UserInvite prise en charge par le domaine consommateur pour une utilisation dans cette session d'accès.

Les trois références d'interface précédentes doivent être positionnées vers des interfaces valides au sein du domaine consommateur.

L'élément sessionInfoIR contient une référence vers l'interface i_UserSessionInfo prise en charge par le domaine consommateur pour une utilisation durant cette session d'accès. Il n'est pas nécessaire de fournir une référence pour cette interface.

La structure t_TerminalConfig contient l'identificateur et le type du terminal, l'identificateur et le type du point d'accès réseau ainsi qu'une liste de propriétés du terminal. Deux types de propriétés ont été définis, à savoir t_TerminalInfo décrite ci-dessous et t_ApplicationInfoList qui définit la liste des applications utilisateur du terminal.

La structure t_TerminalInfo donne les détails concernant le type de terminal, le système d'exploitation, etc.

```
// module SPFEEAccessCommonTypes
struct t_TerminalInfo {
       t_TerminalType terminalType;
       string operatingSystem;              // includes the version
       SPFEECommonTypes::t_PropertyList networkCards;
       SPFEECommonTypes::t_PropertyList devices;
       unsigned short maxConnections;
       unsigned short memorySize;
       unsigned short diskCapacity;
};
```

Projet de définition: cette structure est à l'état de projet et fait actuellement l'objet d'une revue (une nouvelle version sera produite pour la réponse révisée).

L'élément t_TerminalType contient un type énuméré qui indique les types de terminal.

L'élément operatingSystem fournit sous forme de chaîne le type et la version du système d'exploitation.

Les éléments networkCards et devices contiennent des listes de propriétés des équipements physiques du terminal. Les noms et valeurs de propriété ne sont pas définis à l'heure actuelle, de sorte que leur utilisation est propre au revendeur.

L'élément maxConnections indique le nombre maximal de connexions réseau pouvant être prises en charge par le terminal.

L'élément memorySize indique la taille de mémoire RAM en mégaoctets.

L'élément diskCapacity indique la taille de stockage disque en mégaoctets.

8.4.4 Informations de service et de session

```
// module SPFEEAccessCommonTypes
struct t_ServiceInfo {
       t_ServiceId id;
       t_UserServiceName name;
       t_ServiceProperties properties;
};
```

La structure t_ServiceInfo décrit un service auquel le consommateur est abonné.

L'élément t_ServiceId identifie le service d'une manière non ambiguë au sein de l'ensemble des services auxquels le consommateur est abonné (d'autres consommateurs peuvent être abonnés au même service mais utiliseront un identificateur t_ServiceId différent). La valeur de l'élément t_ServiceId persiste durant la session.

L'élément t_UserServiceName fournit sous forme d'une chaîne le nom du service choisi par l'abonné au moment où il s'abonne. Cette chaîne constitue le nom du service affiché pour l'utilisateur.

L'élément t_ServiceProperties contient une liste de propriétés qui définit les caractéristiques du service. Cette liste peut être utilisée pour rechercher les types de service possédant les mêmes caractéristiques, par exemple par le biais de l'opération discoverServices() sur l'interface i_RetailerNamedAccess. Aucune propriété n'est définie à l'heure actuelle pour le type t_ServiceProperties de sorte que son utilisation est propre au revendeur.

```
// module SPFEEAccessCommonTypes
struct t_SessionInfo {
       SPFEECommonTypes::t_SessionId id;
       t_SessionPurpose purpose;
       SPFEECommonTypes::t_ParticipantSecretId secretId;
       SPFEECommonTypes::t_PartyId myPartyId;
       t_UserSessionState state;
       SPFEECommonTypes::t_InterfaceList itfs;
       SPFEECommonTypes::t_SessionModelList sessionModels;
       SPFEECommonTypes::t_SessionProperties properties;
};
```

La structure t_SessionInfo contient des informations qui permettent au domaine consommateur de faire référence à une session donnée lorsqu'il utilise des interfaces au sein d'une session d'accès (par exemple, i_RetailerNamedAccess). Elle contient également des informations concernant la partie utilisation de la session, incluant les références d'interface pour une interaction avec la session.

L'élément id est l'identificateur de cette session. Il désigne de manière non ambiguë cette session parmi toutes les sessions avec lesquelles le consommateur interagit par l'intermédiaire de ce revendeur (ce qui signifie que le consommateur peut éventuellement recevoir en retour des valeurs d'identificateur t_SessionId identiques lorsqu'il interagit simultanément avec plusieurs revendeurs).

L'élément purpose est une chaîne qui indique l'objet de la session. Le contenu de cette chaîne peut être défini au moment de la création de la session ou ultérieurement par des interactions propres au service.

L'élément secretId est un identificateur qui doit être utilisé par le consommateur lorsqu'il interagit sur des interfaces vers la session qui ont été définies par le modèle de session SPFEE (prière de se référer à la partie "utilisation" du point Ret-RP pour plus de détails).

L'élément myPartyId est l'identificateur de participant de ce consommateur. Si la session utilise le modèle de session SPFEE avec l'ensemble de fonctionnalités de participants multiples, cet identificateur est alors utilisé pour désigner ce participant. Les identificateurs t_PartyId des autres participants de la session sont également disponibles par le biais des interfaces d'ensemble de fonctionnalités de participant multiple.

L'élément state contient l'état de la session tel qu'il est perçu par ce consommateur. L'état peut prendre l'une des valeurs UserUnknownSessionState, UserActiveSession, UserSuspendedSession, UserSuspendedParticipation, UserInvited ou UserNotParticipating. Il sera vraisemblablement égal à UserActiveSession du fait que la session vient de démarrer.

L'élément itfs est une liste de types et de références d'interfaces prises en charge par la session (la liste peut contenir des interfaces propres au service permettant au consommateur d'interagir avec la session).

L'élément sessionModels est une liste des modèles de session et des ensembles de fonctionnalités pris en charge par la session. La liste peut contenir des références d'interface vers des interfaces prises en charge pour chacun des ensembles de fonctionnalités.

L'élément properties est une liste des propriétés de la session. Son utilisation est propre au revendeur.

8.5 Définitions d'interface d'accès: interfaces de domaine consommateur

Le présent sous-paragraphe décrit en détail les interfaces consommateur prises en charge au niveau du point Ret-RP. Il définit chacune des interfaces et les opérations qu'elle prend en charge.

Un grand nombre de ces opérations sont héritées d'autres interfaces. Elles sont toutefois décrites ici comme si elles étaient définies au niveau du point Ret-RP. Les interfaces décrites ici sont les seules qui doivent être prises en charge au niveau du point Ret-RP. Il n'est pas nécessaire de prendre en charge une hiérarchie d'héritage identique à celle qui a été définie précédemment au 8.2.

Les interfaces suivantes prises en charge par le domaine consommateur sont disponibles au niveau du point Ret-RP:

– i_ConsumerInitial
– i_ConsumerAccess
– i_ConsumerInvite
– i_ConsumerTerminal
– i_ConsumerSessionInfo
– i_ConsumerAccessSessionInfo

8.5.1 Interface i_ConsumerInitial

```
// module SPFEERetConsumerInitial
interface i_ConsumerInitial  :
SPFEEUserInitial::i_UserInitial
{
};
```

Cette interface permet à un revendeur d'initier une session d'accès avec le consommateur et au consommateur de recevoir des invitations en dehors d'une session d'accès.

L'objet de cette interface est de fournir un point de contact initial pour un revendeur qui souhaite contracter le consommateur (comme pour l'interface i_RetailerInitial). Elle est toutefois disponible pour un revendeur uniquement si elle a été immatriculée par le consommateur pour une utilisation en dehors d'une session d'accès en invoquant l'opération registerInterfaceOutsideAccessSession sur l'interface i_RetailerNamedAccess.

Les opérations décrites ci-dessous sont héritées par cette interface à partir de l'interface i_UserInitial prenant en charge les rôles génériques utilisateur-fournisseur. Aucune spécialisation propre au point Ret-RP n'est définie pour cette interface.

Les signatures d'opération suivantes sont fournies par le module SPFEEUserInitial. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEUserInitial:: lorsqu'ils sont utilisés par des clients de l'interface i_ConsumerInitial.

8.5.1.1 Opération requestAccess()

```
void requestAccess (
      in t_ProviderId providerId,
      out t_AccessReply reply
);
```

Projet de définition: cette opération est à l'état de projet.

Cette opération permet au revendeur de demander l'établissement d'une session d'accès entre le consommateur et le revendeur.

Elle permet uniquement au revendeur de formuler la demande sans établir effectivement la session. L'établissement nécessite que le consommateur contacte le revendeur par le biais de l'interface i_RetailerInitial et demande lui-même l'établissement de la session d'accès.

Le revendeur fournit au consommateur son identité t_ProviderId qui est utilisée par ce dernier pour contacter le fournisseur et obtenir une référence vers une interface i_RetailerInitial.

Le paramètre t_AccessReply permet au consommateur de fournir au revendeur des informations concernant l'action qu'il va effectuer en réponse à la demande. Les codes de réponse suivants sont définis:

– SUCCESS – le consommateur accepte d'établir une session d'accès. (Le consommateur établira la session d'accès comme décrit ci-dessus.)

– DECLINE – le consommateur refuse d'initier une session d'accès.

– FAILED – le consommateur n'est pas en mesure d'établir une session d'accès.

– FORBIDDEN – le domaine consommateur n'a pas l'autorisation d'accepter la demande.

8.5.1.2 Opération inviteUserAccessSession()

```
void inviteUserOutsideAccessSession (
      in t_ProviderId providerId,
      in SPFEEAccessCommonTypes::t_SessionInvitation invitation,
      out SPFEECommonTypes::t_InvitationReply reply
);
```

Projet de définition: cette opération est à l'état de projet.

Les paramètres t_SessionInvitation et t_InvitationReply sont définis conformément au projet de Norme "Protocole d'initiation de session" du sous-groupe MMUSIC (*multimedia multiparty session control*) du groupe de travail d'ingénierie Internet.

Cette opération permet à un revendeur d'envoyer une invitation à se joindre à une session de service destinée à un consommateur qui n'est pas impliqué dans une session d'accès.

Cette opération est utilisée lorsque le consommateur a immatriculé au préalable cette interface pour une utilisation en dehors d'une session d'accès et si le consommateur n'est pas engagé actuellement dans une session d'accès. Si le consommateur est engagé dans une session d'accès avec ce revendeur, les invitations ne doivent pas être faites par le biais de cette opération mais doivent être fournies à l'interface i_ConsumerAccess impliquée dans la session d'accès.

Le consommateur doit établir une session d'accès avec le revendeur pour pouvoir se joindre à la session de service décrite dans l'invitation; il utilise pour ce faire l'opération joinSessionWithInvitation() sur l'interface i_RetailerNamedAccess. L'entrée dans la session de service ne peut pas se faire en l'absence d'une session d'accès avec le revendeur.

Le paramètre t_ProviderId identifie le revendeur vis-à-vis du consommateur.

La structure t_SessionInvitation décrit la session de service dans laquelle le consommateur a été invité et fournit un élément t_InvitationId lui permettant d'identifier cette invitation au moment où il se joint à la session (aucune référence n'est fournie au sujet des interfaces vers la session, ni aucune information qui permettrait au consommateur de se joindre à la session sans établir au préalable une session d'accès avec ce revendeur). La structure fournit également un élément t_UserId contenant l'identificateur de l'utilisateur invité. Le domaine consommateur peut vérifier que l'invitation est destinée à un utilisateur final connu du domaine (prière de se référer au paragraphe "Invitations et annonces" pour plus de détails).

Le consommateur renvoie une réponse t_InvitationReply lui permettant fournir au revendeur des informations concernant l'action qu'il va effectuer pour l'invitation (prière de se référer au paragraphe "Invitations et annonces" pour plus de détails).

8.5.1.3 Opération cancelInviteUserOutsideAccessSession()

```
void cancelInviteUserOutsideAccessSession (
      in t_ProviderId providerId,
      in SPFEEAccessCommonTypes::t_InvitationId id
) raises (
      SPFEEAccessCommonTypes::e_InvitationError
);
```

Projet de définition: cette opération est à l'état de projet.

Les paramètres t_SessionInvitation et t_InvitationReply sont définis conformément au projet de Norme "Protocole d'initiation de session" du sous-groupe MMUSIC (*multimedia multiparty session control*) du groupe de travail d'ingénierie Internet.

Cette opération permet à un revendeur d'annuler une invitation à se joindre à une session de service qui a été envoyée à un consommateur. Elle peut être utilisée pour annuler des invitations qui ont été envoyées durant une session d'accès (par le biais de l'opération inviteUser() sur l'interface i_ConsumerInvite) et en dehors d'une session d'accès (par le biais de l'opération inviteUserOutsideAccessSession sur l'interface i_ConsumerInitial).

Le paramètre t_ProviderId identifie le revendeur vis-à-vis du consommateur.

Le paramètre t_InvitationId est utilisé avec l'élément t_ProviderId pour déterminer l'invitation qui doit être annulée (les identificateurs t_InvitationId sont non ambigus pour un consommateur donné; un consommateur peut recevoir des invitations contenant le même identificateur en provenance de plusieurs revendeurs).

Si le consommateur ne connaît pas la liste t_InvitationId, l'opération doit alors activer l'exception e_InvitationError avec le code d'erreur InvalidInvitationId.

8.5.2 Interface i_ConsumerAccess

```
// module SPFEERetConsumerAccess
interface i_ConsumerAccess : SPFEEUserAccess::i_UserAccess
{
};
```

Cette interface permet au revendeur d'accéder au domaine consommateur durant une session d'accès. Elle lui fournit des opérations qui lui permettent de demander des références vers des interfaces prises en charge par le domaine consommateur. Ces interfaces englobent celles qui sont définies au niveau du point Ret-RP ainsi que d'autres, propres au revendeur.

Son objet est similaire à celui de l'interface i_RetailerAccess du fait qu'elle est disponible durant la session d'accès. Elle est fournie au domaine revendeur dans le cadre de l'opération setUserCtxt() sur l'interface i_RetailerNamedAccess.

Les opérations décrites ci-dessous sont héritées de l'interface i_UserAccess prenant en charge les rôles génériques utilisateur-fournisseur. Aucune spécialisation propre au point Ret-RP n'est définie pour cette interface.

Les signatures d'opération suivantes sont fournies par le module SPFEEUserAccess. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEUserAccess:: lorsqu'ils sont utilisés par des clients de l'interface i_ConsumerAccess.

8.5.2.1 Opération cancelAccessSession()

```
void cancelAccessSession(
      in t_CancelAccessSessionProperties options
);
```

Projet de définition: cette opération est à l'état de projet.

L'opération cancelAccessSession() permet au revendeur de mettre fin à une session d'accès avec le consommateur. Le revendeur peut utiliser cette opération pour mettre fin à une session d'accès sans l'autorisation du consommateur.

La relation sécurisée et fiable entre le consommateur et le revendeur se termine lorsque cette opération est invoquée. Aucune des interfaces du côté consommateur et revendeur disponibles durant la session d'accès ne peuvent plus être utilisées pour des demandes (les interfaces qui ont été immatriculées pour une utilisation en dehors d'une session d'accès restent utilisables).

Le paramètre options contient une liste de propriétés qui décrit des options propres au revendeur ou des actions effectuées par le revendeur lorsqu'il met fin à la session d'accès (le revendeur peut, par exemple, suspendre la participation du consommateur dans ses sessions de service actives). Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_CancelAccessSessionProperties, de sorte que son utilisation est propre au revendeur.

Cette opération n'affecte aucune des relations contractuelles entre le consommateur et le revendeur. Le consommateur conserve la faculté de demander l'établissement d'une session d'accès et les autres sessions d'accès se poursuivent.

8.5.2.2 Opération getInterfaceTypes()

```
void getInterfaceTypes (
      out SPFEECommonTypes::t_InterfaceTypeList itfTypes
) raises (
      SPFEECommonTypes::e_ListError
);
```

Cette opération renvoie une liste des types des interfaces prises en charge par le domaine consommateur.

Le paramètre itfTypes contient la liste de tous les types des interfaces prises en charge par le domaine consommateur. Il se constitue d'une sequence de noms t_InterfaceTypeName, qui sont des noms représentant les types des interfaces prises en charge par le consommateur. Le paramètre itfTypes doit fournir tous les types des interfaces pouvant être prises en charge par le consommateur.

L'opération activera l'exception e_ListError avec le code d'erreur ListUnavailable si la liste itfTypes n'est pas disponible parce que les types des interfaces prises en charge par la session ne sont pas connus.

8.5.2.3 Opération getInterface()

```
void getInterface (
      in SPFEECommonTypes::t_InterfaceTypeName itfType,
      in SPFEECommonTypes::t_MatchProperties desiredProperties,
      out SPFEECommonTypes::t_InterfaceStruct itf
) raises (
```

```
                SPFEECommonTypes::e_InterfacesError,
                SPFEECommonTypes::e_PropertyError
);
```

Cette opération renvoie une interface du type demandé, prise en charge par le domaine consommateur.

Le paramètre type identifie le type d'interface de la référence d'interface devant être renvoyée.

Le paramètre desiredProperties peut être utilisé pour indiquer l'interface devant être renvoyée. Le type t_MatchProperties indique les propriétés que doivent posséder les sessions. Il indique également si cette correspondance doit porter sur une, toute ou aucune des propriétés. Aucune valeur de propriété ni aucun nom n'ont été définis à l'heure actuelle pour l'interface au niveau du point Ret-RP, de sorte que l'utilisation de ce paramètre est propre au revendeur.

Le paramètre itf renvoyé par cette opération contient le nom t_InterfaceTypeName, une référence d'interface (t_IntRef) et les propriétés d'interface (t_InterfaceProperties) du type d'interface demandé.

Si le consommateur ne prend pas en charge des interfaces correspondant au paramètre type, l'opération doit alors activer l'exception e_InterfacesError avec le code d'erreur InvalidInterfaceType.

L'opération doit activer l'exception e_PropertyError si la demande porte sur une propriété non valide.

8.5.2.4 Opération getInterfaces()

```
void getInterfaces (
       out SPFEECommonTypes::t_InterfaceList itfs
) raises (
       SPFEECommonTypes::e_ListError
);
```

Cette opération renvoie la liste des interfaces prises en charge par le consommateur.

Le paramètre itfs renvoyé par cette opération se constitue d'une sequence de structures t_InterfaceStruct contenant, pour chaque interface, le nom t_InterfaceTypeName, une référence d'interface (t_IntRef) et les propriétés d'interface (t_InterfaceProperties).

L'opération doit activer l'exception e_ListError en cas d'impossibilité ou de refus de renvoi des interfaces.

8.5.3 i_ConsumerInvite Interface

```
// module SPFEERetConsumerAccess
interface i_ConsumerInvite
       : SPFEEUserAccess::i_UserInvite
{
};
```

Cette interface permet au revendeur d'envoyer au consommateur une invitation à se joindre à une session de service. Elle est utilisable uniquement durant une session d'accès. Elle est transmise au domaine revendeur dans le cadre de l'opération setUserCtxt() sur l'interface i_RetailerNamedAccess. Si le consommateur souhaite recevoir des invitations en dehors d'une session d'accès, il doit alors immatriculer l'interface i_ConsumerInitial.

Les opérations décrites ci-dessous sont héritées de l'interface i_UserInvite prenant en charge les rôles génériques utilisateur-fournisseur. Aucune spécialisation propre au point Ret-RP n'est définie pour cette interface.

Les signatures d'opération suivantes sont fournies par le module SPFEEUserAccess. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEUserAccess:: lorsqu'ils sont utilisés par des clients de l'interface i_ConsumerInvite.

8.5.3.1 Opération inviteUser()

```
void inviteUser (
      in  SPFEEAccessCommonTypes::t_SessionInvitation invitation,
      out SPFEECommonTypes::t_InvitationReply reply
) raises (
      SPFEEAccessCommonTypes::e_InvitationError
);
```

Projet de définition: cette opération est à l'état de projet. Les paramètres t_SessionInvitation et t_InvitationReply sont définis conformément au projet de Norme "Protocole d'initiation de session" du sous-groupe MMUSIC (*multimedia multiparty session control*) du groupe de travail d'ingénierie Internet.

Cette opération permet à un revendeur d'inviter le consommateur à se joindre à une session de service. Elle peut être utilisée uniquement durant une session d'accès.

Le paramètre t_SessionInvitation décrit la session de service dans laquelle le consommateur a été invité et fournit un identificateur t_InvitationId permettant d'identifier cette invitation au moment où le consommateur se joint à la session (aucune référence n'est fournie au sujet des interfaces vers la session, ni aucune information qui permettrait au consommateur de se joindre à la session en dehors d'une session d'accès avec ce revendeur).

Un paramètre t_InvitationReply est renvoyé pour permettre au consommateur de fournir au revendeur des informations concernant les actions qu'il va effectuer pour l'invitation (prière de se référer au paragraphe "Invitations et annonces" pour plus de détails).

Le consommateur peut se joindre, à partir de cette session d'accès, à la session de service décrite par l'invitation ou il peut établir une nouvelle session d'accès avec ce revendeur. Un même identificateur t_InvitationId fera référence à cette invitation dans les deux sessions d'accès. Le consommateur doit utiliser l'opération joinSessionWithInvitation() sur l'interface i_RetailerNamedAccess. L'entrée dans la session de service ne peut pas se faire en l'absence d'une session d'accès avec le revendeur.

8.5.3.2 Opération cancelInviteUser()

```
void cancelInviteUser (
      in SPFEECommonTypes::t_UserId               inviteeId,
      in SPFEEAccessCommonTypes::t_InvitationId   id
) raises (
      SPFEEAccessCommonTypes::e_InvitationError
);
```

Projet de définition: cette opération est à l'état de projet. Les paramètres t_SessionInvitation et t_InvitationReply sont définis conformément au projet de Norme "Protocole d'initiation de session" du sous-groupe MMUSIC (*multimedia multiparty session control*) du groupe de travail d'ingénierie Internet.

Cette opération permet à un revendeur d'annuler une invitation à se joindre à une session de service qui a été envoyée à un consommateur. Elle peut être utilisée pour annuler des invitations qui ont été émises au cours d'une session d'accès (par le biais de l'opération inviteUser() sur l'interface i_ConsumerInvite) et en dehors d'une session d'accès (par le biais de l'opération inviteUserOutsideAccessSession() sur l'interface i_ConsumerInitial).

Le paramètre t_InvitationId est utilisé avec l'indentificateur t_ProviderId qui indique l'invitation devant être annulée (les identificateurs t_InvitationId sont non ambigus pour toutes les sessions d'accès avec un même revendeur).

Si le consommateur ne connaît pas la liste t_InvitationId, l'opération doit alors activer l'exception e_InvitationError avec le code d'erreur InvalidInvitationId (il est possible qu'une opération cancelInviteUser soit reçue avant l'opération inviteUser, en particulier si l'annulation est envoyée

immédiatement après l'établissement de la session d'accès; cette opération doit en tout cas activer l'exception).

8.5.4 Interface i_ConsumerTerminal

```
// module SPFEERetConsumerAccess
interface i_ConsumerTerminal
       : SPFEEUserAccess::i_UserTerminal
{
};
```

Cette interface permet au revendeur d'obtenir des informations concernant la configuration du terminal du domaine du consommateur et les applications. Elle est transmise au domaine revendeur dans le cadre de l'opération setUserCtxt() sur l'interface i_RetailerNamedAccess. Si le consommateur souhaite permettre au revendeur d'accéder aux informations en dehors d'une session d'accès, il doit alors immatriculer cette interface par le biais de l'opération registerInterfaceOutsideAccessSession() sur l'interface i_RetailerNamedAccess.

Projet de définition: cette interface est à l'état de projet. Elle peut en particulier être étendue pour permettre à un revendeur d'effectuer des demandes plus spécifiques concernant le domaine consommateur.

Les opérations décrites ci-dessous sont héritées de l'interface i_UserTerminal prenant en charge les rôles génériques utilisateur-fournisseur. Aucune spécialisation propre au point Ret-RP n'est définie pour cette interface.

Les signatures d'opération suivantes sont fournies par le module SPFEEUserAccess. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEUserAccess:: lorsqu'ils sont utilisés par des clients de l'interface i_ConsumerTerminal.

8.5.4.1 Opération getTerminalInfo()

```
void getTerminalInfo(
       out SPFEEAccessCommonTypes::t_TerminalInfo terminalInfo
);
```

Projet de définition: cette opération est à l'état de projet.

Elle permet au revendeur d'obtenir les informations concernant la configuration du terminal du domaine du consommateur auquel ce dernier souhaite permettre l'accès au revendeur.

L'opération renvoie la structure t_TerminalInfo qui fournit des détails concernant les types de terminal, le système d'exploitation, etc. Voir le 8.3.3 "Informations de contexte utilisateur".

8.5.5 Interface i_ConsumerAccessSessionInfo

```
// module SPFEERetConsumerAccess
interface i_ConsumerAccessSessionInfo
       : SPFEEUserAccess::i_UserAccessSessionInfo
{
};
```

Cette interface permet au revendeur de fournir au consommateur des informations concernant des changements d'état dans d'autres sessions d'accès avec le consommateur (par exemple, la création ou la suppression de sessions d'accès avec le même consommateur). Le consommateur reçoit uniquement les informations concernant les sessions d'accès dans lesquelles il est impliqué.

Cette interface n'est PAS transmise automatiquement au fournisseur dans le cadre de l'opération setUserCtxt() sur l'interface i_RetailerNamedAccess. Si le consommateur souhaite être informé des changements dans une autre session d'accès, il doit alors immatriculer cette interface par le biais de l'opération registerInterface() sur l'interface i_RetailerNamedAccess. Le revendeur fournira dans ce

cas au consommateur des indications concernant les changements dans la session d'accès jusqu'au moment où il est mis fin à l'immatriculation de cette interface ou à la session d'accès en cours.

Si le consommateur souhaite obtenir des informations de changement dans une session d'accès en dehors d'une session d'accès, il doit alors immatriculer cette interface par le biais de l'opération registerInterfaceOutsideAccessSession() sur l'interface i_RetailerNamedAccess. Les opérations n'utilisent pas d'identificateur t_ProviderId de sorte que, si cette interface est immatriculée pour une utilisation en dehors d'une session d'accès, une interface distincte doit alors être immatriculée auprès de chaque revendeur. Les revendeurs ne peuvent pas partager cette interface parce que l'identificateur est non ambigu pour ce consommateur uniquement pour un revendeur donné.

Les opérations décrites ci-dessous sont héritées de l'interface i_UserAccessSessionInfo prenant en charge les rôles génériques utilisateur-fournisseur. Aucune spécialisation propre au point Ret-RP n'est définie pour cette interface.

Les signatures d'opération suivantes sont fournies par le module SPFEEUserAccess. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEUserAccess:: lorsqu'ils sont utilisés par des clients de l'interface i_ConsumerAccessSessionInfo.

8.5.5.1 Opération newAccessSessionInfo()

```
oneway void newAccessSessionInfo (
       in SPFEEAccessCommonTypes::t_AccessSessionInfo accessSession
   );
```

Cette opération est utilisée pour indiquer au consommateur l'établissement d'une nouvelle session d'accès.

Le paramètre t_AccessSessionInfo contient l'identificateur t_AccessSessionId de la nouvelle session d'accès, le nom t_UserCtxtName permettant au consommateur d'identifier le domaine consommateur ou le terminal avec lequel la session d'accès a été établie, ainsi qu'une liste t_AccessSessionProperties de propriétés propres au revendeur qui peut être utilisée pour fournir des informations supplémentaires concernant la session d'accès.

8.5.5.2 Opération endAccessSessionInfo()

```
oneway void endAccessSessionInfo (
       in SPFEEAccessCommonTypes::t_AccessSessionId asId
   );
```

Cette opération est utilisée pour indiquer au consommateur la fin d'une session d'accès.

L'identificateur t_AccessSessionId indique la session d'accès qui s'est terminée.

8.5.5.3 Opération cancelAccessSessionInfo()

```
oneway void cancelAccessSessionInfo (
       in SPFEEAccessCommonTypes::t_AccessSessionId asId
   );
```

Cette opération est utilisée pour indiquer au consommateur qu'une une session d'accès a été supprimée par le revendeur. Voir le 8.5.2.1 "Opération cancelAccessSession()" pour plus de détails. Le paramètre t_AccessSessionId indique la session d'accès qui a été supprimée.

8.5.5.4 Opération newSubscribedServicesInfo()

```
oneway void newSubscribedServicesInfo (
       in SPFEEAccessCommonTypes::t_ServiceList services
   );
```

Cette opération est utilisée pour indiquer au consommateur qu'il a fait l'objet d'un abonnement à de nouveaux services (cet abonnement a pu être souscrit par un service de cette session d'accès, par une autre session d'accès ou par un consommateur qui a abonné ses utilisateurs à un nouveau service).

Le paramètre t_ServiceList contient, sous la forme d'une séquence de structures t_ServiceInfo, les services auxquels l'utilisateur est abonné. Prière de se référer au paragraphe "Informations de service et de session".

8.5.6 Interface i_ConsumerSessionInfo

```
// module SPFEERetConsumerAccess
interface i_ConsumerSessionInfo
       : SPFEEUserAccess::i_UserSessionInfo
{
};
```

Cette interface permet au revendeur de fournir au consommateur des informations concernant des changements d'état des sessions de service dans lesquelles ce consommateur est impliqué. Les opérations d'information sont invoquées chaque fois qu'un changement dans une session de service affecte le consommateur (c'est-à-dire lorsque la session est suspendue), mais ne le sont pas lorsque le changement n'affecte pas le consommateur (c'est-à-dire, lorsqu'un autre participant quitte la session). Cette interface est informée des changements dans toutes les sessions impliquant le consommateur qu'elles soient associées ou non à cette session d'accès.

Cette interface peut être communiquée au revendeur dans le cadre de l'opération setUserCtxt() sur l'interface i_RetailerNamedAccess. Il n'est pas nécessaire de communiquer cette interface dans l'opération setUserCtxt() si le consommateur ne souhaite pas être informé des modifications de ses sessions de service (le consommateur peut, dans ce cas, immatriculer par la suite cette interface par le biais de l'opération registerInterface() sur l'interface i_RetailerNamedAccess. Le revendeur informera alors le consommateur des modifications des sessions de service jusqu'à ce qu'il mette fin à l'immatriculation de cette interface ou jusqu'à la fin de la session d'accès en cours).

Le consommateur doit immatriculer cette interface par le biais de l'opération registerInterfaceOutsideAccessSession() sur l'interface i_RetailerNamedAccess s'il souhaite être informé des modifications de la session de service en dehors d'une session d'accès. Les opérations n'utilisent pas de paramètre t_ProviderId, de sorte que si cette interface est immatriculée pour une utilisation en dehors d'une session d'accès, une interface distincte doit être immatriculée auprès de chaque revendeur. Les revendeurs ne peuvent pas partager cette interface parce que l'identificateur t_SessionId est non ambigu, pour ce consommateur, uniquement au sein d'un fournisseur donné.

Cette interface hérite les opérations décrites ci-dessous de i_UserSessionInfo prenant en charge les rôles génériques utilisateur-fournisseur. Aucune spécialisation propre au point Ret-RP n'est définie pour cette interface.

Les opérations suivantes sont invoquées lorsqu'une session de service effectue une action concernant ce consommateur. Ces opérations permettent à la session d'accès de mettre à jour plus aisément sa connaissance de l'implication du consommateur dans des sessions de service. Elles concernent des événements qui sont en fait propres à l'utilisation (propres au service) mais pour lesquels on considère qu'elles ont un caractère suffisamment général pour qu'il soit utile de les communiquer à la session d'accès.

Seules des actions associées à ce consommateur donnent lieu à des opérations d'information, ce qui signifie qu'un consommateur A reçoit une invocation de l'opération endMyParticipationInfo() s'il met fin à sa participation dans une session, mais ne reçoit aucune information si un autre consommateur B met fin à sa propre participation. Le consommateur A reçoit par contre une information si c'est le consommateur B qui met fin à la participation du consommateur A.

Toutes les interfaces i_ConsumerSessionInfo reçoivent des invocations d'information lorsqu'une action est effectuée dans une session de service. Une de ces interfaces sera immatriculée par chacune

des sessions d'accès. Toutes les interfaces i_ConsumerSessionInfo recevront une invocation d'information quelle que soit la session d'accès dans laquelle la session de service est utilisée.

Les signatures d'opération suivantes sont fournies par le module SPFEEUserAccess. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEUserAccess:: lorsqu'ils sont utilisés par des clients de l'interface i_ConsumerSessionInfo.

```
oneway void newSessionInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );
```

– Le consommateur a démarré une nouvelle session service; le paramètre session contient des informations concernant cette session.

```
oneway void endSessionInfo (
       in SPFEECommonTypes::t_SessionId sessionId
);
```

– Une session de service s'est terminée; le paramètre sessionId identifie cette session.

```
oneway void endMyParticipationInfo (
       in SPFEECommonTypes::t_SessionId sessionId
);
```

– La participation du consommateur dans une session de service s'est terminée; le paramètre sessionId identifie cette session.

```
oneway void suspendSessionInfo (
       in SPFEECommonTypes::t_SessionId sessionId
   );
```

– Une session de service a fait l'objet d'une suspension; le paramètre sessionId identifie cette session.

```
oneway void suspendMyParticipationInfo (
       in SPFEECommonTypes::t_SessionId sessionId
   );
```

– La participation du consommateur dans une session de service a fait l'objet d'une suspension; le paramètre sessionId identifie cette session.

```
oneway void resumeSessionInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );
```

– Une session de service suspendue a fait l'objet d'une reprise; le paramètre sessionId identifie cette session (le consommateur peut ou non s'être joint à la session de service, selon que la reprise de la session a été effectuée par lui-même ou par un autre consommateur). Le paramètre session contient des informations concernant la session qui a fait l'objet d'une reprise.

```
oneway void resumeMyParticipationInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );
```

– La participation du consommateur dans une session de service a fait l'objet d'une reprise. Le paramètre session contient des informations concernant la session dans laquelle le consommateur a repris sa participation.

```
oneway void joinSessionInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );
```

– Le consommateur s'est joint à une session de service. Le paramètre session contient des informations concernant la session concernée.

8.6 Définitions d'interface d'accès: interfaces du domaine revendeur

Le présent sous-paragraphe décrit en détail toutes les interfaces revendeur prises en charge au niveau du point Ret-RP. Il définit chacune des interfaces et les opérations qu'elle prend en charge.

Un grand nombre de ces opérations sont héritées d'autres interfaces. Elles sont toutefois décrites ici comme si elles étaient définies au niveau du point Ret-RP. Les interfaces décrites ici sont les seules qui doivent être prises en charge au niveau du point Ret-RP. Il n'est pas nécessaire de prendre en charge une hiérarchie d'héritage identique à celle qui a été définie précédemment au 8.2.

Les interfaces suivantes prises en charge par le domaine consommateur sont disponibles au niveau du point Ret-RP.

8.6.1 Interface i_RetailerInitial

```
// module SPFEERetRetailerInitial
interface i_RetailerInitial:
      SPFEEProviderInitial::i_ProviderInitial
{
          // Inherited operations shown in following subsections.
};
```

L'interface i_RetailerInitial est le point de contact initial d'un consommateur avec le revendeur. Elle lui permet de demander l'établissement d'une session d'accès avec le revendeur.

Cette interface est renvoyée lorsque le consommateur prend contact avec le revendeur. La spécification du point Ret-RP ne précise pas la manière dont le consommateur prend contact avec le revendeur. Ceci peut se faire, par exemple, de l'une des manières suivantes: au moyen du service de dénomination de l'environnement DPE, au moyen d'un autre type de service d'annuaire tel qu'un courtier, au moyen du domaine commercial du courtier SPFEE avec utilisation du point de référence Bkr ou encore au moyen d'un localisateur URL avec utilisation de la page d'accueil du revendeur. Une interface de ce type est renvoyée au consommateur dans le cadre du scénario de contact du revendeur.

Cette interface hérite de l'interface i_ProviderInitial. Elle définit toutes les opérations qui ont un caractère général pour les rôles d'accès utilisateur-fournisseur et peut être réutilisée pour d'autres points de référence entre domaines.

Cette interface joue un rôle dans la sécurité et peut utiliser les procédures de sécurité de l'environnement DPE pour le chiffrement des messages et l'authentification des domaines. Ceci signifie que le transfert de messages par l'environnement DPE est protégé par chiffrement pour divers niveaux de sécurité et que les justificatifs des deux domaines sont échangés lors de l'authentification. Il n'existe toutefois aucune obligation d'utiliser l'environnement DPE pour l'authentification et l'échange de justificatifs, de sorte que l'interface i_RetailerAuthenticate permet d'effectuer l'authentification de l'utilisateur indépendamment de la sécurité de l'environnement DPE. Une référence vers l'interface i_RetailerAuthenticate est transmise au domaine consommateur par le biais des opérations requestNamedAccess() et requestAnonymousAccess() si l'utilisateur n'est pas authentifié par les procédures de sécurité de l'environnement DPE.

Les signatures d'opération suivantes sont fournies par le module SPFEEProviderInitial. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEProviderInitial:: lorsqu'ils sont utilisés par des clients de l'interface i_RetailerInitial.

8.6.1.1 Opération requestNamedAccess()

```
void requestNamedAccess (
      in SPFEECommonTypes::t_UserId userId,
      in SPFEECommonTypes::t_UserProperties userProperties,
      out Object  namedAccessIR,                  // type:
i_ProviderNamedAccess
      out SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
```

```
            out SPFEEAccessCommonTypes::t_AccessSessionId asId
        ) raises (
            e_AccessNotPossible,
            e_AuthenticationError,
            SPFEEAccessCommonTypes::e_UserPropertiesError
        );
```

L'opération requestNamedAccess() permet au consommateur de s'identifier et de demander l'établissement d'une session d'accès avec le revendeur. La session d'accès fournit, par le biais d'une interface i_RetailerNamedAccess, l'accès à l'utilisation des services auxquels il est abonné, etc.

Lorsque les services de sécurité CORBA sont utilisés par les environnements DPE du consommateur et du revendeur, les justificatifs des deux domaines et les autres informations d'authentification seront échangées par l'environnement DPE avant l'invocation de cette opération par le revendeur. Ceci signifie qu'un contexte sécurisé pour les messages peut déjà avoir été établi entre les domaines et que l'identité du consommateur aura été authentifiée. Dans un tel cas, une session d'accès est établie et une référence vers l'interface i_RetailerNamedAccess sera renvoyée. Un identificateur t_AccessSessionSecretId est renvoyé en outre à des fins d'utilisation dans toutes les demandes effectuées sur la nouvelle interface.

Aucun contexte sécurisé pour les messages n'est établi si les services de sécurité CORBA ne sont pas utilisés et les messages DPE peuvent éventuellement être interceptés et lus par des tiers.

L'opération échoue si le consommateur n'a pas déjà été authentifié et si l'environnement DPE n'est pas en mesure d'effectuer l'authentification et d'établir une session d'accès lors de l'invocation de cette opération. L'opération active une exception e_AuthenticationError avec une référence vers une interface i_RetailerAuthenticate. Cette interface peut être utilisée pour authentifier et établir le contexte sécurisé. Cette opération peut ensuite être utilisée de nouveau lors de l'établissement de la session d'accès.

Le paramètre userId identifie le consommateur vis-à-vis du revendeur. Prière de se référer au paragraphe "Informations utilisateur" pour des détails concernant la structure de l'identificateur userId.

Le paramètre userProperties contient une séquence de propriétés utilisateur associées à ce consommateur. Ce dernier n'enverra en général au revendeur aucune information sensible tant qu'une session d'accès n'a pas été établie. Ce paramètre peut toutefois être utilisé pour communiquer au revendeur le mot de passe du consommateur lorsque les deux domaines utilisent les procédures de sécurité de l'environnement DPE pour le chiffrement des messages. Il est possible de transmettre également le contexte de sécurité ainsi que d'autres informations qui sont comprises par le revendeur en question. Prière de se référer au paragraphe "Informations utilisateur" pour plus de détails.

Les paramètres de sortie suivants sont renvoyés si la demande réussit et si le consommateur a été authentifié:

Le paramètre namedAccessIR contient la référence vers l'interface i_RetailerNamedAccess utilisée par le domaine consommateur durant la session d'accès.

NOTE – Bien que la spécification IDL précise (dans le texte) le type d'interface i_ProviderNamedAccess, ceci ne concerne que le type de référence de base. Une interface abstraite (référence) n'est jamais exportée au niveau d'un point de référence. L'opération requestNamedAccess() est définie pour l'interface i_ProviderNamedAccess qui est ensuite héritée par l'interface i_RetailerNamedAccess et utilise de ce fait des interfaces (références) du type i_Retailer<...>. La raison de l'utilisation du type de référence de base dans la spécification IDL est d'en permettre la réutilisation pour des définitions d'autres points de référence. L'interface namedAccessIR pourrait, par exemple, être utilisée également pour un type i_3ptyNamedAccess.

Le paramètre asSecretId contient un identificateur t_AccessSessionSecretId utilisé lorsque le domaine consommateur invoque une opération sur l'interface namedAccessIR au sein de cette session d'accès. Le paramètre asSecretId identifie le domaine consommateur à partir duquel sont

faites les invocations sur l'interface namedAccessIR. Ce paramètre doit uniquement être utilisé durant cette session d'accès et uniquement par le domaine consommateur auquel il a été renvoyé. Voir 8.4.

Le paramètre asId contient un identificateur t_AccessSessionId utilisé pour cette session d'accès. Il est disponible pour toutes les sessions d'accès à ce consommateur. Il peut être utilisé pour identifier cette session d'accès lorsque des demandes sont faites sur toute interface i_RetailerNamedAccess entre ce consommateur et ce revendeur, par exemple au moyen de l'opération listServiceSessions(); un identificateur t_AccessSessionId peut être utilisé pour limiter cette liste aux sessions qui ont été démarrées à partir d'une session d'accès donnée.

Lorsque la demande est infructueuse cela indique que le consommateur n'a pas été authentifié ou que l'authentification a échoué.

L'opération active une exception e_AccessNotPossible si le revendeur n'est pas en mesure d'établir une telle session d'accès ou refuse de permettre au domaine consommateur d'établir cette session d'accès.

L'opération active une exception e_AuthenticationError si le revendeur n'a pas authentifié le consommateur. Elle contient une liste de méthodes d'authentification pouvant être utilisées avec l'interface i_RetailerAuthenticate. Cette interface est renvoyée, selon le revendeur, soit sous la forme d'une référence d'interface, soit sous la forme d'une référence d'objet contenue dans une chaîne. La référence est utilisée pour authentifier le consommateur vis-à-vis du revendeur. Après la réussite de cette authentification (en utilisant l'une des méthodes indiquées), le consommateur peut alors invoquer une nouvelle fois cette opération pour demander l'établissement d'une session d'accès et obtenir une référence vers l'interface i_RetailerNamedAccess.

Si l'exception e_UserPropertiesError est activée, ceci indique alors un problème concernant le paramètre userProperties. Le motif de l'erreur est indiqué dans le code errorCode.

8.6.1.2 Opération requestAnonymousAccess()

```
void requestNamedAccess (
       in SPFEECommonTypes::t_UserId userId,
       in SPFEECommonTypes::t_UserProperties userProperties,
       out Object  namedAccessIR,      // type: i_ProviderNamedAccess
       out SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       out SPFEEAccessCommonTypes::t_AccessSessionId asId
) raises (
       e_AccessNotPossible,
       e_AuthenticationError,
       SPFEEAccessCommonTypes::e_UserPropertiesError
);
```

L'opération requestAnonymousAccess() permet au consommateur de demander l'établissement d'une session d'accès avec le revendeur. Elle est utilisée lorsque le consommateur ne fournit pas au revendeur l'identité de l'utilisateur, soit parce qu'il n'a pas contacté ce revendeur au préalable, soit parce qu'il souhaite rester anonyme.

Cette opération renvoie une référence vers une interface i_RetailerAnonAccess pouvant être utilisée par le consommateur pour accéder à des services et s'immatriculer auprès du revendeur en tant qu'utilisateur nommé, s'il le souhaite.

Lorsque les services de sécurité CORBA sont utilisés par les environnements DPE du consommateur et du revendeur, les deux domaines peuvent alors échanger des justificatifs par le biais de ces environnements avant l'invocation de cette opération par le revendeur. Ceci signifie qu'un contexte sécurisé pour les messages peut déjà avoir été établi entre les domaines, mais les justificatifs ne contiendront aucune information concernant l'identité du consommateur en question.

Aucun contexte sécurisé pour les messages n'est établi si les services de sécurité CORBA ne sont pas utilisés et les messages DPE peuvent éventuellement être interceptés et lus par des tiers.

Le paramètre userProperties contient une séquence de propriétés utilisateur associées à ce consommateur et pouvant contenir un contexte de sécurité et d'autres informations qui sont comprises par le revendeur en question. Prière de se référer au paragraphe "Informations utilisateur" pour plus de détails.

Une session d'accès a été établie avec le consommateur si la demande réussit. Les paramètres de sortie suivants sont renvoyés:

Le paramètre anonAccessIR contient la référence vers une interface i_RetailerAnonAccess utilisée par le domaine consommateur pendant la session d'accès.

Le paramètre asSecretId contient un identificateur t_AccessSessionSecretId utilisé chaque fois que le domaine consommateur invoque une opération sur l'interface namedAccessIR au sein de cette session d'accès. Le paramètre asSecretId identifie le domaine consommateur à partir duquel ces invocations sont faites sur l'interface namedAccessIR. Ce paramètre doit uniquement être utilisé durant cette session d'accès et uniquement par le domaine consommateur auquel il a été renvoyé. Voir 8.4.

Le paramètre asId contient un identificateur t_AccessSessionId qui désigne cette session d'accès. Il est utilisable par toutes les sessions d'accès à ce consommateur. Il peut être utilisé pour identifier cette session d'accès pour toute demande faite sur l'interface i_RetailerNamedAccess dans une session d'accès entre ce consommateur et ce revendeur (des utilisateurs anonymes ne disposent en général que d'une seule session d'accès avec le revendeur, du fait que chaque session d'accès de chaque utilisateur anonyme doit être traitée de manière distincte. Comme les consommateurs ne sont pas connus du revendeur, chacun d'eux se présente comme un individu distinct, même s'il s'agit en réalité de la même personne).

En cas d'échec de la demande, le consommateur n'a pas été authentifié ou l'authentification a échoué.

L'opération active une exception e_AccessNotPossible si le revendeur n'est pas en mesure d'établir une telle session d'accès ou refuse de permettre au domaine consommateur d'établir cette session d'accès.

L'opération active une exception e_AuthenticationError si le revendeur exige l'authentification du domaine consommateur. Elle contient une liste de méthodes d'authentification pouvant être utilisées avec l'interface i_RetailerAuthenticate (ces méthodes peuvent authentifier uniquement les domaines et non le consommateur concerné). L'interface est renvoyée, selon le revendeur, soit sous la forme d'une référence d'interface, soit sous la forme d'une référence d'objet contenue dans une chaîne. Cette référence est utilisée pour authentifier le consommateur vis-à-vis du revendeur. Après la réussite de cette authentification (en utilisant l'une des méthodes indiquées), le consommateur peut alors invoquer une nouvelle fois cette opération pour demander l'établissement d'une session d'accès et obtenir une référence vers l'interface i_RetailerNamedAccess.

Si l'exception e_UserPropertiesError est activée, ceci indique alors un problème concernant le paramètre userProperties. Le motif de l'erreur est indiqué par le champ errorCode.

8.6.2 Interface i_RetailerAuthenticate

```
// module SPFEERetRetailerInitial
interface i_RetailerAuthenticate:
       SPFEEProviderInitial::i_ProviderAuthenticate
{
           // Inherited operations shown in following subsections.
};
```

L'interface i_RetailerAuthenticate permet l'authentification du consommateur et du revendeur. Elle fournit un procédé général d'authentification pouvant être utilisé pour la prise en charge de divers protocoles d'authentification.

Cette interface a pour objet de vérifier que le consommateur et le revendeur de chaque domaine interagissent avec le domaine pour lequel ils ont été autorisés, ce qui implique l'authentification mutuelle des deux domaines. L'utilisation de cette interface est également possible avec des procédés qui authentifient un seul des domaines. Cette interface fournit un ensemble d'opérations générales utilisables pour l'authentification. Ces opérations fournissent toutefois uniquement un mécanisme de "transport" des informations d'authentification. Les deux domaines doivent utiliser ces opérations pour mettre en œuvre un protocole d'authentification commun. La spécification du point Ret-RP ne recommande aucun protocole d'authentification particulier. L'opération getAuthenticationMethods() peut être utilisée sur cette interface pour déterminer les protocoles d'authentification pris en charge par le revendeur, dont l'un sera choisi pour l'authentification. Le protocole d'authentification peut identifier ou non le consommateur individuel. Il peut uniquement identifier et authentifier le domaine du consommateur.

Les signatures d'opération suivantes sont fournies par le module SPFEEProviderInitial. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEProviderInitial:: lorsqu'ils sont utilisés par des clients de l'interface i_RetailerInitial.

8.6.2.1 Opération getAuthenticationMethods()

```
void getAuthenticationMethods {
       in  t_AuthMethodSearchProperties desiredProperties,
       out t_AuthMethodDescList authMethods
) raises (
       e_AuthMethodPropertiesError,
       SPFEECommonTypes::e_ListError
);
```

L'opération getAuthenticationMethods() permet au consommateur de demander au revendeur une liste des méthodes d'authentification prises en charge. Une méthode d'authentification particulière peut ensuite être choisie par le consommateur pour une utilisation dans l'opération authenticate().

Le paramètre desiredProperties contient une liste des propriétés dont le consommateur souhaite la prise en charge par la méthode d'authentification (prière de se référer au type t_MatchProperties défini dans le paragraphe "Propriétés et listes de propriétés"). Le consommateur peut, par exemple, demander que les méthodes d'authentification renvoyées prennent en charge l'authentification mutuelle ou uniquement l'authentification du revendeur. Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_AuthMethodSearchProperties, de sorte que son utilisation est propre au revendeur.

Le paramètre authMethods contient une liste des méthodes d'authentification qui correspondent aux propriétés desiredProperties et qui sont prises en charge par le revendeur. La structure t_AuthMethodDesc contient l'identificateur de la méthode d'authentification et une liste de propriétés de la méthode. On fait l'hypothèse que le consommateur et le revendeur connaissent le protocole qui doit être utilisé pour la méthode d'authentification concernée.

La liste authMethods peut être vide. Ce cas peut se présenter si le revendeur ne prend en charge aucune méthode correspondant aux méthodes demandées ou s'il ne souhaite pas permettre au consommateur d'effectuer une authentification au moyen d'une méthode possédant les propriétés souhaitées, par exemple si le consommateur demande une méthode authentifiant uniquement le revendeur et si ce dernier souhaite une authentification mutuelle.

L'opération active une exception e_PropertyError si le paramètre desiredProperties n'a pas un format correct ou fournit un nom ou une valeur de propriété non valide (les noms de propriété non valides

peuvent être ignorés si le paramètre desiredProperties demande uniquement la concordance de certaines ou d'aucune des propriétés).

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste authMethods n'est pas disponible.

8.6.2.2 Opération authenticate()

```
void authenticate(
      in t_AuthMethod authMethod,
      in string securityName,
      in t_opaque authenData,
      in t_opaque privAttribReq,
      out t_opaque privAttrib,
      out t_opaque continuationData,
      out t_opaque authSpecificData,
      out t_AuthenticationStatus authStatus
) raises (
      e_AuthMethodNotSupported
);
```

L'opération authenticate() permet au consommateur de choisir une méthode d'authentification et de transmettre au revendeur des données d'authentification.

Cette opération est utilisée pour le transport vers le revendeur des données d'authentification et d'autres justificatifs une fois que le domaine consommateur a choisi une méthode d'authentification avec le revendeur. Ces données sont utilisées pour effectuer le type d'authentification convenant à la méthode d'authentification (il peut s'agir d'une authentification mutuelle ou d'une authentification concernant le domaine consommateur ou revendeur seulement, etc.).

Le revendeur peut ensuite renvoyer (si nécessaire) ses données d'authentification, des données de mise à l'épreuve auxquelles le consommateur doit répondre (si nécessaire) par le biais d'une opération continueAuthentication(), ainsi que (si possible) les justificatifs demandés. Si d'autres protocoles d'authentification sont requis avant le renvoi des justificatifs, ces derniers peuvent être renvoyés par le biais de l'opération continueAuthentication().

Les paramètres suivants sont renvoyés au revendeur par le consommateur:

Le paramètre authMethod identifie la méthode d'authentification proposée par le consommateur. Il influe sur la méthode de composition et de génération des autres paramètres de données opaques. Aucune valeur particulière de méthode d'authentification n'a été définie à l'heure actuelle pour la structure t_AuthMethod, de sorte que son utilisation est propre au revendeur.

Le paramètre securityName fournit le nom présumé du consommateur devant être authentifié. Il peut contenir une chaîne vide, selon la méthode d'authentification utilisée.

Le paramètre authenData fournit des données opaques contenant des attributs du consommateur devant être authentifié. Le format de ce paramètre dépend de la méthode d'authentification utilisée.

Le paramètre privAttribReq contient des données opaques utilisées pour spécifier les droits et les privilèges que le domaine consommateur demande au domaine revendeur. Ces données peuvent correspondre à des niveaux de sécurité pour l'accès à différentes régions du domaine revendeur. Le format de ce paramètre dépend de la méthode d'authentification utilisée.

Les paramètres suivants sont renvoyés au consommateur par le revendeur:

Le paramètre privAttrib contient des données opaques qui définissent les attributs de privilège accordés au consommateur en fonction de ses données d'authentification et du paramètre privAttribReq. Le format de ce paramètre dépend de la méthode d'authentification utilisée.

Le paramètre continuationData contient des données opaques utilisées pour la mise à l'épreuve du consommateur. Le consommateur n'a pas encore été authentifié; il doit traiter ces données et

renvoyer le résultat au revendeur par le biais de l'opération continueAuthentication(). Le format de ce paramètre dépend de la méthode d'authentification utilisée. Ce paramètre peut être ignoré si la valeur du paramètre authStatus n'est pas égale à SecAuthContinue.

Le paramètre authSpecificData contient des données opaques propres à la méthode d'authentification utilisée.

Le paramètre authStatus identifie le statut du processus d'authentification. Il contient un type énuméré pouvant prendre l'une des valeurs suivantes:

– SecAuthSuccess

  L'authentification s'est terminée correctement. Aucune autre invocation de l'opération continueAuthentication() n'est plus nécessaire. Le consommateur peut invoquer l'opération requestNamedAccess() sur l'interface i_RetailerInitial pour obtenir une référence vers l'interface i_RetailerNamedAccess (ou invoquer l'opération requestAnonymousAccess() sur l'interface i_RetailerAnonAccess s'il souhaite devenir un utilisateur anonyme).

– SecAuthFailure

  L'authentification s'est terminée de manière incorrecte. Le consommateur n'a pas été authentifié et ne sera pas en mesure d'établir une session d'accès. De nouvelles invocations de l'opération requestNamedAccess() activeront l'exception e_AccessNotPossible ou e_AuthenticationError.

– SecAuthContinue

  L'authentification se poursuit et le consommateur doit répondre par le biais de l'opération continueAuthentication().

– SecAuthExpired

  Un débordement de temporisation s'est manifesté en cours de l'authentification. Le consommateur n'a pas effectué l'invocation de l'opération continueAuthentication() en temps voulu après la réponse de l'opération authenticate() ou après l'invocation précédente de l'opération continueAuthentication(). L'authentification doit reprendre depuis le début par l'invocation de l'opération authenticate(). Cette valeur du type énuméré ne doit pas être renvoyée par l'opération authenticate().

8.6.2.3 Opération continueAuthentication()

```
void continueAuthentication{
      in t_opaque responseData,
      out t_opaque privAttrib,
      out t_opaque continuationData,
      out t_opaque authSpecificData,
      out t_AuthenticationStatus authStatus
);
```

L'opération continueAuthentication() permet au consommateur de poursuivre un protocole d'authentification, qui a été démarré par l'invocation de l'opération authenticate(), et de transmettre au revendeur des données d'authentification.

Cette opération doit être invoquée par le consommateur si le paramètre authStatus renvoyé par l'opération authenticate() ou par une invocation précédente de l'opération continueAuthentication() fournit la valeur SecAuthContinue. Les deux opérations utilisent le paramètre authStatus pour indiquer si le consommateur doit invoquer à nouveau cette opération. Les paramètres renvoyés par cette opération doivent être traités par le consommateur conformément à la méthode d'authentification et les résultats doivent être utilisés dans les paramètres de l'invocation suivante de cette opération.

Le paramètre responseData contient des données opaques fournies par le consommateur. Elles ont été générées par le consommateur conformément à la méthode d'authentification, sur la base du

paramètre continuationData renvoyé par l'invocation précédente de l'opération authenticate() ou continueAuthentication(). Les modalités précises de génération et de formatage de ces données sont propres à la méthode d'authentification utilisée.

Le paramètre continuationData contient des données opaques utilisées pour la mise à l'épreuve du consommateur. Ce dernier n'a pas encore été authentifié; il doit traiter ces données et renvoyer le résultat au revendeur par le biais de l'opération continueAuthentication(). Le format de ce paramètre dépend de la méthode d'authentification utilisée. Ce paramètre peut être ignoré si la valeur du paramètre authStatus n'est pas égale à SecAuthContinue.

Le paramètre authSpecificData contient des données opaques propres à la méthode d'authentification utilisée.

Le paramètre authStatus identifie le statut du processus d'authentification. Il possède les mêmes valeurs que pour l'opération authenticate().

8.6.3 Interface i_RetailerAccess

```
// module SPFEEProviderAccess
interface i_RetailerAccess
{
};
```

L'interface i_RetailerAccess est une interface abstraite utilisée pour fournir par héritage des opérations communes aux interfaces i_RetailerNamedAccess et i_RetailerAnonAccess.

Cette interface destinée à l'héritage ne doit pas être disponible au niveau du point Ret-RP et aucune instance ne doit être créée pour ce type d'interface.

Aucune opération n'est définie à l'heure actuelle pour cette interface. Elle contiendra des opérations partagées entre les interfaces i_RetailerNamedAccess et i_RetailerAnonAccess. A l'heure actuelle toutes les opérations sont définies pour l'interface i_RetailerNamedAccess et aucune opération n'est définie pour l'interface i_RetailerAnonAccess.

8.6.4 Interface i_RetailerNamedAccess

```
// module SPFEEProviderAccess
interface i_RetailerNamedAccess
      : i_ProviderNamedAccess, i_RetailerAccess
{
         // Inherited operations shown in following subsections.
};
```

L'interface i_RetailerNamedAccess permet à un consommateur connu d'accéder aux services auxquels il est abonné. Le consommateur l'utilise pour toutes les opérations au sein d'une session d'accès avec le revendeur.

Cette interface est renvoyée par l'invocation de l'opération requestNamedAccess() sur l'interface i_RetailerInitial une fois que le consommateur a été authentifié par le revendeur et qu'une session d'accès a été établie.

Cette interface hérite des interfaces i_ProviderNamedAccess et i_RetailerAccess. L'interface i_ProviderNamedAccess définit toutes les opérations qui ont un caractère général pour les rôles d'accès utilisateur-fournisseur et qui peuvent être réutilisées pour d'autres points de référence entre domaines. Toutes les opérations sur cette interface sont héritées à ce niveau. L'interface i_RetailerAccess est actuellement vide. Elle contiendra toutes les opérations qu'elle partage avec l'interface i_RetailerAnonAccess et qui sont propres au point Ret-RP.

Les signatures d'opération suivantes sont fournies par le module SPFEEProviderAccess. Tous les types non précisés doivent être qualifiés par l'identificateur SPFEEProviderAccess:: lorsqu'ils sont utilisés par des clients de l'interface i_RetailerAccess.

8.6.4.1 Opération setUserCtxt()

```
void setUserCtxt (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in t_UserCtxt userCtxt
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_UserCtxtError
);
```

L'opération setUserCtxt() permet au consommateur de fournir au revendeur des informations concernant des interfaces situées au sein du domaine consommateur ainsi qu'au sujet d'autres informations de ce domaine (par exemple, les applications utilisateur disponibles dans le domaine consommateur, le système d'exploitation utilisé, etc.).

Le paramètre userCtxt contient les informations de configuration et d'interfaces du domaine consommateur.

Cette opération doit être invoquée immédiatement après la réception de la référence vers cette interface. Si l'invocation de cette opération échoue, il est alors possible que les opérations suivantes activent l'exception e_AccessError avec un code d'erreur UserCtxtNotSet.

L'opération active une exception e_UserCtxtError avec le code d'erreur adéquat en cas de problème avec le paramètre userCtxt.

8.6.4.2 Opération getUserCtxt()

```
void getUserCtxt (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_UserCtxtName ctxtName,
       out t_UserCtxt userCtxt
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_UserCtxtError
);
```

Cette opération permet au consommateur d'extraire des informations concernant les contextes utilisateur qui ont été immatriculés auprès du revendeur.

Le paramètre ctxtName contient le nom du contexte dont le consommateur souhaite connaître les informations de contexte utilisateur (le paramètre ctxtName est positionné par le consommateur lorsqu'il enregistre un contexte utilisateur et désigne le terme qu'il utilise pour ce contexte, par exemple "domicile", "bureau", "le château de mon père", etc.

La structure userCtxt contient les informations de configuration et d'interfaces du domaine consommateur.

8.6.4.3 Opération getUserCtxts()

```
void getUserCtxts (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in t_SpecifiedUserCtxt ctxt,
       out t_UserCtxtList userCtxts
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_UserCtxtError,
       SPFEECommonTypes::e_ListError
);
```

8.6.4.4 Opération listAccessSessions()

```
void listAccessSessions (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      out SPFEEAccessCommonTypes::t_AccessSessionList asList
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEECommonTypes::e_ListError
);
```

L'opération listAccessSessions() renvoie la liste de toutes les sessions d'accès qui sont actuellement établies par le consommateur avec ce revendeur. Cette liste se présente sous la forme d'une sequence de structures t_AccessSessionInfo constituées des éléments t_AccessSessionId, t_UserCtxtName et t_AccessSessionProperties. Le dernier de ces éléments est une liste du type t_PropertyList. Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_AccessSessionProperties, de sorte que son utilisation est propre au revendeur.

Les informations renvoyées par cette opération peuvent être utilisées par le consommateur pour identifier les autres sessions d'accès qui sont actuellement établies, mettre fin à certaines de ces sessions d'accès (prière de se référer à l'opération endAccessSession()), établir la liste des sessions de service de ces sessions d'accès (prière de se référer à l'opération listServiceSessions()) et obtenir des informations concernant les modifications de ces sessions d'accès et sessions de service (prière de se référer aux interfaces i_ConsumerAccessSessionInfo et i_ConsumerSessionInfo).

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste asList n'est pas disponible en raison de l'indisponibilité des sessions d'accès du consommateur.

8.6.4.5 Opération endAccessSessions()

```
void endAccessSession(
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEEAccessCommonTypes::t_SpecifiedAccessSession as,
      in t_EndAccessSessionOption option
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEEAccessCommonTypes::e_SpecifiedAccessSessionError,
      e_EndAccessSessionError
);
```

L'opération endAccessSession() permet au consommateur de mettre fin à une session d'accès.

L'opération peut mettre fin à la session d'accès en cours, à une session d'accès donnée ou à toutes les sessions d'accès (y compris la session actuelle), en utilisant le paramètre t_SpecifiedAccessSession.

Le paramètre t_EndAccessSessionOptions permet au consommateur d'indiquer les actions devant être effectuées par le revendeur dans le cas où il reste des sessions de service actives ou suspendues lorsque la session se termine. Les actions sont utilisées uniquement dans le cadre de cette invocation. Le revendeur ne se souvient pas de l'action choisie (les revendeurs peuvent définir une politique par défaut utilisable pour des sessions de service lorsqu'un consommateur met fin à la session d'accès au sein de laquelle elles ont été créées; il peut également permettre au consommateur de définir cette politique. Le point Ret-RP ne prend pas actuellement en charge la définition d'une telle politique par le consommateur).

L'opération active une exception e_SpecifiedAccessSessionError si le paramètre as n'a pas un format correct ou fournit un identificateur de session d'accès non valide.

L'opération active une exception e_EndAccessSessionError si le paramètre option n'est pas valide ou s'il reste des sessions de service actives ou suspendues non autorisées par le revendeur (un consommateur peut mettre fin à une session d'accès au moment de quitter les sessions actives ou suspendues, si cela est autorisé par la politique du revendeur pour ce consommateur).

8.6.4.6 Opération getUserInfo()

```
void getUserInfo(
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      out SPFEEAccessCommonTypes::t_UserInfo userInfo
) raises (
      SPFEEAccessCommonTypes::e_AccessError
);
```

L'opération getUserInfo() permet au consommateur de demander des informations le concernant.

Cette opération renvoie comme paramètre en sortie une structure t_UserInfo contenant l'identité t_UserId du consommateur, son nom et une liste de propriétés utilisateur. Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_UserProperties, de sorte que son utilisation est propre au revendeur.

8.6.4.7 Opération listSubscribedServices()

```
void listSubscribedServices (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in t_SubscribedServiceProperties desiredProperties,
      out SPFEEAccessCommonTypes::t_ServiceList services
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEECommonTypes::e_PropertyError,
      SPFEECommonTypes::e_ListError
);
```

L'opération listSubscribedServices() renvoie une liste des services auxquels le consommateur a été abonné au préalable.

Le paramètre desiredProperties peut être utilisé pour limiter le contenu de la liste des services qui ont fait l'objet d'un abonnement. La liste t_SubscribedServiceProperties indique les propriétés qui doivent concorder avec celles des services de l'abonnement. Elle définit également si la concordance d'un tel service doit correspondre à une, à toutes ou à aucune des propriétés (prière de se référer à la description du type t_MatchProperties dans le paragraphe "Propriétés et listes de propriétés"). Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_SubscribedServiceProperties, de sorte que son utilisation est propre au revendeur.

La liste des services de l'abonnement du consommateur qui correspondent au paramètre desiredProperties est renvoyée dans le paramètre t_ServiceList. Elle se constitue d'une sequence de structures t_ServiceInfo contenant les éléments t_ServiceId, t_UserServiceName (nom des consommateurs pour le service) et une sequence de propriétés de service t_ServiceProperties. Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_ServiceProperties, de sorte que son utilisation est propre au revendeur.

L'opération active une exception desiredProperties si le paramètre desiredProperties n'a pas un format correct ou fournit un nom de propriété non valide (les noms de propriété non valides peuvent être ignorés si le paramètre desiredProperties demande uniquement la concordance de certaines ou d'aucune des propriétés).

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste services n'est pas disponible parce que les services du revendeur ne sont pas disponibles.

8.6.4.8 Opération discoverServices()

```
void discoverServices(
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in t_DiscoverServiceProperties desiredProperties,
      in unsigned long howMany,
      out SPFEEAccessCommonTypes::t_ServiceList services,
      out Object iteratorIR       // type: i_DiscoverServicesIterator
```

```
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       SPFEECommonTypes::e_PropertyError,
       SPFEECommonTypes::e_ListError
);
```

L'opération discoverServices() renvoie une liste des services fournis par ce revendeur.

Cette opération est utilisée pour trouver les services fournis par le revendeur à des fins de mise en œuvre par le consommateur. Elle peut être utilisée pour extraire des informations concernant tous les services ou ceux qui sont limités aux propriétés spécifiées par le paramètre desiredProperties (prière de se référer à la définition de la structure t_MatchProperties dans le paragraphe "Propriétés et listes de propriétés").

La liste de revendeurs de services correspondant au paramètre desiredProperties est renvoyée dans le paramètre services constitué d'une sequence de structures t_ServiceInfo contenant les éléments t_ServiceId, t_UserServiceName (nom des consommateurs pour le service) et une sequence de propriétés de service t_ServiceProperties. Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_ServiceProperties, de sorte que son utilisation est propre au revendeur.

Le paramètre howMany définit le nombre de structures t_ServiceInfo devant être renvoyées dans le paramètre services. La longueur de la liste services ne sera pas supérieure à ce nombre. Tout autre service correspondant au paramètre desiredProperties mais qui ne figure pas dans la liste services est accessible au moyen du paramètre iteratorIR sur l'interface i_DiscoverServicesIterator. Le paramètre iteratorIR doit être nul s'il n'existe pas d'autre service.

L'opération active une exception e_PropertyError si le paramètre desiredProperties n'a pas un format correct ou fournit un nom ou une valeur de propriété non valide (les noms de propriété non valides peuvent être ignorés si le paramètre desiredProperties demande uniquement la concordance de certaines ou d'aucune des propriétés).

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste services n'est pas disponible.

8.6.4.9 Opération getServiceInfo()

```
void getServiceInfo (
     in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
     in SPFEEAccessCommonTypes::t_ServiceId serviceId,
     in SPFEEProviderAccess::t_SubscribedServiceProperties
     desiredProperties,
     out SPFEEAccessCommonTypes::t_ServiceProperties serviceProperties
) raises (
     SPFEEAccessCommonTypes::e_AccessError,
     SPFEEProviderAccess::e_ServiceError
);
```

L'opération getServiceInfo() renvoie des informations concernant un service identifié par son identificateur de service. Le paramètre desiredProperties peut être utilisé pour préciser l'étendue des informations devant être renvoyées.

8.6.4.10 Opération listRequiredServiceComponents()

```
void listRequiredServiceComponents (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEEAccessCommonTypes::t_ServiceId serviceId,
       in SPFEEAccessCommonTypes::t_TerminalConfig terminalConfig,
       in SPFEEAccessCommonTypes::t_TerminalInfo terminalInfo,
       // Example of usage for Java applet download:
       // name-value pair describing the url of a Java applet
       // name = "URL"
```

```
        // type = "string"
        out SPFEECommonTypes::t_PropertyList locations
    ) raises (
        SPFEEAccessCommonTypes::e_AccessError,
        SPFEEProviderAccess::e_ServiceError
    );
```

Cette opération extrait des informations concernant le mode de déchargement de l'application ss-UAP dans le cas d'appelettes Java. Le paramètre terminalInfo est présent en entrée afin d'éviter une invocation explicite de l'opération getTerminalInfo.

8.6.4.11 Opération listServiceSessions()

```
    void listServiceSessions (
        in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
        in SPFEEAccessCommonTypes::t_SpecifiedAccessSession as,
        in t_SessionSearchProperties desiredProperties,
        out SPFEEAccessCommonTypes::t_SessionList sessions
    ) raises (
        SPFEEAccessCommonTypes::e_AccessError,
        SPFEEAccessCommonTypes::e_SpecifiedAccessSessionError,
        SPFEECommonTypes::e_PropertyError,
        SPFEECommonTypes::e_ListError
    );
```

L'opération listServiceSessions() renvoie une liste des sessions de service actives et suspendues dans lesquelles est impliqué le consommateur.

Le paramètre as permet de limiter le contenu de la liste des sessions de service en indiquant une session d'accès dans laquelle les sessions de service sont impliquées. Il peut identifier la session d'accès en cours, une liste de sessions d'accès ou la totalité des sessions d'accès (une session est associée à une session d'accès si elle est utilisée au sein de la session ou si elle a fait l'objet d'une suspension (ou si la participation a fait l'objet d'une suspension) et si elle était en cours d'utilisation au sein de cette session d'accès lorsqu'elle a fait l'objet d'une suspension).

Le paramètre desiredProperties peut être utilisé pour limiter le contenu de la liste des sessions. La liste t_SessionSearchProperties indique des propriétés qui doivent concorder avec celles des sessions. Il définit également si cette concordance concerne l'une, la totalité ou aucune des sessions. (prière de se référer au type t_MatchProperties défini dans le paragraphe "Propriétés et listes de propriétés"). Les valeurs et noms suivants ont été définis pour la liste t_SessionSearchProperties:

– nom: "SessionState"
  valeur: t_SessionState

  si une propriété de la liste t_SessionSearchProperties porte le nom "SessionState", la session doit se trouver dans le même état t_SessionState que celui qui est indiqué par la valeur de la propriété.

– nom: "UserSessionState"
  valeur: t_UserSessionState

  si une propriété de la liste t_SessionSearchProperties porte le nom "UserSessionState", la session doit se trouver dans le même état t_UserSessionState que celui qui est indiqué par la valeur de la propriété.

La liste desiredProperties peut également définir d'autres propriétés propres au revendeur.

La liste de sessions correspondant aux listes desiredProperties et accessSession est renvoyée dans le paramètre sessions d'une sequence de structures t_SessionInfo contenant les éléments t_SessionId, t_ParticipantSecretId, t_PartyId, t_UserSessionState, t_InterfaceList, t_SessionModelList et t_SessionProperties de la session.

L'opération active une exception e_SpecifiedAccessSessionError si le paramètre as n'a pas un format correct ou fournit un identificateur de session d'accès non valide.

L'opération active une exception e_PropertyError si le paramètre desiredProperties n'a pas un format correct ou fournit un nom ou une valeur de propriété non valide (les noms de propriété non valides peuvent être ignorés si le paramètre desiredProperties demande uniquement la concordance de certaines ou d'aucune des propriétés).

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste des sessions n'est pas disponible parce que les sessions du consommateur ne sont pas connues.

8.6.4.12 Opération getSessionModels()

```
void getSessionModels (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId,
       out SPFEECommonTypes::t_SessionModelList sessionModels
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError,
       SPFEECommonTypes::e_ListError
);
```

L'opération getSessionModels() renvoie une liste des modèles de session pris en charge par une session de service. Elle peut être utilisée pour des sessions actives ou suspendues.

Le paramètre sessionId identifie la session dont les modèles de session font l'objet de l'extraction.

Le paramètre sessionModels fournit la liste des modèles de session pris en charge par la session. Il se constitue d'une sequence de structures t_SessionModel contenant le nom du modèle de session et une liste de propriétés de ce modèle. Un modèle de session SPFEE a été défini pour l'environnement SPFEE. La référence [6] fournit des informations complémentaires concernant ce modèle d'objets de session générique évolué, qui est toutefois en dehors du domaine d'application du présent supplément.

L'opération active une exception e_SessionError si le paramètre sessionId n'est pas valide, si l'état de la session interdit l'accès aux modèles de session (par exemple, si la session est suspendue) ou si la session refuse de renvoyer la liste sessionModels.

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste sessionModels n'est pas disponible parce que les modèles de session pris en charge par la session ne sont pas connus.

8.6.4.13 Opération getSessionInterfaceTypes()

```
void getSessionInterfaceTypes (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId,
       out SPFEECommonTypes::t_InterfaceTypeList itfTypes
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError,
       SPFEECommonTypes::e_ListError
);
```

L'opération getSessionInterfaceTypes() renvoie une liste de types des interfaces prises en charge par une session de service. Elle peut être utilisée pour des sessions actives ou suspendues.

Le paramètre sessionId identifie la session dont les types d'interface font l'objet de l'extraction.

Le paramètre itfTypes contient la liste de tous les types des interfaces prises en charge par la session. Il se constitue d'une sequence d'éléments t_InterfaceTypeName contenant chacun une chaîne représentant un type d'interface pris en charge par la session. Le paramètre itfTypes doit contenir tous les types des interfaces pouvant être prises en charge par la session.

L'opération active une exception e_SessionError si le paramètre sessionId n'est pas valide, si l'état de la session interdit l'accès aux modèles de session (par exemple, si la session est suspendue) ou si la session refuse de renvoyer la liste itfTypes.

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste itfTypes n'est pas disponible parce que les types des interfaces prises en charge par la session ne sont pas connus.

8.6.4.14 Opération getSessionInterface()

```
void getSessionInterface (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId,
       in SPFEECommonTypes::t_InterfaceTypeName itfType,
       out SPFEECommonTypes::t_InterfaceStruct itf
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError,
       SPFEECommonTypes::e_InterfacesError
);
```

L'opération getSessionInterface() renvoie une interface du type demandé, prise en charge par une session de service. Elle peut être utilisée pour des sessions actives.

Le paramètre sessionId identifie la session dont les interfaces font l'objet de l'extraction.

Le paramètre itfType identifie le type d'interface de la référence d'interface devant être renvoyée.

Le paramètre itf renvoyé par cette opération contient le nom t_InterfaceTypeName, une référence d'interface (t_IntRef) et les propriétés d'interface (t_InterfaceProperties) des types d'interfaces demandés.

L'opération active une exception e_SessionError si le paramètre sessionId n'est pas valide, si l'état de la session interdit l'accès aux modèles de session (par exemple, si la session est suspendue) ou si la session refuse de renvoyer la liste itfTypes.

L'opération active une exception e_SessionInterfacesError avec le code d'erreur InvalidSessionInterfaceType si la session ne prend pas en charge des interfaces du type itfType.

8.6.4.15 Opération getSessionInterfaces()

```
void getSessionInterfaces (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId,
       out SPFEECommonTypes::t_InterfaceList itfs
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError,
       SPFEECommonTypes::e_ListError
);
```

L'opération getSessionInterfaces() renvoie la liste des interfaces prises en charge par une session de service. Elle peut uniquement être utilisée pour des sessions actives.

Le paramètre sessionId identifie la session dont les interfaces font l'objet de l'extraction.

Cette opération renvoie le paramètre itfs, constitué d'une sequence de structures t_InterfaceStruct contenant le nom t_InterfaceTypeName, une référence d'interface (t_IntRef) et les propriétés d'interface (t_InterfaceProperties) de chaque interface.

L'opération active une exception e_SessionError si le paramètre sessionId n'est pas valide, si l'état de la session interdit l'accès aux modèles de session (par exemple, si la session est suspendue) ou si la session refuse de renvoyer la liste itfTypes.

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste itfs n'est pas disponible parce que l'interface prise en charge par la session n'est pas connue.

8.6.4.16 Opération listSessionInvitations()

```
void listSessionInvitations (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       out SPFEEAccessCommonTypes::t_InvitationList invitations
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       SPFEECommonTypes::e_ListError
);
```

L'opération listSessionInvitations() renvoie une liste des invitations à se joindre à une session de service qui ont été envoyées au consommateur par l'intermédiaire de ce revendeur.

Cette opération renvoie le paramètre invitations contenant une sequence de structures t_SessionInvitation.

```
struct t_SessionInvitation {
       t_InvitationId id;
       t_UserId inviteeId;
       t_SessionPurpose purpose;
       t_InvitationReason reason;
       t_InvitationOrigin origin;
};
```

Le paramètre id identifie une invitation particulière. Il identifie cette invitation de manière non ambiguë parmi celles qui ont été faites pour ce consommateur au niveau de ce revendeur (d'autres consommateurs de ce revendeur peuvent faire l'objet d'invitations avec le même identificateur). Cet identificateur est utilisé dans l'opération joinSessionWithInvitation() pour permettre au consommateur de se joindre à la session concernée par cette invitation.

Le paramètre inviteeId contient l'identificateur du consommateur (ce paramètre n'est pas nécessaire dans ce contexte parce que l'identificateur de l'utilisateur est connu par le biais de la session d'accès. Il figure dans cette structure afin de permettre la livraison des invitations en dehors d'une session d'accès et permettre au destinataire de vérifier qu'il est concerné par l'invitation).

Le paramètre purpose contient une chaîne indiquant le but de la session.

Le paramètre reason contient une chaîne indiquant le motif pour lequel ce consommateur a été invité à se joindre à cette session.

Le paramètre origin contient une structure constituée de l'identificateur d'utilisateur du consommateur qui a demandé l'envoi de l'invitation et de l'identificateur sessionId de la session à laquelle le consommateur destinataire a été invité à se joindre (l'identificateur sessionId est fourni afin que le consommateur invité puisse identifier la session à laquelle se réfère le consommateur invitant lorsqu'il contacte ce dernier).

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si la liste d'invitations n'est pas disponible.

8.6.4.17 Opération listSessionAnnouncements()

```
void listSessionAnnouncements (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in t_AnnouncementSearchProperties desiredProperties,
       out SPFEECommonTypes::t_AnnouncementList announcements
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       SPFEECommonTypes::e_PropertyError,
       SPFEECommonTypes::e_ListError
);
```

L'opération listSessionAnnouncements() renvoie une liste d'annonces de session qui ont été faites par le biais de ce revendeur.

L'annonce des sessions peut se faire à la suite de demandes des participants de session (prière de se référer à l'ensemble de fonctionnalités de participants multiples) ou en raison des propriétés d'initialisation de la session, de l'atelier de service ou de politiques de l'utilisateur qui démarre le service. Le processus d'annonce des sessions n'est pas défini au niveau du point Ret-RP. Cette opération est toutefois fournie afin de permettre au consommateur de demander une liste de sessions qui ont fait l'objet d'une annonce (il est possible de préciser les annonces afin de limiter leur distribution à des groupes donnés). Cette opération renvoie une liste des annonces correspondant à la liste desiredProperties telle qu'elle est spécifiée par le consommateur.

Le paramètre desiredProperties peut être utilisé pour limiter le contenu de la liste des annonces. Le paramètre t_AnnouncementSearchProperties indique les propriétés qui doivent correspondre à celles des annonces (prière de se référer au type t_MatchProperties défini dans le paragraphe "Propriétés et listes de propriétés"). Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_AnnouncementSearchProperties, de sorte que son utilisation est propre au revendeur.

Le paramètre announcements continent une liste d'annonces disponibles pour le consommateur et concordant avec la liste desiredProperties. Ce paramètre se constitue d'une sequence de structures t_SessionAnnouncement contenant les propriétés t_AnnouncementProperties de l'annonce. Aucune valeur de propriété ni aucun nom n'ont été définis de manière spécifique pour la liste t_AnnouncementProperties, de sorte que son utilisation est propre au revendeur.

L'opération active une exception e_PropertyError si le paramètre desiredProperties n'a pas un format correct ou fournit un nom ou une valeur de propriété non valide (les noms de propriété non valides peuvent être ignorés si le paramètre desiredProperties demande uniquement la concordance de certaines ou d'aucunes des propriétés).

L'opération active une exception e_ListError avec le code d'erreur ListUnavailable si aucune liste d'annonce n'est disponible.

8.6.4.18 Opération startService()

```
void startService (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEEAccessCommonTypes::t_ServiceId serviceId,
      in t_ApplicationInfo app,
      in SPFEECommonTypes::t_SessionModelReq sessionModelReq,
      in t_StartServiceUAProperties uaProperties,
      in t_StartServiceSSProperties ssProperties,
      out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      e_ServiceError,
      e_ApplicationInfoError,
      SPFEECommonTypes::e_SessionModelError,
      e_StartServiceUAPropertyError,
      e_StartServiceSSPropertyError
);
```

L'opération startService() démarre une session de service du consommateur.

Le paramètre serviceId contient un identificateur de type de service indiquant le type de service de la session que le consommateur souhaite démarrer.

Le paramètre app est une structure se composant des informations qui concernent l'application et qui sont utilisées pour interagir avec la session de service. La structure contient le nom de l'application, la version, le numéro de série, une liste de propriétés, etc. Elle contient également une liste d'interfaces prises en charge par l'application pouvant contenir, de manière optionnelle, des

références vers certaines de ces interfaces (si elles sont disponibles), une liste de modèles et d'ensembles de fonctionnalités de session, se composant à son tour des références d'interfaces éventuelles et d'une liste de descriptions d'interface de flux.

Le paramètre sessionModelReq définit les modèles et les ensembles de fonctionnalités de session que le consommateur souhaite pour la session. Il permet au consommateur de demander la prise en charge d'un, de la totalité ou d'aucun des modèles de session.

Le paramètre uaProperties contient une liste de propriétés qui sera interprétée par le domaine revendeur avant le démarrage de la session de service. Aucune valeur ni nom de propriété ne sont définis, de sorte que l'utilisation de cette liste est propre au revendeur. Elle permet au consommateur de définir certaines préférences ou d'autres contraintes qu'il souhaite voir appliquer uniquement à cette session de service et dont le revendeur doit avoir connaissance avant le démarrage de la session (ces propriétés peuvent affecter le choix de l'atelier de service pour la session).

Le paramètre ssProperties contient une liste de propriétés qui sera interprétée par la session de service au moment de son démarrage (c'est-à-dire, avant que les références vers la session soient renvoyées au domaine consommateur). Aucune valeur ni nom de propriété ne sont définis. L'utilisation de ce paramètre est propre au service et l'interprétation des propriétés concerne uniquement la session de service (ce paramètre permet au domaine consommateur ou à l'application de transférer des informations spécifiques vers la session de service, sans que le domaine revendeur soit impliqué dans leur interprétation).

Le paramètre sessionInfo est une structure qui contient des informations permettant au domaine consommateur de faire référence à cette session dans d'autres opérations sur cette interface. Il contient également des informations pour la partie "utilisation" de la session, incluant les références d'interface qui permettent l'interaction avec la session (prière de se référer au paragraphe "Informations de service et de session").

Les exceptions suivantes sont activées par cette opération:

L'opération active une exception e_ServiceError si le paramètre serviceId n'est pas valide ou n'est pas connu du revendeur, ou s'il n'est pas possible de créer une session de service.

L'opération active une exception e_ApplicationInfoError si la liste t_ApplicationInfo contient des valeur non connues ou non valides ou si l'application n'est pas compatible avec le type du service devant être démarré.

L'opération active une exception e_SessionModelError si des modèles de session ou des ensembles de fonctionnalités non valides sont nécessaires pour la session de service.

L'opération active une exception e_StartServiceUAPropertyError si la liste de propriétés uaProperties contient une erreur. Les codes d'erreur de propriété sont les mêmes que pour l'exception e_PropertyError (prière de se référer au paragraphe "Propriétés et listes de propriétés" pour plus de détails).

L'opération active une exception e_StartServiceSSPropertyError si la liste de propriétés ssProperties contient une erreur. Les codes d'erreur de propriété sont les mêmes que pour l'exception e_PropertyError.

8.6.4.19 Opération endSession()

```
void endSession (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError
);
```

L'opération endSession() met fin à une session de service du consommateur. Elle peut être utilisée pour des sessions dans lesquelles le consommateur est actuellement actif et des sessions qui ont été suspendues ou dans lesquelles le consommateur a suspendu sa participation.

Le paramètre sessionId contient l'identificateur de la session qui doit se terminer.

L'opération active une exception e_SessionError si l'identificateur sessionId n'est pas valide, si la session refuse de se terminer en raison de l'état de session de l'utilisateur ou si l'utilisateur ne possède pas d'autorisation.

8.6.4.20 Opération endMyParticipation()

```
void endMyParticipation (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEECommonTypes::t_SessionId sessionId
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      e_SessionError
);
```

L'opération endMyParticipation() met fin à la participation du consommateur dans une session de service. Elle peut être utilisée pour des sessions dans lesquelles le consommateur est actuellement actif et des sessions qui ont été suspendues ou dans lesquelles le consommateur a suspendu sa participation.

Le paramètre sessionId contient l'identificateur de la session pour laquelle l'utilisateur souhaite mettre fin à sa participation.

L'opération active une exception e_SessionError si l'identificateur sessionId n'est pas valide, si la session refuse de se terminer en raison de l'état de session de l'utilisateur ou si l'utilisateur ne possède pas d'autorisation.

8.6.4.21 Opération suspendSession()

```
void suspendSession (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEECommonTypes::t_SessionId sessionId
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      e_SessionError
);
```

L'opération suspendSession() suspend une session de service du consommateur. Elle peut être utilisée pour des sessions dans lesquelles le consommateur est actuellement actif et des sessions dans lesquelles le consommateur a déjà suspendu sa participation.

Le paramètre sessionId contient l'identificateur de la session devant faire l'objet d'une suspension.

L'opération active une exception e_SessionError si l'identificateur sessionId n'est pas valide, si la session refuse la suspension en raison de l'état de session de l'utilisateur ou si l'utilisateur ne possède pas d'autorisation.

8.6.4.22 Opération suspendMyParticipation()

```
void suspendMyParticipation (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEECommonTypes::t_SessionId sessionId
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      e_SessionError
);
```

L'opération suspendMyParticipation() suspend la participation du consommateur dans une session de service. Elle peut être utilisée pour des sessions dans lesquelles le consommateur est actuellement actif.

Le paramètre sessionId contient l'identificateur de la session dans laquelle l'utilisateur suspend sa participation.

L'opération active une exception e_SessionError si l'identificateur sessionId n'est pas valide, si la session refuse de suspendre la participation de l'utilisateur en raison de l'état de session de ce dernier ou si l'utilisateur ne possède pas d'autorisation.

8.6.4.23 Opération resumeSession()

```
void resumeSession (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId,
       in t_ApplicationInfo app,
       out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError,
       e_ApplicationInfoError
);
```

L'opération resumeSession() effectue la reprise d'une session de service. Elle est utilisée pour une session suspendue.

Le paramètre sessionId contient l'identificateur de la session devant faire l'objet d'une reprise.

Le paramètre app est une structure qui contient des informations concernant l'application qui sera utilisée pour l'interaction avec la session de service. Cette application peut différer de celle qui était mise en œuvre précédemment par l'utilisateur lorsque la session a fait l'objet d'une suspension.

Le paramètre sessionInfo est une structure qui contient des informations permettant au domaine consommateur de faire référence à cette session dans d'autres opérations sur cette interface. Il contient également des informations pour la partie "utilisation" de la session, incluant les références d'interface permettant l'interaction avec la session (prière de se référer au paragraphe "Informations de service et de session").

L'opération active une exception e_SessionError si sessionId n'est pas valide, si la session refuse la reprise en raison de l'état de session de l'utilisateur ou si l'utilisateur ne possède pas d'autorisation.

L'opération active une exception e_ApplicationInfoError si la liste t_ApplicationInfo contient des valeurs non connues ou non valides, ou si l'application est incompatible avec le type de service devant être repris.

8.6.4.24 Opération resumeMyParticipation()

```
void resumeMyParticipation (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId,
       in t_ApplicationInfo app,
       out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError,
       e_ApplicationInfoError
);
```

L'opération resumeMyParticipation() effectue la reprise de la participation du consommateur dans une session de service. Elle peut être utilisée pour une session dans laquelle le consommateur a précédemment suspendu sa participation.

Le paramètre sessionId contient l'identificateur de la session dans laquelle l'utilisateur reprend sa participation.

Le paramètre app est une structure qui contient des informations concernant l'application qui sera utilisée pour l'interaction avec la session de service. Cette application peut différer de celle qui était mise en œuvre précédemment par l'utilisateur lorsque la session a fait l'objet d'une suspension.

Le paramètre sessionInfo est une structure qui contient des informations permettant au domaine consommateur de faire référence à cette session dans d'autres opérations sur cette interface. Il contient également des informations pour la partie "utilisation" de la session, incluant les références d'interface permettant l'interaction avec la session (prière de se référer au paragraphe "Informations de service et de session").

L'opération active une exception e_SessionError si le paramètre sessionId n'est pas valide, si la session refuse la reprise en raison de l'état de session de l'utilisateur ou si l'utilisateur ne possède pas d'autorisation.

L'opération active une exception e_ApplicationInfoError si la liste t_ApplicationInfo contient des valeurs non connues ou non valides, ou si l'application est incompatible avec le type de service devant être repris.

8.6.4.25 Opération joinSessionWithInvitation()

```
void joinSessionWithInvitation (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEEAccessCommonTypes::t_InvitationId invitationId,
      in t_ApplicationInfo app,
      in SPFEECommonTypes::t_PropertyList  joinProperties,
      out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      e_SessionError,
      SPFEEAccessCommonTypes::e_InvitationError,
      e_ApplicationInfoError
);
```

L'opération joinSessionWithInvitation() permet au consommateur de se joindre à une session de service existante dans laquelle il a été invité.

Le paramètre invitationId contient l'identificateur de l'invitation. L'invitation détenue par le revendeur contient des informations suffisantes permettant à ce dernier d'entrer en contact avec la session de service et de demander que le consommateur soit autorisé à se joindre à la session.

Le paramètre app est une structure qui contient des informations concernant l'application qui sera utilisée pour l'interaction avec la session de service.

Le paramètre sessionInfo est une structure qui contient des informations permettant au domaine consommateur de faire référence à cette session dans d'autres opérations sur cette interface. Il contient également des informations pour la partie "utilisation" de la session, incluant les références d'interface permettant l'interaction avec la session (prière de se référer au paragraphe "Informations de service et de session").

L'opération active une exception e_SessionError si la session refuse d'accueillir le consommateur.

L'opération active une exception e_InvitationError si le paramètre invitationId n'est pas valide.

L'opération active une exception e_ApplicationInfoError si la liste t_ApplicationInfo contient des valeurs non connues ou non valides, ou si l'application est incompatible avec le type de service devant être joint.

8.6.4.26 Opération joinSessionWithAnnouncement()

```
void joinSessionWithAnnouncement (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEEAccessCommonTypes::t_AnnouncementId announcementId,
      in t_ApplicationInfo app,
      in SPFEECommonTypes::t_PropertyList  joinProperties,
      out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      e_SessionError,
      e_AnnouncementError,
      e_ApplicationInfoError
);
```

L'opération joinSessionWithAnnouncement() permet au consommateur de se joindre à une session de service existante pour laquelle le consommateur a détecté une annonce. Les annonces de session peuvent être obtenues d'un certain nombre de manières (non décrites au niveau du point Ret-RP), y compris par le biais d'une session de service spécialisée.

Le paramètre announcementId est l'identificateur de l'annonce. L'annonce détenue par le revendeur contient des informations suffisantes permettant à ce dernier d'entrer en contact avec la session de service et de demander que le consommateur soit autorisé à se joindre à la session.

Le paramètre app est une structure qui contient des informations concernant l'application qui sera utilisée pour l'interaction avec la session de service.

Le paramètre sessionInfo est une structure qui contient des informations permettant au domaine consommateur de faire référence à cette session dans d'autres opérations sur cette interface. Il contient également des informations pour la partie "utilisation" de la session, incluant les références d'interface permettant l'interaction avec la session (prière de se référer au paragraphe "Informations de service et de session").

L'opération active une exception e_SessionError si la session refuse d'accueillir le consommateur.

L'opération active une exception e_AnnouncementError si le paramètre announcementId n'est pas valide.

L'opération active une exception e_ApplicationInfoError si les paramètres ne sont pas connus ou si la liste t_ApplicationInfo contient des valeurs non connues ou non valides, ou si l'application est incompatible avec le type de service devant être joint.

8.6.4.27 Opération replyToInvitation()

```
void replyToInvitation (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEEAccessCommonTypes::t_InvitationId invitationId,
      in SPFEECommonTypes::t_InvitationReply reply
) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEEAccessCommonTypes::e_InvitationError,
      SPFEECommonTypes::e_InvitationReplyError
);
```

L'opération replyToInvitation() permet au consommateur de répondre à la réception d'une invitation. Elle lui permet d'indiquer au revendeur s'il a l'intention ou non de se joindre à la session ou de lui indiquer un emplacement à rechercher pour le consommateur (cette opération ne lui permet pas de se joindre à la session).

Le paramètre invitationId contient l'identificateur de l'invitation.

Le paramètre reply est une structure qui contient les informations concernant la session (prière de se référer au paragraphe "Invitations et annonces" pour plus de détails).

L'opération active une exception e_InvitationError si le paramètre invitationId n'est pas valide.

L'opération active une exception e_InvitationReplyError si le paramètres reply est erroné.

8.6.5 Interface i_RetailerAnonAccess

```
interface i_RetailerAnonAccess
      : i_RetailerAccess
{
      // No operations defined at present
};
```

L'interface i_RetailerAnonAccess fournit l'accès aux services pour un consommateur anonyme. Elle est utilisée par ce dernier pour toutes les opérations au sein d'une session d'accès avec le revendeur.

Cette interface est renvoyée par l'invocation de l'opération requestAnonAccess() sur l'interface i_RetailerInitial lorsque le consommateur a établi une session d'accès anonyme avec le revendeur.

Cette interface hérite de l'interface i_RetailerAccess, qui est actuellement vide. L'interface i_RetailerAnonAccess contiendra toutes les opérations qu'elle partage avec l'interface i_RetailerNamedAccess, ce qui signifie que les opérations fournies par cette interface feront l'objet d'une modification future.

8.6.6 Interface i_DiscoverServicesIterator

```
interface i_DiscoverServicesIterator
{
      // Operations defined in the following subsections
};
```

Cette interface renvoyée par l'opération discoverServices() sur l'interface i_RetailerNamedAccess est utilisée pour accéder aux services restants qui n'ont pas été renvoyés par l'invocation de l'opération discoverServices().

L'opération discoverServices() renvoie une liste des services dont les propriétés correspondent à certaines propriétés définies par le consommateur. Cette interface permet au consommateur d'accéder aux services restants qui n'ont pas été renvoyés par l'invocation de l'opération discoverServices(). Ceci est nécessaire parce que la liste des service en correspondance peut être très longue et mettre éventuellement en jeu des volumes d'informations trop importants pour être traités par l'application du consommateur.

L'opération discoverServices(), suivie éventuellement d'un certain nombre d'invocations de l'opération nextN() sur cette interface permet au consommateur d'accéder à tous les services qui correspondent aux propriétés sans en recevoir la totalité en une seule fois.

8.6.6.1 Opération maxLeft()

```
void maxLeft (
     out unsigned long n
) raises (
     e_UnknownDiscoverServicesMaxLeft
);
```

L'opération maxLeft() renvoie le nombre maximal des services qui seront reçus sur cette interface. L'accès à ces services se fait par des invocations répétées de l'opération nextN().

L'opération maxLeft() active l'exception (e_UnknownDiscoverServicesMaxLeft) si le revendeur n'est pas en mesure de déterminer le nombre maximal de services pouvant être renvoyés.

8.6.6.2 Opération nextN()

```
void nextN (
      in unsigned long n,
      out SPFEEAccessCommonTypes::t_ServiceList services,
      out boolean moreLeft
) raises (
      SPFEECommonTypes::e_ListError
);
```

L'opération nextN() permet au consommateur d'accéder aux services restants qui n'ont pas été renvoyés par l'opération discoverServices() ou par des invocations précédentes de cette opération. L'accès à ces services peut se faire par le biais d'invocations répétées de l'opération nextN().

Le paramètre n indique le nombre maximal de services devant être renvoyés. La longueur de la liste services n'excédera pas la valeur du paramètre n.

Les services restants sont renvoyés dans le paramètre t_ServiceList services qui contient une sequence de structures t_ServiceInfo constituées des éléments t_ServiceId, t_UserServiceName (nom de l'utilisateur pour le service) et d'une sequence de propriétés de service t_ServiceProperties.

Le paramètre moreLeft contient un type boolean indiquant au consommateur s'il reste d'autres services après cette invocation de l'opération nextN().

8.6.6.3 Opération destroy()

```
void destroy ();
```

L'opération destroy() est utilisée pour informer le revendeur que le consommateur n'utilise plus l'interface i_DiscoverServicesIterator. Elle peut être invoquée à tout instant par le consommateur (c'est-à-dire que le consommateur peut détruire cette interface avant d'avoir extrait tous les services). Le consommateur ne sera plus en mesure d'utiliser la référence vers l'interface i_DiscoverServicesIterator une fois que cette invocation s'est terminée.

8.7 Gestion de l'abonnement

Le présent sous-paragraphe décrit les interfaces offertes par le revendeur au consommateur pour la prise en charge des fonctionnalités liées à l'abonnement. Deux types d'interfaces existent en fonction du type de consommateur: la première est mise en œuvre par un utilisateur anonyme qui souhaite devenir un abonné et la deuxième par un abonné qui souhaite gérer les informations liées à son abonnement. Les interactions liées à l'abonnement ont un caractère général (indépendant du service). Il convient de noter que des services auxiliaires peuvent implémenter des fonctionnalités d'abonnement en ligne; ces fonctionnalités sont toutefois propres au service et, de ce fait, en dehors du domaine d'application du présent supplément. L'accès aux interfaces d'abonnement du revendeur se fait dans le contexte de la session d'accès; il permet d'extraire la liste des services auxquels le consommateur, l'abonné ou l'utilisateur est associé et les profils de service correspondants, ainsi que de modifier les données d'abonnement.

Les principales fonctionnalités offertes à un utilisateur anonyme sont les suivantes:

– extraction de la liste des services (c'est-à-dire, des services disponibles auprès de ce revendeur);

– création d'un nouvel abonné.

Les principales fonctionnalités offertes à un abonné sont les suivantes:

– extraction de la liste des services contenant, soit les services qui sont disponibles auprès de ce revendeur, soit ceux qui ont fait l'objet d'un abonnement;

– création, modification, suppression et demandes concernant les informations liées à un nouvel abonné (utilisateurs finaux associées, groupes d'utilisateurs finaux, etc.);

– création, modification, suppression et demandes concernant les contrats de service (définitions des profils des services qui ont fait l'objet d'un abonnement);

– extraction des profils de service (profil de service de groupe SAG) pour un utilisateur donné (ou un terminal, ou un point NAP).

8.7.1 Définitions du type de gestion d'abonnement

Le présent sous-paragraphe contient les définitions en langage IDL des informations nécessaires pour le traitement des abonnements, des abonnés et des utilisateurs finaux au sein d'un domaine fournisseur. Ces définitions facilitent la compréhension des descriptions d'interface.

La Figure 8-4 représente les relations entre un service et un abonné, impliquant un certain nombre de profils de service (squelette de service, profil d'abonnement et profil de service de groupe SAG).

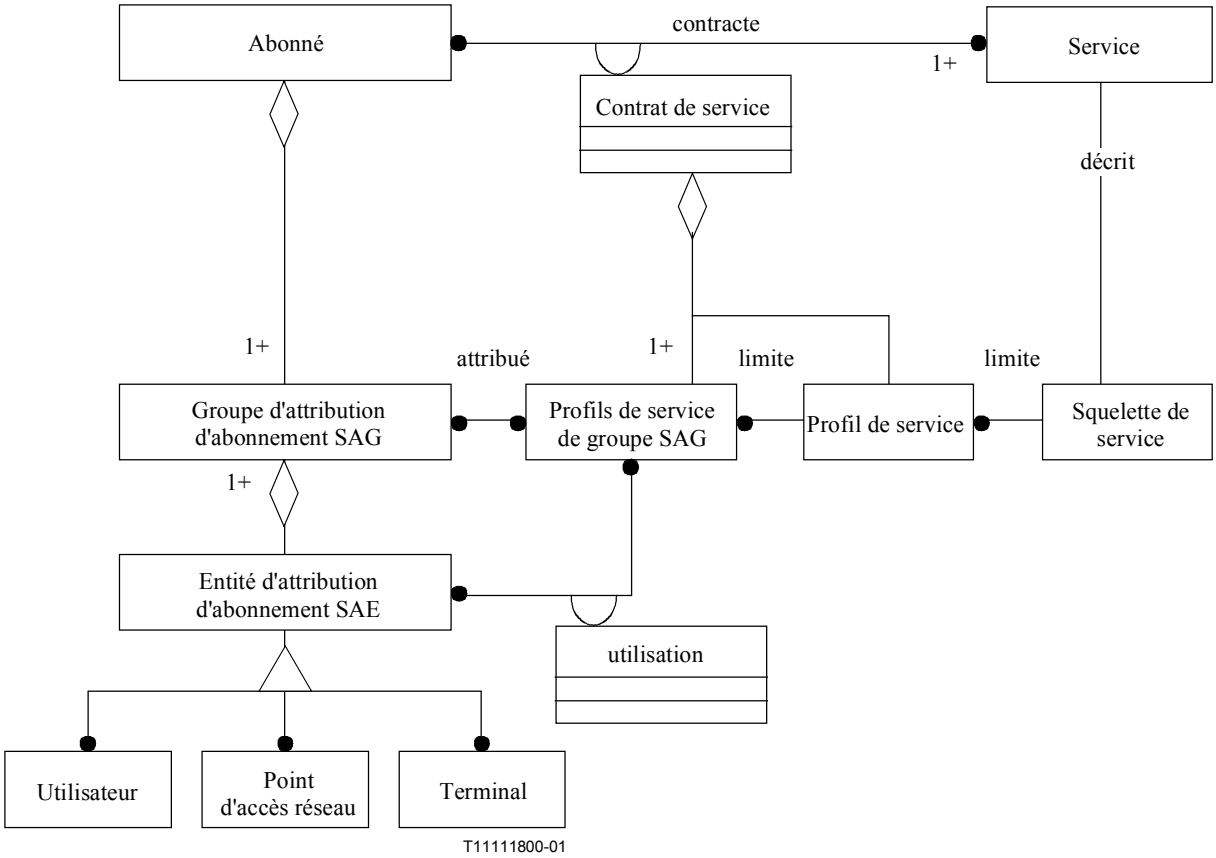


Figure 8-4 – Modèle d'informations de gestion d'abonnement

Un abonné contacte un certain nombre de services (au moins un considéré comme tel). Les informations suivantes sont associées à l'abonné:

```
struct t_Subscriber {
   t_AccountNumber                    accountNumber;
   SPFEECommonTypes::t_UserId         subscriberName;
   t_Person                           identificationInfo;
   t_Person                           billingContactPoint;
   string                             RatePlan;
   any                                paymentRecord;
   any                                credit;
};
```

Le champ accountNumber généré par l'abonné est non ambigu dans son domaine. Il est utilisé au sein du domaine pour identifier l'abonné et éventuellement ses factures. Le champ SubscriberName contient un nom choisi par l'abonné[41], qui se présente en général sous une forme plus conviviale que le numéro accountNumber. Il existe une correspondance biunivoque entre ces deux identificateurs. Le champ identificationInfo est utilisé pour stocker des informations telles que le nom et l'adresse de l'abonné, etc. Le champ billingContactPoint contient les informations nécessaires à l'envoi des factures. Le champ paymentRecord contient des informations concernant le paiement des dernières factures à des fins de vérification du statut de facturation. Le champ credit contient des informations concernant les acomptes, les crédits accordés à l'abonné, etc.

Le contrat de service qui fait l'objet d'un accord définit les conditions de fourniture du service pour chacun des abonnements au service. Il est défini comme suit:

```
struct t_ServiceContract {
   SPFEEAccessCommonTypes::t_ServiceId       serviceId;
   t_AccountNumber                           accountNumber;
   short                                     maxNumOfServiceProfiles;
   t_DateTime                                actualStart;
   t_DateTime                                requestedStart;
   t_Person                                  requester;
   t_Person                                  technicalContactPoint;
   t_AuthLimit                               authorityLimit;
   t_SubscriptionProfile                     subscriptionProfile;
   t_SagServiceProfileList                   sagServiceProfileList;
};
```

Les champs serviceId et accountNumber identifient en commun sans ambiguïté un contrat de service. Les profils de service constituent la partie principale du contrat de service. Les autres champs fournissent des informations supplémentaires concernant le contrat (date de début, date de début demandée, demandeur, contact technique, etc.).

Un squelette de service décrit les caractéristiques du service accessible par l'intermédiaire du revendeur.

```
struct t_ServiceTemplate {
   SPFEEAccessCommonTypes::t_ServiceId           serviceInstanceId;
   SPFEEAccessCommonTypes::t_UserServiceName     serviceInstanceName;
   t_ServiceIdList                               requiredServices;
   t_ServiceDescription                          serviceDescription;
};
```

La description du service contient les caractéristiques d'un type général de service. Elle est réutilisée dans le squelette de service pour décrire les caractéristiques d'une instance particulière de service (implémentation propre à un type de service) et dans les profils de service pour représenter le contrat de service souscrit par un abonné (pour l'ensemble des utilisateurs associés ou pour un groupe d'entre eux).

```
struct t_ServiceDescription {
   SPFEEAccessCommonTypes::t_ServiceId           serviceTypeId;
   SPFEEAccessCommonTypes::t_UserServiceName     serviceTypeName;
   t_ParameterList                               serviceCommonParams;
   t_ParameterList                               serviceSpecificParams;
};
```

La liste de paramètres se constitue d'une succession de structures contenant un nom de paramètre, une configuration de paramètre et une valeur de paramètre.

---

[41] Ce nom peut être employé pour générer des identificateurs d'utilisateur. Un utilisateur X abonné auprès d'un détaillant A peut, par exemple, avoir un identificateur X@A.

```
typedef string  t_ParameterName;
enum t_ParameterConfigurability { FIXED_BY_PROVIDER,
   CONFIGURABLE_BY_SUBSCRIBER, CUSTOMIZABLE_BY_USER };

typedef any     t_ParameterValue;

struct t_Parameter {
   t_ParameterName                      name;
   t_ParameterConfigurability           configurability;
   t_ParameterValue                     value;
};

typedef sequence<t_Parameter> t_ParameterList;
```

Le revendeur peut fournir à l'abonné l'option de sélectionner des paramètres propres au service, s'appliquant à toutes ses entités associées[42] dans le cas du profil d'abonnement ou à un groupe de ces entités dans le cas de profils de service de groupe SAG; cette sélection limite les choix fournis dans le squelette de service (en imposant des restrictions pour les associations représentées par la Figure 8-4). Ces profils constituent la partie principale du contrat de service.

```
typedef string t_ServiceProfileId;
struct t_ServiceProfile {
   t_ServiceProfileId             spId;
   t_ServiceDescription           serviceDescription;
};
typedef t_ServiceProfile t_SagServiceProfile;
typedef t_ServiceProfile t_SubscriptionProfile;
```

Il est possible d'associer à un abonné un ensemble d'entités, appelé "liste d'entités d'assignation d'abonnement" (SAE) concernant des utilisateurs, des terminaux ou des points NAP.

```
enum t_entityType {user, terminal, nap};

/* Entity Id identifies uniquely a SAE inside the provider domain. */
union t_entityId switch (t_entityType) {
   case user:       SPFEECommonTypes::t_UserId             userId;
   case terminal:   SPFEEAccessCommonTypes::t_TerminalId   terminalId;
   case nap:        SPFEEAccessCommonTypes::t_NAPId        napId;
};

typedef sequence<t_entityId> t_entityIdList;

/* A SAE is characterized by an identifier, a name and a set of properties. */
struct t_Sae {
   t_entityId                              entityId;
   string                                  entityName;
   SPFEECommonTypes::t_PropertyList        properties;
};
```

L'abonné peut souhaiter ne pas accorder à toutes ces entités les mêmes caractéristiques de service (ou privilèges). Il peut, pour ce faire, les regrouper dans un ensemble de groupes d'attribution d'abonnement (SAG):

```
typedef short    t_SagId;

/* A SAG is characterized by its identifier, a textual description of the
* group and the list of entities composing it. */
struct t_Sag {
   t_SagId                    sagId;
   string                     sagDescription;
   t_entityIdList             entityList;
};
```

[42] Utilisateurs, terminaux ou points d'accès réseau.

L'abonné peut assigner à chaque groupe des profils de service particuliers (profil de service de groupe SAG). La principale raison de l'utilisation de ce regroupement est de faciliter le processus d'abonnement (assignation de profils à des utilisateurs) pour des domaines d'abonné dans lesquels les utilisateurs finaux sont classés d'une manière naturelle en catégories, organisations ou zones géographiques, etc., qui nécessitent les mêmes privilèges d'utilisation de service. La seule limitation est que chaque entité SAE doit être assignée à un groupe SAG et un seul pour chaque service.

Un groupe SAG par défaut avec un identificateur SAG(0) est créé pour chaque abonné. Toute entité SAE est toujours assignée à ce groupe SAG même si elle est assignée par ailleurs à un autre groupe SAG particulier. Lorsqu'un utilisateur est retiré d'un groupe SAG quelconque, il reste associé au groupe SAG par défaut. Le groupe SAG par défaut ne peut pas être associé à des services par défaut et les utilisateurs ne peuvent pas être associés à ce groupe SAG (ils le sont implicitement au moment de leur création).

Il est également possible d'assigner des profils de service à des entités SAE individuelles ou d'annuler une telle assignation. Ceci est particulièrement intéressant pour de petites organisations d'abonnés (par exemple des abonnés résidentiels) dans lesquelles la définition des groupes d'utilisateurs n'est pas strictement nécessaire et ne fournit pas d'aide à l'abonné en ce qui concerne la gestion de l'abonnement. Ce processus permet par ailleurs une grande souplesse dans l'assignation du profil de service, du fait que certains utilisateurs dans un groupe peuvent faire l'objet d'une discrimination pour un service donné sans qu'il soit nécessaire de les retirer du groupe ou de créer un nouveau groupe.

La définition de ces limitations utilise la structure suivante:

```
enum t_UsagePermitFlag {USAGE_ALLOWED, USAGE_DISALLOWED};
struct t_UsagePermit {
   t_entityId                 entityId;
   t_ServiceProfileId         serviceProfileId;
   t_UsagePermitFlag          flag;
};
```

Les paragraphes qui suivent décrivent les interfaces offertes au consommateur par le revendeur pour la prise en charge des fonctionnalités d'abonnement. Ces dernières se constituent d'un ensemble d'interfaces propres offertes au niveau du point de référence Ret pour la gestion en ligne des abonnements au service. Des interfaces distinctes sont offertes pour chaque type d'utilisateur (abonnés ou opérateurs revendeurs).

8.7.2 Interface i_SubscriberSubscriptionMgmt

```
// module SPFEERetSubscriberSubscriptionMgmt
interface i_SubscriberSubscriptionMgmt
{
};
```

Cette interface dédiée aux abonnés fournit des opérations pour l'abonnement auprès du revendeur, la conclusion de contrats pour des services et la fourniture d'informations de contrat de service. Elle permet l'annulation et la modification de l'abonnement et du contrat de service ainsi que l'interrogation pour toutes les informations liées à l'abonné.

Les opérations suivantes prennent en charge la candidature, la souscription et l'annulation pour des contrats de service:

– **listServices**() – Renvoie la liste des services fournis par le revendeur.

– **subscribe**() – Permet de créer un contrat d'abonnement avec le revendeur. Les paramètres en entrée sont les informations d'abonné et une liste des services que l'abonné souhaite souscrire. L'opération renvoie l'identificateur de l'abonné et une liste d'identificateurs de contrats de service. Ces derniers seront utilisés pour faire référence à un contrat de service donné.

– **unsubscribe**() – Permet de mettre fin à un ou plusieurs contrats de service ou à la totalité de la relation avec le revendeur.

– **contractService**() – Souscrit l'abonné à un service et renvoie une référence d'interface lorsqu'il est possible de définir le contrat de service (i_ServiceContractInfoMgmt).

– **listSubscribedServices**() – Renvoie la liste des identificateurs des contrats de service et des identificateurs de contrat de service liés. Si un utilisateur est spécifié, l'opération renvoie alors la liste des services accordés par l'abonné à un utilisateur donné. L'identificateur d'abonné (numéro de compte) est fourni comme paramètre en entrée.

Opérations de traitement des informations d'abonné:

– **listSAEs**() – Renvoie la liste des entités associées à l'abonné. Lorsqu'un identificateur SAG (ou une liste de tels identificateurs) est spécifié, l'opération renvoie uniquement les utilisateurs assignés à ce ou à ces groupes SAG.

– **listSAGs**() – Renvoie la liste des identificateurs de groupe SAG pour cet abonné.

– **getSubscriberInfo**() – Renvoie les informations concernant l'abonné.

– **createSAEs**() – Créée les entités spécifiées comme paramètre et renvoie un identificateur pour chacune d'elles.

– **deleteSAEs**() – Supprime les entités spécifiées comme paramètre ainsi que les assignations éventuelles de ces entités à des groupes SAG.

– **createSAGs**() – Crée un ou plusieurs groupes SAG. Une liste d'entités peut être spécifiée pour chaque groupe SAG. Un identificateur de groupe SAG est renvoyé pour faciliter une gestion ultérieure.

– **assignSAEs**() – Assigne une liste d'entités à un groupe SAG.

– **removeSAEs**() – Supprime une liste d'entités dans un groupe SAG.

– **setSubscriberInfo**() – Modifie les informations concernant l'abonné.

Opérations de définition, de modification et d'interrogation de contrats de service:

– **getServiceTemplate**() – Renvoie le squelette de définition du profil de service pour un service donné.

– **defineServiceContract**() – Permet de définir le contrat de service pour un service donné. Ce contrat contient, entre autres informations, l'ensemble de profils de service constituant le contrat de service, à savoir le profil d'abonnement (s'appliquant à tous les utilisateurs) et l'ensemble des profils de service de groupe SAG (chacun de ces profils s'appliquant à un groupe SAG, en accord avec le profil d'abonnement). Renvoie une liste d'identificateurs de profils de groupe SAG facilitant une référence ultérieure. Cette opération est utilisée pour la définition et la modification des contrats de service.

– **defineServiceProfiles**() – Permet de définir un ensemble de profils de service pour un contrat de service, à savoir le profil d'abonnement et l'ensemble de profils de service de groupe SAG. Renvoie une liste d'identificateurs de profil de groupe SAG facilitant une référence ultérieure. Cette opération est utilisée pour la définition et la modification des profils de service.

– **deleteServiceProfiles**() – Supprime un profil de service.

– **getServiceContractInfo**() – Renvoie les informations liées au contrat de service dont l'identificateur est fourni comme paramètre. L'ensemble des profils de groupe SAG est renvoyé si une liste d'identificateurs de profil de groupe SAG est spécifiée.

Opérations d'autorisation et d'activation de profils de service:

– **assignServiceProfile**() – Associe une liste d'entités SAE et de groupes SAG à un profil de service de groupe SAG. Les entités SAE (indiquées de manière explicite ou contenues dans les groupes SAG) seront en mesure d'utiliser le service si le profil de service est actif. Ces composants d'accès SAE recevront une notification et le profil de service de groupe SAG sera mis à leur disposition. L'entité SAE sera en mesure, à partir de ce profil, de personnaliser son propre profil de service utilisateur par le biais du service de gestion de profil de service.

– **removeServiceProfile**() – Supprime l'association entre une liste d'entités SAE et de groupes SAG d'une part et un profil de service de groupe SAG d'autre part. Les entités SAE (indiquées de manière explicite ou contenues dans les groupes SAG) ne seront plus en mesure d'utiliser ce service, sauf si elles sont associées à un autre profil de service actif.

– **activateServiceProfiles**() **–** Active une liste de profils de service de groupe SAG et rend ces services disponibles pour une utilisation. Les entités SAE et les groupes SAG actifs sont les seuls à pouvoir utiliser ce service.

– **deactivateServiceProfiles**() **–** Met fin à l'activation d'une liste de profils de service de groupe SAG. Les utilisateurs (ou entités SAE) assignés à ces profils de service ne seront plus en mesure d'utiliser le service.

8.7.3 Interface i_RetailerSubscriptionMgmt

```
// module SPFEERetRetailerSubscriptionMgmt
interface i_RetailerSubscriptionMgmt
{
};
```

Cette interface fournit des capacités complètes permettant l'abonnement des abonnés, l'ajout, la modification, la suppression et l'interrogation de contrats de service ainsi que l'ajout, la suppression, la modification et l'interrogation d'informations d'abonné. Elle fournit en général l'accès à la totalité de la base de données d'abonnement. Elle constitue un sur-ensemble de l'interface précédente, dédié à un consommateur du type "opérateur de revendeur".

Les opérations suivantes sont fournies:

– **listSubscribers**() – Renvoie la liste des identificateurs d'abonné. Si un identificateur de service est spécifié, l'opération renvoie alors la liste des abonnés de ce service.

– **listServiceContracts**() – Renvoie la liste des identificateurs de contrats de service. Si un identificateur de service est spécifié, l'opération renvoie alors la liste des contrats concernant ce service. Cette opération est équivalente à l'opération listSubscribedServices() sur l'interface i_SubscriberInfoMgmt pour un abonné donné lorsque ce dernier est spécifié.

– **listUsers**() – Renvoie la liste des identificateurs d'utilisateur pour un service donné.

9 Spécifications complètes en langage IDL

9.1 Définitions IDL communes

9.1.1 SPFEECommonTypes.idl

```
// File SPFEECommonTypes.idl
#ifndef spfeecommontypes_idl
#define spfeecommontypes_idl

module SPFEECommonTypes {

// ElementIds (mainly used in usage part)
```

```
// Element types
enum t_ElTypes {
   SpfeeParty,            // see also t_PartyId
   SpfeeResource, SpfeeMember,
   SpfeeGroup, SpfeeMemberGroup, SpfeePartyGroup, SpfeeResourceGroup,
   SpfeeRelation, SpfeeRelationGroup, SpfeeControlRelation,
   SpfeeStreamBinding, SpfeeStreamFlow, SpfeeStreamInterface, SpfeeSFEP
};

// Element Identifier (elements in a service session)
typedef unsigned long t_ElId;

// Element Type Identifiers
typedef t_ElTypes t_ElTypeId;

// Overall element Identifier
struct t_ElementId
{
   t_ElId id;
   t_ElTypeId elType;
};

typedef sequence <t_ElementId> t_ElementIdList;

// t_PropertyList
// list of properties, (name value pairs).
// Used in many operations to allow a list of as yet undefined
// properties, and values, to be sent.
//

typedef string t_PropertyName;
typedef sequence<t_PropertyName> t_PropertyNameList;
typedef any t_PropertyValue;

struct t_Property {
   t_PropertyName name;
   t_PropertyValue value;
};

typedef sequence<t_Property> t_PropertyList;

enum t_HowManyProps {none, some, all};

union t_SpecifiedProps switch (t_HowManyProps) {
   case some: t_PropertyNameList prop_names;
   case none: octet dummy1;
   case all: octet dummy2;
};

typedef string Istring;

// enum t_ReferenceSort {
// ObjectRef,
// StringifiedReference
// };

// union t_Reference switch(t_ReferenceSort) {
// case ObjectRef: Object IRef;
// case StringifiedReference: SPFEECommonTypes::Istring IORef;
// };

enum t_WhichProperties {
   NoProperties,         // don't can ignore all the properties
```

```
   SomeProperties,       // match at least one property (name & value)
   SomePropertiesNamesOnly, // check name only (ignore value)
   AllProperties,        // match all properties (name & value)
   AllPropertiesNamesOnly    // check name only (ignore value)
};

struct t_MatchProperties {
   t_WhichProperties whichProperties;
   t_PropertyList properties;
};

typedef /*CORBA::*/Object t_Interface;

typedef string t_InterfaceTypeName;
typedef sequence<t_InterfaceTypeName> t_InterfaceTypeList;
typedef t_PropertyList t_InterfaceProperties;

struct t_InterfaceStruct {
        t_InterfaceTypeName itfType;
        Object ref;
          // if NULL: use getInterface(type)
          // to get the reference
        t_InterfaceProperties properties;
          // interface type specific properties
          // interpreted by the session.
};

typedef sequence<t_InterfaceStruct> t_InterfaceList;

typedef unsigned long t_InterfaceIndex;
typedef sequence<t_InterfaceIndex> t_InterfaceIndexList;

// when registering multiple interfaces need to match index vs itfType &
props:
struct t_RegisterInterfaceStruct {
      t_InterfaceTypeName itfType; // set before call to registerInterfaces
      Object ref;
t_InterfaceProperties properties;  // as above
t_InterfaceIndex index;            // set on return
};

typedef sequence<t_RegisterInterfaceStruct> t_RegisterInterfaceList;

// Session Models

// t_SessionModelName name:
// defined names
//    "SPFEEServiceSessionModel"    SPFEE Service Session Model
//    "SPFEECommSessionModel"       SPFEE Communication Session Model
// No other names defined at present
//
//  (previous  versions  of  Ret-RP  used  "SPFEESessionModel"  and  "SPFEE
//    Session Model" for the SPFEE Service Session Model (previously named
//    SPFEE Session Model)

typedef string t_SessionModelName;

typedef sequence<t_SessionModelName> t_SessionModelNameList;

// t_SessionModelProperties properties:
//
// t_SessionModelName name: "SPFEEServiceSessionModel"
// defined Property names:
// name:       "FEATURE SETS"
```

```
// value:      t_FeatureSetList
// No other names defined at present

// t_SessionModelName name: "SPFEECommSessionModel"
// defined Property names:
// name:       "FEATURE SETS"
// value:      t_FeatureSetList
//
// No feature sets are defined for the TIACommSessionModel at present.
//
// No other names defined at present

typedef t_PropertyList t_SessionModelProperties;

struct t_SessionModel {
   t_SessionModelName name;                // reserved names defined below
   t_SessionModelProperties properties;    // properties defined above
};

typedef sequence<t_SessionModel> t_SessionModelList;

enum t_WhichSessionModels {
   NoSessionModels,    // can ignore all the SessionModels
   SomeSessionModels,  // match at least one SessionModel (name & value)
   SomeSessionModelsNamesOnly,     // check name only (ignore value)
   AllSessionModels,   // match all SessionModels (name & value)
   AllSessionModelsNamesOnly   // check name only (ignore value)
};

struct t_SessionModelReq {
   t_WhichSessionModels which;
   t_SessionModelList sessionModels;
};

typedef string t_FeatureSetName;

struct t_FeatureSet {
   t_FeatureSetName name;
   t_InterfaceList itfs;
                // can return ref or NULL
};

typedef sequence<t_FeatureSet> t_FeatureSetList;

// User Info

typedef Istring t_UserId;
typedef Istring t_UserName;
typedef t_Property t_UserProperty;
typedef t_PropertyList t_UserProperties;
// Property Names defined for t_UserProperties:
// name:  "PASSWORD"
// value: string
// use:   user password, as a string.

// name:  "SecurityContext"
// value: opaque
// use:   to carry a retailer specific security context
//        e.g. could be used for an encoded user password.

struct t_UserDetails {
   t_UserId id;
   t_UserProperties properties;
};
```

```
typedef Istring t_UserCtxtName;
typedef sequence<t_UserCtxtName> t_UserCtxtNameList;

typedef sequence<octet, 16> t_ParticipantSecretId;
typedef t_ElId t_PartyId;    // corresponds to SpfeeParty enum t_ElTypes
typedef sequence<t_PartyId> t_PartyIdList;

// SessionId
// A SessionId is generated by a UA when a new session is started.
// This Id is unique within a UA, and can be used to identify a
// session to the UA.
// User's joining a session will have a different SessionId generated
// by their UA for the session.

typedef unsigned long t_SessionId;
typedef sequence<t_SessionId> t_SessionIdList;
typedef t_PropertyList t_SessionProperties;

// Invitation and Announcements
//
enum t_InvitationReplyCodes { // Based on MMUSIC replys
   SUCCESS,            // user agrees to participate
   UNSUCCESSFUL,             // couldn't contact user
   DECLINE,            // user declines
   UNKNOWN,            // user is unknown
   ERROR,              // for some unknown reason
   FORBIDDEN,                // authorisation failure
   RINGING,            // user is being contacted
   TRYING,                   // some further action is being taken
   STORED,                   // invitation is stored
   REDIRECT,           // try this different address
   NEGOTIATE,                // alternatives described in properties
      // Not MMUSIC replys, can be treated as UNSUCCESSFUL
   BUSY,               // couldn't contact because busy
   TIMEOUT                   // timed out while trying to contact
};

typedef t_PropertyList t_InvitationReplyProperties;

struct t_InvitationReply {
   t_InvitationReplyCodes reply;
   t_InvitationReplyProperties properties;
};

typedef t_PropertyList t_AnnouncementProperties;

struct t_SessionAnnouncement {
   t_AnnouncementProperties properties;
};

typedef sequence<t_SessionAnnouncement> t_AnnouncementList;

// Exceptions

enum t_PropertyErrorCode {
   UnknownPropertyError,
   InvalidProperty,
   // UnknownPropertyName: If the server receives a property name
   // it doesnot know, it can raise an exception, using this code.
   // However, servers may decide to ignore a property with an
   // unknown property name, and not raise an exception.
   UnknownPropertyName,
   InvalidPropertyName,
   InvalidPropertyValue,
   NoPropertyError           // the Property is not in error
};
```

```
// defined so it can be used in other exceptions
struct t_PropertyErrorStruct {
   t_PropertyErrorCode errorCode;
   t_PropertyName name;
   t_PropertyValue value;
};

exception e_PropertyError {
   t_PropertyErrorCode errorCode;
   t_PropertyName name;
   t_PropertyValue value;
};

enum t_InterfacesErrorCode {
   UnknownInterfacesError,
   InvalidInterfaceTypeName, // Thats not a valid i/f type name
   InvalidInterfaceRef,
   InvalidInterfaceProperty,
   InvalidInterfaceIndex
};

// must remain consistent with e_InterfacesError
struct t_InterfacesErrorStruct {
   t_InterfacesErrorCode errorCode;
   t_InterfaceTypeName itfType;
   t_PropertyErrorStruct propertyError;
      //PropertyError, if errorCode= InvalidInterfaceProperty
};

exception e_InterfacesError {
   t_InterfacesErrorCode errorCode;
   t_InterfaceTypeName itfType;
   t_PropertyErrorStruct propertyError;
      //PropertyError, if errorCode= InvalidInterfaceProperty
};

enum t_RegisterErrorCode {
   UnableToRegisterInterfaceType
};

exception e_RegisterError {
   t_RegisterErrorCode errorCode;
   t_InterfaceTypeName itfType;
   t_InterfaceProperties properties;
};

enum t_UnregisterErrorCode {
   UnableToUnregisterInterface
};

exception e_UnregisterError {
   t_UnregisterErrorCode errorCode;
   t_InterfaceIndexList indexes;   // can unregister multiple itfs
};

enum t_SessionModelErrorCode {
   UnknownSessionModelError,
   InvalidSessionModelName,     // Thats not a valid i/f type name
   SessionModelNotSupported,
   InvalidFeatureSetName,
   FeatureSetNotSupported,
   InvalidFeatureSetInterfaceType
};
```

```
exception e_SessionModelError {
   t_SessionModelErrorCode errorCode;
   t_SessionModelName sessionModelName;
   t_FeatureSetName featureSetName;// Only for FeatureSet errs
   t_InterfaceTypeName itfType;    // Only for FeatureSet errs
};

enum t_UserDetailsErrorCode {
   InvalidUserName,
   InvalidUserProperty
};

exception e_UserDetailsError {
   t_UserDetailsErrorCode errorCode;
   t_UserName name;
   t_PropertyErrorStruct propertyError;
           // Return the properties in error
};

enum t_ListErrorCode {
   ListUnavailable
};

exception e_ListError {
   t_ListErrorCode errorCode;
};

enum t_InvitationReplyErrorCode {
   InvalidInvitationReplyCode,
   InvitationReplyPropertyError
};

exception e_InvitationReplyError {
   t_InvitationReplyErrorCode errorCode;
   t_PropertyErrorStruct propertyError;
};

}; // end module SPFEECommonTypes

#endif
```

9.1.2 SPFEEAccessCommonTypes.idl

```
// File SPFEEAccessCommonTypes.idl

#ifndef spfeeaccesscommontypes_idl
#define spfeeaccesscommontypes_idl

#include "SPFEECommonTypes.idl"

module SPFEEAccessCommonTypes {

// User Info

// The following Login-Password combination may be used
// for non-CORBA Security-compliant systems, which still
// relies on a traditional, login name & password combination.
// It is mainly provided for compability reasons for the legacy
// systems, and is not expected to be used with the CORBA compliant
// part at the same time.
// legacy authentication

typedef SPFEECommonTypes::Istring t_UserPassword;
```

```
struct t_UserInfo {
   SPFEECommonTypes::t_UserId userId;
   SPFEECommonTypes::t_UserName name;
   SPFEECommonTypes::t_UserProperties userProperties;
};

// Access Session

typedef sequence <octet, 16> t_AccessSessionSecretId;
   // 128b array generated by Retailer(should be self checking)
   // (is big enough to hold the GUID favored by DCE & DCOM)
   // (Globally Unique IDentifier)

typedef unsigned long t_AccessSessionId;

enum t_WhichAccessSession {
   CurrentAccessSession,
   SpecifiedAccessSessions,
   AllAccessSessions
};

typedef sequence<t_AccessSessionId> t_AccessSessionIdList;

// Implementation Note:
// Orbix does not allow the creator of a union to set the
// discriminator (switch tag). If true, this union requires
// dummy cases for the other enums of t_WhichAccessSession.

union t_SpecifiedAccessSession switch (t_WhichAccessSession) {
   case SpecifiedAccessSessions: t_AccessSessionIdList asIdList;
   case CurrentAccessSession: octet dummy1;
   case AllAccessSessions: octet dummy2;
           // dummy var's values should not be processed
};

typedef SPFEECommonTypes::t_PropertyList t_AccessSessionProperties;

struct t_AccessSessionInfo {
   t_AccessSessionId id;
   SPFEECommonTypes::t_UserCtxtName ctxtName;
   t_AccessSessionProperties properties;
};

typedef sequence<t_AccessSessionInfo> t_AccessSessionList;

// Terminal Info

typedef string t_TerminalId;
typedef sequence<string> t_NAPId;


typedef sequence<t_NAPId> t_NAPIdList;

typedef string t_NAPType;

typedef SPFEECommonTypes::t_PropertyList t_TerminalProperties;

// t_TerminalProperties properties:
//
// defined Property names:
// name:   "TERMINAL INFO"
// value:  t_TerminalInfo
```

```
// name:   "APPLICATION INFO LIST"
// value:  t_ApplicationInfoList
// Applications on the terminal

// No other names defined at present

// t_TermType
// DESCRIPTION:
// List of terminal types.
// COMMENTS:
//  - This list can be expanded.

enum t_TerminalType {
   PersonalComputer, WorkStation, TVset,
   Videotelephone, Cellularphone, PBX, VideoServer,
   VideoBridge, Telephone, G4Fax
};

// t_TermInfo
// DESCRIPTION:
// This structure contains information related to a specific terminal
// COMMENTS:
// To be defined further.

struct t_TerminalInfo {
   t_TerminalType terminalType;
   string operatingSystem; // includes the version
   SPFEECommonTypes::t_PropertyList networkCards;
   SPFEECommonTypes::t_PropertyList devices;
   unsigned short maxConnections;
   unsigned short memorySize;
   unsigned short diskCapacity;
};

// Provider Agent Context

struct t_TerminalConfig {
   t_TerminalId terminalId;
   t_TerminalType terminalType;
   t_NAPId napId;
   t_NAPType napType;
   t_TerminalProperties properties;
};

// Service Types
//

typedef unsigned long t_ServiceId;

typedef sequence<t_ServiceId> t_ServiceIdList;

typedef SPFEECommonTypes::Istring t_UserServiceName;

typedef SPFEECommonTypes::t_PropertyList t_ServiceProperties;

struct t_ServiceInfo {
   t_ServiceId id;
   t_UserServiceName name;
   t_ServiceProperties properties;
};

typedef sequence<t_ServiceInfo> t_ServiceList;
```

```
// Session State
// State of the session as seen from the users point of view

enum t_UserSessionState {
   UserUnknownSessionState,    // Session State is not known
   UserActiveSession,
   UserSuspendedSession,     // Session has been suspended
   UserSuspendedParticipation,     // User has suspendedParticipation
          // but is continuing in his absence.
          // (may have been quit subsequently)
   UserInvited,        // User has been invited to join
   UserNotParticipating            // User is not in the session
};

// Session Info

typedef SPFEECommonTypes::Istring t_SessionPurpose;

struct t_SessionOrigin {
   SPFEECommonTypes::t_UserId userId; // user creating the session
   SPFEECommonTypes::t_SessionId sessionId;
           // id (unique to originating user)
};

struct t_SessionInfo {
   SPFEECommonTypes::t_SessionId id;     // my session id,
                // unique to UA. (scope by UA).
   t_SessionPurpose purpose;

   SPFEECommonTypes::t_ParticipantSecretId secretId;
   SPFEECommonTypes::t_PartyId myPartyId;
   t_UserSessionState state;

   SPFEECommonTypes::t_InterfaceList itfs;
   SPFEECommonTypes::t_SessionModelList sessionModels;
   SPFEECommonTypes::t_SessionProperties properties;
};

// for listing active/suspended sessions
typedef sequence<t_SessionInfo> t_SessionList;

// Invitations and Announcements.

typedef unsigned long t_InvitationId;
typedef SPFEECommonTypes::Istring t_InvitationReason;

struct t_InvitationOrigin {
   SPFEECommonTypes::t_UserId userId;    // user creating the invitation
   SPFEECommonTypes::t_SessionId sessionId;
      // so they which session they invited you from, if you contact them
};

struct t_SessionInvitation {
   t_InvitationId id;

   SPFEECommonTypes::t_UserId inviteeId;   // id of invited user, so you
                                           can check
                // the invitation was for you.
t_SessionPurpose purpose;
t_ServiceInfo serviceInfo;
t_InvitationReason reason;
t_InvitationOrigin origin;
SPFEECommonTypes::t_PropertyList invProperties;
};
```

```
typedef sequence<t_SessionInvitation> t_InvitationList;

typedef unsigned long t_AnnouncementId;

// Start Service Properties and  Application Info.

typedef SPFEECommonTypes::Istring t_AppName;
typedef SPFEECommonTypes::Istring t_AppVersion;
typedef SPFEECommonTypes::Istring t_AppSerialNum;
typedef SPFEECommonTypes::Istring t_AppLicenceNum;

struct t_ApplicationInfo {
   t_AppName name;
   t_AppVersion version;
   t_AppSerialNum serialNum;
   t_AppLicenceNum licenceNum;
   SPFEECommonTypes::t_PropertyList properties;
   SPFEECommonTypes::t_InterfaceList itfs;
   SPFEECommonTypes::t_SessionModelList sessionModels;
};

// t_StartServiceUAProperties properties:
// properties to be interpreted by the User Agent, when starting a service
//
// defined Property names:
// None defined at present
typedef SPFEECommonTypes::t_PropertyList t_StartServiceUAProperties;

// t_StartServiceSSProperties properties:
// properties to be interpreted by the Service Session, when starting a
service
//
// defined Property names:
// None defined at present
typedef SPFEECommonTypes::t_PropertyList t_StartServiceSSProperties;


// Exceptions

enum t_AccessErrorCode {
        UnknownAccessError,
        InvalidAccessSessionSecretId,
        AccessDenied,
        SecurityContextNotSatisfied
};

exception e_AccessError {
        t_AccessErrorCode errorCode;
};

enum t_UserPropertiesErrorCode {
   InvalidUserPropertyName,
   InvalidUserPropertyValue
};

exception e_UserPropertiesError {
   t_UserPropertiesErrorCode errorCode;
   SPFEECommonTypes::t_UserProperty userProperty;
};
```

```
enum t_SpecifiedAccessSessionErrorCode {
   UnknownSpecifiedAccessSessionError,
   InvalidWhichAccessSession,
   InvalidAccessSessionId
};

exception e_SpecifiedAccessSessionError {
   t_SpecifiedAccessSessionErrorCode errorCode;
   t_AccessSessionId id;           // Invalid AccessSessionId
};

enum t_InvitationErrorCode {
   InvalidInvitationId
};

exception e_InvitationError {
   t_InvitationErrorCode errorCode;
};
};
#endif
```

9.2 User and Provider General IDLs

9.2.1 SPFEEUserInitial.idl

```
// File SPFEEUserInitial.idl
#ifndef spfeeuserinitial_idl
#define spfeeuserinitial_idl

#include "SPFEECommonTypes.idl"
#include "SPFEEAccessCommonTypes.idl"

module SPFEEUserInitial {

// requestAccess() types

typedef string t_ProviderId;

// inviteUserWithoutAccessSession() types


enum t_AccessReplyCodes {
   SUCCESS,      // user agrees to initiate an access session
   DECLINE,      // user declines to initiate an access session
   FAILED,       // for some unknown reason
   FORBIDDEN     // authorisation failure
};

typedef SPFEECommonTypes::t_PropertyList t_AccessReplyProperties;

struct t_AccessReply {
   t_AccessReplyCodes reply;
   t_AccessReplyProperties properties;
};

interface i_UserInitial
{
// behaviour
// behaviourText
// "This interface is provided to allow a Provider to invite
// a user to a session outside of an access session; and request
// the establishment of an access session";
// usage
// " ";
```

```
    void requestAccess (
        in t_ProviderId providerId,
        out t_AccessReply reply
    );

    void inviteUserOutsideAccessSession (
        in t_ProviderId providerId,
        in SPFEEAccessCommonTypes::t_SessionInvitation invitation,
        out SPFEECommonTypes::t_InvitationReply reply
    );

    void cancelInviteUserOutsideAccessSession (
        in t_ProviderId providerId,
        in SPFEEAccessCommonTypes::t_InvitationId id
    ) raises (
        SPFEEAccessCommonTypes::e_InvitationError
    );

};   // i_UserInitial
};
#endif
```

9.2.2 SPFEEUserAccess.idl

```
// File SPFEEUserAccess.idl

#ifndef spfeeuseraccess_idl
#define spfeeuseraccess_idl

#include "SPFEECommonTypes.idl"
#include "SPFEEAccessCommonTypes.idl"

module SPFEEUserAccess {

typedef SPFEECommonTypes::t_PropertyList t_CancelAccessSessionProperties;

interface i_UserAccessGetInterfaces {

// behaviour
// behaviourText
// "This interface allows the provider  domain to get
// interfaces exported by this user domain."

// usage
// "This interface is not to be exported across
// Ret RP. It is inherited into the exported interfaces."

    void getInterfaceTypes (
        out SPFEECommonTypes::t_InterfaceTypeList itfTypes
    ) raises (
        SPFEECommonTypes::e_ListError
    );

    void getInterface (
        in SPFEECommonTypes::t_InterfaceTypeName itfType,
        in SPFEECommonTypes::t_MatchProperties desiredProperties,
        out SPFEECommonTypes::t_InterfaceStruct itf
    ) raises (
        SPFEECommonTypes::e_InterfacesError,
        SPFEECommonTypes::e_PropertyError
    );
```

```
   void getInterfaces (
      out SPFEECommonTypes::t_InterfaceList itfs
   ) raises (
      SPFEECommonTypes::e_ListError
   );

}; // i_UserAccessGetInterfaces

interface i_UserAccess
   : i_UserAccessGetInterfaces
{
// behaviour
// behaviourText
// "This interface is provided to a UA to perform actions during an
// access session. It inherits from i_UserAccessGetInterfaces to
// allow the retailer to ask for other interfaces exported by the
// consumer domain. (The retailer cannot register his own // // /
// interfaces!)";
   // usage
   // " ";


// DRAFT: this operation is draft only, any feedback on this operation is
//        most welcome.
       void cancelAccessSession(
          in t_CancelAccessSessionProperties options
   );

}; // i_UserAccess

interface i_UserInvite
{
// behaviour
// behaviourText
// "This interface is provided to a UA, to invite the user to:
//     - join a service session
//     - request an access session
// ";
// usage
//  " ";

// inviteUser() types

   void inviteUser (
      in  SPFEEAccessCommonTypes::t_SessionInvitation invitation,
      out SPFEECommonTypes::t_InvitationReply reply
      )
   raises (SPFEEAccessCommonTypes::e_InvitationError)
   ;

   void cancelInviteUser (
      in SPFEECommonTypes::t_UserId inviteeId,
      in SPFEEAccessCommonTypes::t_InvitationId id
   ) raises (
      SPFEEAccessCommonTypes::e_InvitationError
   );

}; // i_UserInvite


interface i_UserTerminal {
```

```
// behaviour
// behaviourText
// "This interface is provided to a UA, to gain information about the
// terminal context.";
   // usage
   // " ";

// getTerminalInfo() types


   void getTerminalInfo(
      out SPFEEAccessCommonTypes::t_TerminalInfo terminalInfo
   );
}; // i_UserTerminal

interface i_UserAccessSessionInfo
{
// ...AccessSessionInfo() types

oneway void newAccessSessionInfo (
       in SPFEEAccessCommonTypes::t_AccessSessionInfo accessSession
   );

oneway void endAccessSessionInfo (
       in SPFEEAccessCommonTypes::t_AccessSessionId asId
   );

oneway void cancelAccessSessionInfo (
       in SPFEEAccessCommonTypes::t_AccessSessionId asId
   );

oneway void newSubscribedServicesInfo (
       in SPFEEAccessCommonTypes::t_ServiceList services
   );

}; // i_UserAccessSessionInfo

interface i_UserSessionInfo
{
oneway void newSessionInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );

oneway void endSessionInfo (
       in SPFEECommonTypes::t_SessionId sessionId
   );

oneway void endMyParticipationInfo (
       in SPFEECommonTypes::t_SessionId sessionId
   );

oneway void suspendSessionInfo (
       in SPFEECommonTypes::t_SessionId sessionId
   );

oneway void suspendMyParticipationInfo (
       in SPFEECommonTypes::t_SessionId sessionId
   );

oneway void resumeSessionInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );
```

```
oneway void resumeMyParticipationInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );

oneway void joinSessionInfo (
       in SPFEEAccessCommonTypes::t_SessionInfo session
   );

}; // i_UserSessionInfo

};

#endif
```

9.2.3 SPFEEProviderInitial.idl

```
// File SPFEEProviderInitial.idl
#ifndef spfeeproviderinitial_idl
#define spfeeproviderinitial_idl

#include "SPFEECommonTypes.idl"
#include "SPFEEAccessCommonTypes.idl"

module SPFEEProviderInitial {

enum t_AuthenticationStatus {
   SecAuthSuccess,
   SecAuthFailure,
   SecAuthContinue,
   SecAuthExpired
};

typedef unsigned long t_AuthMethod;


typedef SPFEECommonTypes::t_PropertyList t_AuthMethodProperties;
typedef SPFEECommonTypes::t_MatchProperties t_AuthMethodSearchProperties;

struct t_AuthMethodDesc {
   t_AuthMethod method;
   t_AuthMethodProperties properties;
};

typedef sequence<t_AuthMethodDesc> t_AuthMethodDescList;

exception e_AuthMethodNotSupported {
   // removed t_AuthMethodDescList authMethods;
};

exception e_AccessNotPossible {
};

exception e_AuthenticationError {
   SPFEECommonTypes::Istring sIOR;
};

exception e_AuthMethodPropertiesError {
   SPFEECommonTypes::t_PropertyErrorStruct propertyError;
};

interface i_ProviderInitial {
```

```
// behaviour
// behaviourText
// " A reference to an interface of this type is returned to the PA
// when it has authenticated (or requires no authentication)
// to obtain specific userAgent interfaces.";
// usage
// "to obtain a userAgent reference according to the business
// needs of the consumer";

// requestNamedAccess() types


// Operation 'requestNamedAccess()'
// Used when the user is known to the provider and has already been
// authenticated, either by DPE security or by authenticate()
// input:
// userId: (user name identifying requested user agent.)
//    user_name="anonymous" for anonymous access.
//    user_name may be an empty string, if the provider is
//    using userProperties to identify the          user.
// userProperties: PropertyList which can be used to send
//    additional provider specific user privilege
//    information. This is generic, and can be used to send
//    any type of info to the provider
// output:
// i_nameduaAccess: return: Interface reference of the UserAgent.
// accessSessionId: Identifies the access session the operation is
//    associated with. Must be supplied in all subsequent
//    operations with the InitialAgent and UserAgent.

   void requestNamedAccess (
      in SPFEECommonTypes::t_UserId userId,
      in SPFEECommonTypes::t_UserProperties userProperties,
      out Object  namedAccessIR,           // type: i_ProviderNamedAccess
      out SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      out SPFEEAccessCommonTypes::t_AccessSessionId asId
   ) raises (
      e_AccessNotPossible,
      e_AuthenticationError,
      SPFEEAccessCommonTypes::e_UserPropertiesError
   );

// Operation 'requestAnonymousAccess()'
// Used when the user wants to access anonymously to the provider.
// A secure session may already be established by DPE security or by
// authenticate() (the laater does not mean the user is known to the
// provider if a third party authentication protocol is used.)
// input:
// userProperties: may be a null list
// output:  as request_access

   void requestAnonymousAccess (
      in SPFEECommonTypes::t_UserProperties userProperties,
      out Object anonAccessIR, // type: i_ProviderAnonAccess
      out SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      out SPFEEAccessCommonTypes::t_AccessSessionId asId
   ) raises (
      e_AccessNotPossible,
      e_AuthenticationError,
      SPFEEAccessCommonTypes::e_UserPropertiesError
   );

}; // i_ProviderInitial
```

```
interface i_ProviderAuthenticate {

// behaviour
// behaviourText
// " A reference to an interface of this type is returned to the PA
// when it wishes to choose this route to authenticate
// itself/mutually to the provider. ";
// usage
// "to agree authentication options supported, acquire
// privelege attributes for the consumer and establish
// an access session between the consumer and the provider";

// getAuthenticationMethods() types


typedef sequence<octet> t_opaque;

//Operations 'getAuthenticationMethods ()'
//input:
// property list used to filter output
//output:
// list of available authentication configurations

   void getAuthenticationMethods (
      in  t_AuthMethodSearchProperties desiredProperties,
      out t_AuthMethodDescList authMethods
   ) raises (
      e_AuthMethodPropertiesError,
      SPFEECommonTypes::e_ListError
   );

// Operation 'authenticate()'
// Used to authenticate a consumer attempting to gain access to a
// user agent. invocation is a prerequsite to establishing client /
// side credentails for establishing secure bindings unless
// an alternative route is used
//input:
// Method: used to identify the authentication method proposed by
//    the client, reflects the composition and generation
//    method of other opaque data
// securityName: name assumed by consumer for authentication. may be
//    null accroding to the authentication method used.
// authenData: opaque data containing consumer attributes to be
//    authenticated
// privAttribReq: opaque specification of the privileges requested
//    by the consumer to create credential for subsequent
//    interactions.
//output:
// privAttrib: privilege attributes returned in response to request.
// continuationData: contains challenge data for the client if the
//    authentication method requires continuation of the
//    protocol
// authSpecificData: data specific to the authentication service
//    used.
// raises:
// e_AuthMethodNotSupported: when the authentication mechanism used
//    by client is not supported by i_iaAuthenticate
```

```
    void authenticate(
        in t_AuthMethod authMethod,
        in string securityName,
        in t_opaque authenData,
        in t_opaque privAttribReq,
        out t_opaque privAttrib,
        out t_opaque continuationData,
        out t_opaque authSpecificData,
        out t_AuthenticationStatus authStatus
    ) raises (
        e_AuthMethodNotSupported
    );

// Operation continue_authentication ()'
// To complete an authentication protocol initiated by authenticate.
// used for second and subsequent continuations.
// input:
// responseData: response from the client to the continuationData
//     output from the to authenticate() or previous calls to
//     continue_authenticate()
// output:
// continuation_data
//     as per authenticate, if continuation is necessary.
// credential_data:
//     as per authenticate, initialiation values or extra
//     items.

    void continueAuthentication(
        in t_opaque responseData,
        out t_opaque privAttrib,
        out t_opaque continuationData,
        out t_opaque authSpecificData,
        out t_AuthenticationStatus authStatus
    );

}; // i_ProviderAuthenticate

};

#endif
```

9.2.4 SPFEEProviderAccess.idl

```
// File SPFEEProviderAccess.idl

#ifndef spfeeprovideraccess_idl
#define spfeeprovideraccess_idl

#include "SPFEECommonTypes.idl"
#include "SPFEEAccessCommonTypes.idl"

#include "SPFEEUserInitial.idl"

module SPFEEProviderAccess {


typedef string t_DateTimeRegistered;     // DRAFT ONLY

struct t_RegisteredInterfaceStruct {
    SPFEECommonTypes::t_InterfaceIndex index;
    SPFEECommonTypes::t_InterfaceStruct interfaceStruct;
        // DateTimeRegistered DRAFT ONLY: need some info on when
        // interface was registered.
    t_DateTimeRegistered when;
```

```
    SPFEECommonTypes::t_UserCtxtName where;
};

typedef sequence<t_RegisteredInterfaceStruct> t_RegisteredInterfaceList;


struct t_UserCtxt {
    SPFEECommonTypes::t_UserCtxtName ctxtName;
    SPFEEAccessCommonTypes::t_AccessSessionId asId;
    Object accessIR;          // type: i_UserAccess
    Object terminalIR;        // type: i_UserTerminal
    Object inviteIR;          // type: i_UserInvite
    Object sessionInfoIR;           // type: i_UserSessionInfo
    SPFEEAccessCommonTypes::t_TerminalConfig terminalConfig;
};


typedef sequence<t_UserCtxt> t_UserCtxtList;

enum t_WhichUserCtxt {
    CurrentUserCtxt,
    SpecifiedUserCtxts,
    AllUserCtxts
};

// Implementation Note:
// Orbix does not allow the creator of a union to set the
// discriminator (switch tag). If true, this union requires
// dummy cases for the other enums of t_WhichUserCtxt.

union t_SpecifiedUserCtxt switch (t_WhichUserCtxt) {
   case SpecifiedUserCtxts: SPFEECommonTypes::t_UserCtxtNameList ctxtNames;
   case CurrentUserCtxt: octet dummy1;
   case AllUserCtxts: octet dummy2;       // value should not be processed
};


typedef SPFEECommonTypes::t_MatchProperties t_DiscoverServiceProperties;
typedef SPFEECommonTypes::t_MatchProperties t_SubscribedServiceProperties;

typedef SPFEECommonTypes::t_MatchProperties t_SessionSearchProperties;
typedef SPFEECommonTypes::t_MatchProperties t_AnnouncementSearchProperties;

enum t_EndAccessSessionOption {
    DefaultEndAccessSessionOption,
    SuspendActiveSessions,
    SuspendMyParticipationActiveSessions,
    EndActiveSessions,
    EndMyParticipationActiveSessions,
    EndAllSessions,
    EndMyParticipationAllSessions
};


typedef SPFEEAccessCommonTypes::t_AppName t_AppName;
typedef SPFEEAccessCommonTypes::t_AppVersion t_AppVersion;
typedef SPFEEAccessCommonTypes::t_AppSerialNum t_AppSerialNum;
typedef SPFEEAccessCommonTypes::t_AppLicenceNum t_AppLicenceNum;

typedef SPFEEAccessCommonTypes::t_ApplicationInfo t_ApplicationInfo;

typedef                 SPFEEAccessCommonTypes::t_StartServiceUAProperties
t_StartServiceUAProperties;
typedef                 SPFEEAccessCommonTypes::t_StartServiceSSProperties
t_StartServiceSSProperties;
```

```
// Exceptions

enum t_SessionErrorCode {
   UnknownSessionError,
   InvalidSessionId,
   SessionDoesNotExist,
   InvalidUserSessionState,
   SessionNotAllowed,
   SessionNotAccepted,
   SessionOpNotSupported
};

exception e_SessionError {
   t_SessionErrorCode errorCode;
   SPFEEAccessCommonTypes::t_UserSessionState state;
};

enum t_UserCtxtErrorCode {
   InvalidUserCtxtName,
   InvalidUserAccessIR,
   InvalidUserTerminalIR,
   InvalidUserInviteIR,
   InvalidTerminalId,
   InvalidTerminalType,
   InvalidNAPId,
   InvalidNAPType,
   InvalidTerminalProperty,
   UserCtxtNotAvailable
};

exception e_UserCtxtError {
   t_UserCtxtErrorCode errorCode;
   SPFEECommonTypes::t_UserCtxtName ctxtName;
   SPFEECommonTypes::t_PropertyErrorStruct propertyError;
      //PropertyError, if errorCode= InvalidTerminalProperty
};

enum t_RegisterUserCtxtErrorCode {
   UnableToRegisterUserCtxt
};

exception e_RegisterUserCtxtError {
   t_RegisterUserCtxtErrorCode errorCode;
};

enum t_EndAccessSessionErrorCode {
   EASE_UnknownError,
   EASE_InvalidOption,
   EASE_ActiveSession,
   EASE_SuspendedSession,
   EASE_SuspendedParticipation
};

exception e_EndAccessSessionError {
   t_EndAccessSessionErrorCode errorCode;
           // sessions causing a problem.
   SPFEECommonTypes::t_SessionIdList sessions;
};

// ExceptionCodes t_ServiceErrorCode;
// Example of Exception codes definition
enum t_ServiceErrorCode {
   InvalidServiceId,
   ServiceUnavailable,
```

```
   SessionCreationDenied,
   SessionNotPossibleDueToUserCtxt
   };

exception e_ServiceError {
   t_ServiceErrorCode errorCode;
};

//  e_StartServiceUAPropertyError & e_StartServiceSSPropertyError
//  are defined to distinguish property errors in
//  t_StartServiceUAProperties & t_StartServiceUAProperties respectively
exception e_StartServiceUAPropertyError {
    // use the errorCodes as for e_PropertyError
    SPFEECommonTypes::t_PropertyErrorStruct propertyError;
};

exception e_StartServiceSSPropertyError {
   // use the errorCodes as for e_PropertyError
   SPFEECommonTypes::t_PropertyErrorStruct propertyError;
};

enum t_ApplicationInfoErrorCode {
   UnknownAppInfoError,
   InvalidApplication,    // Can't use this application with this
          // service/session
   InvalidAppInfo,
   UnknownAppName,                 // I didn't recognise your app name
   InvalidAppName,                 // I don't understand your app name
          // (eg. badly formatted)
   UnknownAppVersion,
   InvalidAppVersion,
   InvalidAppSerialNum,
   InvalidAppLicenceNum,
   AppPropertyError,
   AppSessionInterfacesError,
   AppSessionModelsError,
   AppSIDescError
};

exception e_ApplicationInfoError {
   t_ApplicationInfoErrorCode errorCode;

// t_PropertyErrorStruct:
// Contains the t_PropertyName and t_PropertyValue in error,
// (if t_ApplicationInfoError.errorCode==AppPropertyError
// OR t_PropertyErrorStruct.errorCode==NoPropertyError),
// if error is not due to a property

   SPFEECommonTypes::t_PropertyErrorStruct propertyError;

// t_SessionInterfacesErrorStruct
// Only used if:
// t_AppInfoError.errorCode == AppIntRefInfoError

   SPFEECommonTypes::t_InterfacesErrorStruct itfsError;
};


enum t_AnnouncementErrorCode {
   InvalidAnnouncementId
};
```

```
exception e_AnnouncementError {
   t_AnnouncementErrorCode errorCode;
};

interface i_ProviderAccessGetInterfaces {

// behaviour
// behaviourText
// "This interface allows the user domain to get
// interfaces exported by this provider domain."

// usage
// "This interface is not to be exported across
// Ret RP. It is inherited into the exported interfaces."

   void getInterfaceTypes (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      out SPFEECommonTypes::t_InterfaceTypeList itfTypes
   ) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEECommonTypes::e_ListError
   );

   void getInterface (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEECommonTypes::t_InterfaceTypeName itfType,
      in SPFEECommonTypes::t_MatchProperties desiredProperties,
      out SPFEECommonTypes::t_InterfaceStruct itf
   ) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEECommonTypes::e_InterfacesError,
      SPFEECommonTypes::e_PropertyError
   );

   void getInterfaces (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEECommonTypes::t_MatchProperties desiredProperties,
      out SPFEECommonTypes::t_InterfaceList itfs
   ) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEECommonTypes::e_PropertyError,
      SPFEECommonTypes::e_ListError
   );

   }; // i_ProviderAccessGetInterfaces

   interface i_ProviderAccessRegisterInterfaces {

   // behaviour
   // behaviourText
   // "This interface allows the client to register interfaces
   // exported by the client domain."

   // usage
   // "This interface is not to be exported across Ret RP.
   // It is inherited into the exported interfaces."


         // register interfaces to be used only during
         // current access session

      void registerInterface (
         in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
         inout SPFEECommonTypes::t_RegisterInterfaceStruct itf
      ) raises (
         SPFEEAccessCommonTypes::e_AccessError,
```

```
            SPFEECommonTypes::e_InterfacesError,
            SPFEECommonTypes::e_RegisterError
        );

        void registerInterfaces (
            in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            inout SPFEECommonTypes::t_RegisterInterfaceList itfs
        ) raises (
            SPFEEAccessCommonTypes::e_AccessError,
            SPFEECommonTypes::e_InterfacesError,
            SPFEECommonTypes::e_RegisterError
        );

            // register interfaces which will be accessible
            // outside the access session.

        void registerInterfaceOutsideAccessSession (
            in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            inout SPFEECommonTypes::t_RegisterInterfaceStruct itf
        ) raises (
            SPFEEAccessCommonTypes::e_AccessError,
            SPFEECommonTypes::e_InterfacesError,
            SPFEECommonTypes::e_RegisterError
        );

        void registerInterfacesOutsideAccessSession (
            in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            inout SPFEECommonTypes::t_RegisterInterfaceList itfs
        ) raises (
            SPFEEAccessCommonTypes::e_AccessError,
            SPFEECommonTypes::e_InterfacesError,
            SPFEECommonTypes::e_RegisterError
        );

        void listRegisteredInterfaces (
            in SPFEEAccessCommonTypes:: t_AccessSessionSecretId asSecretId,
            in SPFEEAccessCommonTypes::t_SpecifiedAccessSession as,
            out t_RegisteredInterfaceList itfs
        ) raises (
            SPFEEAccessCommonTypes::e_AccessError,
            SPFEEAccessCommonTypes::e_SpecifiedAccessSessionError,
            SPFEECommonTypes::e_ListError
        );
        void unregisterInterface (
            in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in SPFEECommonTypes::t_InterfaceIndex index
        ) raises (
            SPFEEAccessCommonTypes::e_AccessError,
            SPFEECommonTypes::e_InterfacesError,
            SPFEECommonTypes::e_UnregisterError
        );

        void unregisterInterfaces (
            in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
            in SPFEECommonTypes::t_InterfaceIndexList indexes
        ) raises (
            SPFEEAccessCommonTypes::e_AccessError,
            SPFEECommonTypes::e_InterfacesError,
            SPFEECommonTypes::e_UnregisterError
        );

};  // i_ProviderAccessRegisterInterfaces
```

```
interface i_ProviderAccessInterfaces
          : i_ProviderAccessGetInterfaces,
            i_ProviderAccessRegisterInterfaces
{
// behaviour
// behaviourText
// "This interface allows the client to get interfaces
// exported by this domain, and register interfaces exported
// by the client domain."

// usage
// "This interface is not to be exported across Ret RP.
// It is inherited into the exported interfaces."

// No additional operations are defined

}; // i_ProviderAccessInterfaces

interface i_ProviderAccess: i_ProviderAccessInterfaces
{
// behaviour
// behaviourText
// "This interface is the place to put operations which should be
// shared between i_ProviderNamedAccess and i_ProviderAnonAccess.
// Currently none are defined."

// usage
// "This interface is not to be exported across Ret RP.
// It is inherited into the exported interfaces."

// No additional operations are defined

}; // interface i_ProviderAccess


interface i_ProviderNamedAccess
       : i_ProviderAccess
{

       void setUserCtxt (
          in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
          in t_UserCtxt userCtxt
       ) raises (
          SPFEEAccessCommonTypes::e_AccessError,
          e_UserCtxtError
       );

       void getUserCtxt (
          in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
          in SPFEECommonTypes::t_UserCtxtName ctxtName,
          out t_UserCtxt userCtxt
       ) raises (
          SPFEEAccessCommonTypes::e_AccessError,
          e_UserCtxtError
       );

       void getUserCtxts (
          in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
          in t_SpecifiedUserCtxt ctxt,
          out t_UserCtxtList userCtxts
       ) raises (
          SPFEEAccessCommonTypes::e_AccessError,
          e_UserCtxtError,
```

```
    SPFEECommonTypes::e_ListError
);

void getUserCtxtsAccessSessions (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in SPFEEAccessCommonTypes::t_SpecifiedAccessSession as,
    out t_UserCtxtList userCtxts
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    SPFEEAccessCommonTypes::e_SpecifiedAccessSessionError,
    SPFEECommonTypes::e_ListError
);

void registerUserCtxtsAccessSessions (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in SPFEEAccessCommonTypes::t_SpecifiedAccessSession as
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    SPFEEAccessCommonTypes::e_SpecifiedAccessSessionError,
    e_RegisterUserCtxtError
);


void listAccessSessions (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    out SPFEEAccessCommonTypes::t_AccessSessionList asList
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    SPFEECommonTypes::e_ListError
);

void endAccessSession(
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in SPFEEAccessCommonTypes::t_SpecifiedAccessSession as,
    in t_EndAccessSessionOption option
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    SPFEEAccessCommonTypes::e_SpecifiedAccessSessionError,
    e_EndAccessSessionError
);

void getUserInfo(
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    out SPFEEAccessCommonTypes::t_UserInfo userInfo
) raises (
    SPFEEAccessCommonTypes::e_AccessError
);

void listSubscribedServices (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in t_SubscribedServiceProperties desiredProperties,
    out SPFEEAccessCommonTypes::t_ServiceList services
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    SPFEECommonTypes::e_PropertyError,
    SPFEECommonTypes::e_ListError
);

void discoverServices(
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in t_DiscoverServiceProperties desiredProperties,
    in unsigned long howMany,
```

```
            out SPFEEAccessCommonTypes::t_ServiceList services,
                   // type: i_DiscoverServicesIterator
            out Object iteratorIR
         ) raises (
            SPFEEAccessCommonTypes::e_AccessError,
            SPFEECommonTypes::e_PropertyError,
            SPFEECommonTypes::e_ListError
         );

   void getServiceInfo (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEEAccessCommonTypes::t_ServiceId serviceId,
      in                   SPFEEProviderAccess::t_SubscribedServiceProperties
      desiredProperties,
      out SPFEEAccessCommonTypes::t_ServiceProperties serviceProperties
   ) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEEProviderAccess::e_ServiceError
   );

   void listRequiredServiceComponents (
      in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
      in SPFEEAccessCommonTypes::t_ServiceId serviceId,
      in SPFEEAccessCommonTypes::t_TerminalConfig terminalConfig,
      in SPFEEAccessCommonTypes::t_TerminalInfo terminalInfo,
      // Example of usage for Java applet download:
      // name-value pair describing the url of a Java applet
      // name = "URL"
      // type = "string"
      out SPFEECommonTypes::t_PropertyList locations
   ) raises (
      SPFEEAccessCommonTypes::e_AccessError,
      SPFEEProviderAccess::e_ServiceError
   );

      void listServiceSessions (
         in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
         in SPFEEAccessCommonTypes::t_SpecifiedAccessSession as,
         in t_SessionSearchProperties desiredProperties,
         out SPFEEAccessCommonTypes::t_SessionList sessions
      ) raises (
         SPFEEAccessCommonTypes::e_AccessError,
         SPFEEAccessCommonTypes::e_SpecifiedAccessSessionError,
         SPFEECommonTypes::e_PropertyError,
         SPFEECommonTypes::e_ListError
      );

      void getSessionModels (
         in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
         in SPFEECommonTypes::t_SessionId sessionId,
         out SPFEECommonTypes::t_SessionModelList sessionModels
      ) raises (
         SPFEEAccessCommonTypes::e_AccessError,
         e_SessionError,
         SPFEECommonTypes::e_ListError
      );

      void getSessionInterfaceTypes (
         in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
         in SPFEECommonTypes::t_SessionId sessionId,
         out SPFEECommonTypes::t_InterfaceTypeList itfTypes
      ) raises (
         SPFEEAccessCommonTypes::e_AccessError,
         e_SessionError,
```

```
    SPFEECommonTypes::e_ListError
);

void getSessionInterface (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in SPFEECommonTypes::t_SessionId sessionId,
    in SPFEECommonTypes::t_InterfaceTypeName itfType,
    out SPFEECommonTypes::t_InterfaceStruct itf
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    e_SessionError,
    SPFEECommonTypes::e_InterfacesError
);

void getSessionInterfaces (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in SPFEECommonTypes::t_SessionId sessionId,
    out SPFEECommonTypes::t_InterfaceList itfs
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    e_SessionError,
    SPFEECommonTypes::e_ListError
);

void listSessionInvitations (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    out SPFEEAccessCommonTypes::t_InvitationList invitations
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    SPFEECommonTypes::e_ListError
);

void listSessionAnnouncements (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in t_AnnouncementSearchProperties desiredProperties,
    out SPFEECommonTypes::t_AnnouncementList announcements
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    SPFEECommonTypes::e_PropertyError,
    SPFEECommonTypes::e_ListError
);

void startService (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in SPFEEAccessCommonTypes::t_ServiceId serviceId,
    in t_ApplicationInfo app,
    in SPFEECommonTypes::t_SessionModelReq sessionModelReq,
    in t_StartServiceUAProperties uaProperties,
    in t_StartServiceSSProperties ssProperties,
    out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
    SPFEEAccessCommonTypes::e_AccessError,
    e_ServiceError,
    e_ApplicationInfoError,
    SPFEECommonTypes::e_SessionModelError,
    e_StartServiceUAPropertyError,
    e_StartServiceSSPropertyError
);

void endSession (
    in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
    in SPFEECommonTypes::t_SessionId sessionId
```

```
) raises (
   SPFEEAccessCommonTypes::e_AccessError,
   e_SessionError
);

void endMyParticipation (
   in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
   in SPFEECommonTypes::t_SessionId sessionId
) raises (
   SPFEEAccessCommonTypes::e_AccessError,
   e_SessionError
);

void suspendSession (
   in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
   in SPFEECommonTypes::t_SessionId sessionId
) raises (
   SPFEEAccessCommonTypes::e_AccessError,
   e_SessionError
);

void suspendMyParticipation (
   in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
   in SPFEECommonTypes::t_SessionId sessionId
) raises (
   SPFEEAccessCommonTypes::e_AccessError,
   e_SessionError
);

void resumeSession (
   in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
   in SPFEECommonTypes::t_SessionId sessionId,
   in t_ApplicationInfo app,
   out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
   SPFEEAccessCommonTypes::e_AccessError,
   e_SessionError,
   e_ApplicationInfoError
);

void resumeMyParticipation (
   in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
   in SPFEECommonTypes::t_SessionId sessionId,
   in t_ApplicationInfo app,
   out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
   SPFEEAccessCommonTypes::e_AccessError,
   e_SessionError,
   e_ApplicationInfoError
);

void joinSessionWithInvitation (
   in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
   in SPFEEAccessCommonTypes::t_InvitationId invitationId,
   in t_ApplicationInfo app,
   in SPFEECommonTypes::t_PropertyList joinProperties,
   out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
) raises (
   SPFEEAccessCommonTypes::e_AccessError,
   e_SessionError,
   SPFEEAccessCommonTypes::e_InvitationError,
   e_ApplicationInfoError
);
```

```
    void joinSessionWithAnnouncement (
        in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
        in SPFEEAccessCommonTypes::t_AnnouncementId announcementId,
        in t_ApplicationInfo app,
        in SPFEECommonTypes::t_PropertyList joinProperties,
        out SPFEEAccessCommonTypes::t_SessionInfo sessionInfo
    ) raises (
        SPFEEAccessCommonTypes::e_AccessError,
        e_SessionError,
        e_AnnouncementError,
        e_ApplicationInfoError
    );

    void replyToInvitation (
        in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
        in SPFEEAccessCommonTypes::t_InvitationId invitationId,
        in SPFEECommonTypes::t_InvitationReply reply
    ) raises (
        SPFEEAccessCommonTypes::e_AccessError,
        SPFEEAccessCommonTypes::e_InvitationError,
        SPFEECommonTypes::e_InvitationReplyError
    );

}; // i_ProviderNamedAccess

interface i_ProviderAnonAccess
        : i_ProviderAccess
{
    // No additional operations defined.

}; // i_ProviderAnonAccess

interface i_DiscoverServicesIterator {

    exception e_UnknownDiscoverServicesMaxLeft {};

    void maxLeft (
        out unsigned long n
    ) raises (
        e_UnknownDiscoverServicesMaxLeft
    );

    // moreLeft: true if there are more interfaces,
    // (accessible thru the iterator (This interface)),
    // false if there are no more interfaces to retrieve

    void nextN (
        in unsigned long n,
             // n >= number returned in services
        out SPFEEAccessCommonTypes::t_ServiceList services,
        out boolean moreLeft
    ) raises (
        SPFEECommonTypes::e_ListError
    );

    void destroy ();

}; // i_DiscoverServicesIterator

}; // module SPFEEProviderAccess

#endif
```

9.3 Définitions IDL pour le point Ret-RP

```
// File SPFEERetConsumerInitial.idl
#ifndef spfeeretconsumerinitial_idl
#define spfeeretconsumerinitial_idl
#include "SPFEEUserInitial.idl"

module SPFEERetConsumerInitial {

interface i_ConsumerInitial: SPFEEUserInitial::i_UserInitial {
// No additional operations for Consumer.
};
};
#endif

// File SPFEERetConsumerAccess.idl
#ifndef spfeeretconsumeraccess_idl
#define spfeeretconsumeraccess_idl
#include "SPFEEUserAccess.idl"

module SPFEERetConsumerAccess {

interface i_ConsumerAccess: SPFEEUserAccess::i_UserAccess {
// No additional operations for Consumer.
}; // i_ConsumerAccess

interface i_ConsumerInvite: SPFEEUserAccess::i_UserInvite {
// No additional operations for Consumer.
}; // i_ConsumerInvite

interface i_ConsumerTerminal: SPFEEUserAccess::i_UserTerminal {
// No additional operations for Consumer.
}; // i_ConsumerTerminal

interface i_ConsumerAccessSessionInfo:
SPFEEUserAccess::i_UserAccessSessionInfo {
// No additional operations for Consumer.
}; // i_ConsumerAccessSessioninfo

interface i_ConsumerSessionInfo: SPFEEUserAccess::i_UserSessionInfo {
// No additional operations for Consumer.
}; // i_ConsumerSessionInfo

};

#endif


// File SPFEERetRetailerInitial.idl
#ifndef spfeeretretailerinitial_idl
#define spfeeretretailerinitial_idl

#include "SPFEEProviderInitial.idl"

module SPFEERetRetailerInitial {

interface i_RetailerInitial: SPFEEProviderInitial::i_ProviderInitial {
// No Retailer specific operations defined.
}; // i_RetailerInitial
```

```
interface i_RetailerAuthenticate:
SPFEEProviderInitial::i_ProviderAuthenticate {
// No Retailer specific operations defined.
}; // i_RetailerAuthenticate
};
#endif


// File SPFEERetRetailerAccess.idl
#ifndef spfeeretretaileraccess_idl
#define spfeeretretaileraccess_idl
#include "SPFEEProviderAccess.idl"

module SPFEERetRetailerAccess {

interface i_RetailerAccess {
// behaviour
// behaviourText
// "This interface is the place to put operations which should be
// shared between i_RetailerNamedAccess and i_RetailerAnonAccess.
// These are specific to the Ret RP, so they don't go in i_ProviderAccess
// Currently none are defined."

// usage
// "This interface is not to be exported across Ret RP.
// It is inherited into the exported interfaces."
// No retailer specific operations are defined
}; // i_RetailerAccess

interface i_RetailerNamedAccess
          :SPFEEProviderAccess::i_ProviderNamedAccess,
          i_RetailerAccess
{
   // Change the name of the interface for Ret RP.
}; // i_RetailerNamedAccess

interface i_RetailerAnonAccess
          : SPFEEProviderAccess::i_ProviderAnonAccess,
          i_RetailerAccess
{
   // Change the name of the interface for Ret RP
}; // i_RetailerAnonAccess
}; // module SPFEERetRetailerAccess
#endif
```

9.4 Spécifications IDL pour l'abonnement du point Ret-RP

9.4.1 SPFEESubCommonTypes.idl

```
// File SPFEESubCommonTypes.idl
// Contents:
// This file include common definitions required by the subscription
// magement component and its clients.
// @see SPFEECommonTypes
// @see SPFEEAccessCommonTypes
//

#ifndef SPFEESUBCOMMONTYPES_IDL
#define SPFEESUBCOMMONTYPES_IDL

// This file provides definitions that are common to the service
architecture.
#include "SPFEEAccessCommonTypes.idl"
```

```
// Module with common types definitions for subscription management.
module SPFEESubCommonTypes {

// List of Service Identifiers.
typedef sequence<SPFEEAccessCommonTypes::t_ServiceId> t_ServiceIdList;

// Service Types
typedef string t_ServiceType;

// Terminal Type: Just an example.
enum t_TermType {UndefinedTermType,PersonalComputer, WorkStation, TVset,
Videotelephone, Cellularphone, PBX, VideoServer, VideoBridge, Telephone,
G4Fax};

// NAP type: used to determine the instantation of available QoS.
enum t_NapType {UndefinedNapType,NapTypeFixed, NapTypeWireless};

// List of NAPs
typedef sequence<SPFEEAccessCommonTypes::t_NAPId> t_NAPIdList;

// Terminal presentation technology. This is just an example
enum t_PresentationSupport { UndefinedPresSupp,X11R6, WINDOWS95, MGEG };

// The Account Number represents the Subscriber identifier.
typedef string t_AccountNumber;
typedef sequence<t_AccountNumber> t_SubscriberIdList;

// Types required for SAG management

// Users, terminals and NAPs are considered (subscription) entities
enum t_entityType {user, terminal, nap};

// Entity Id allows to identity uniquely an entity inside the retailer
domain.
union t_entityId switch (t_entityType) {
        case user:     SPFEECommonTypes::t_UserId           userId;
        case terminal: SPFEEAccessCommonTypes::t_TerminalId  terminalId;
        case nap:      SPFEEAccessCommonTypes::t_NAPId       napId;
};
typedef sequence<t_entityId> t_entityIdList;

// An entity is characterized by its identifier and name and a set of
properties.
struct t_Entity {
        t_entityId                      entityId;
        string                          entityName;
        SPFEECommonTypes::t_PropertyList   properties;
};
typedef sequence<t_Entity> t_EntityList;

// The SAE is characterized by an identifier, a name and a set of
properties
struct t_Sae {
        t_entityId      entityId;
        string          entityName;
        SPFEECommonTypes::t_PropertyList   properties; // like password
};

// The SAG identifier identifies a SAG uniquely inside the retailer
domain.
typedef short    t_SagId;
typedef sequence<t_SagId> t_SagIdList;
```

```
// A SAG is characterized by its identifier, a textual description of the
// group and the list of entities composing it.
// The identifier is the same as the one for SAG Service profile
corresponding to that SAG.
struct t_Sag {
        t_SagId          sagId;
        string           sagDescription;
        t_entityIdList   entityList;
};
typedef sequence<t_Sag>   t_SagList;

// Subscriber Information:

// Time and Date.
struct t_DateTime {
        string           date;
        string           time;
};

// Textual identification of a person. For example, name, address and
position.
typedef string       t_Person;

// Indicates the date and time an authorization expires on and the person
who granted it.
struct t_AuthLimit {
        t_DateTime       limitDate;
        string           authority;
}; // note: Shouldn't this be per service contract?

// A subscriber is identified by its account number and characterized by
// name, address, monthly charge, payment record, credit information,
// date which its subscription expires on, the list of subscribed services
// and the list of defined SAGs.
struct t_Subscriber {
        t_AccountNumber           accountNumber;
        SPFEECommonTypes::t_UserId subscriberName;
        t_Person                  identificationInfo;
        t_Person                  billingContactPoint;
        string                    RatePlan;
        any                       paymentRecord;
        any                       credit;
};
typedef sequence<t_Subscriber> t_SubscriberList;

// This structure contains information about the minimal required
configuration
// of a service. This is used to specify a configuration for a particular
service session
struct t_RequiredConfiguration {
   t_TermType termType;
   t_NapType nap_type;
   t_PresentationSupport presentation_support;
   SPFEECommonTypes::t_PropertyList others; // to be determined.
};

// Service access rights.
enum t_AccessRight {create,join,be_invited};

// List of possible service access rights.
typedef sequence<t_AccessRight> t_AccessRightList;

// Service Parameter name.
typedef string  t_ParameterName;
```

```
// Service Parameter configurability.
enum t_ParameterConfigurability {
    FIXED_BY_PROVIDER, CONFIGURABLE_BY_SUBSCRIBER, CUSTOMIZABLE_BY_USER
};

// Service Parameter value.
typedef any     t_ParameterValue;

// Service Parameters definition:
struct t_Parameter {
    t_ParameterName             name;
    t_ParameterConfigurability  configurability;
    t_ParameterValue            value;
};
typedef sequence<t_Parameter> t_ParameterList;

struct t_ServiceDescription {
        SPFEEAccessCommonTypes::t_ServiceId              serviceId;
        SPFEEAccessCommonTypes::t_UserServiceName        serviceName;
        t_ParameterList                                  serviceCommonParams;
        t_ParameterList                                  serviceSpecificParams;
};

// t_serviceTemplate: It describes a service instance.
struct t_ServiceTemplate {
        SPFEEAccessCommonTypes::t_ServiceId         serviceInstanceId;
        SPFEEAccessCommonTypes::t_UserServiceName   serviceInstanceName;
        t_ServiceIdList                            requiredServices;
        t_ServiceDescription                        serviceDescription;
};
typedef sequence<t_ServiceTemplate> t_ServiceTemplateList;

// t_ServiceProfile: It describes a service customization.
typedef string t_ServiceProfileId;
typedef sequence<t_ServiceProfileId> t_ServiceProfileIdList;
struct t_ServiceProfile {
        t_ServiceProfileId                          spId;
        t_ServiceDescription                        serviceDescription;
};
typedef sequence<t_ServiceProfile> t_ServiceProfileList;

// Service Profile for a SAG.
typedef t_ServiceProfile t_SagServiceProfile;

// Service Profile by default in a Service Contract.
typedef t_ServiceProfile t_SubscriptionProfile;

// List of SAG Service Profiles
typedef sequence<t_SagServiceProfile> t_SagServiceProfileList;

// Service Contract: Describes the relationship of a subscriber with the
// provider for the provision of a service.
struct t_ServiceContract {
        SPFEEAccessCommonTypes::t_ServiceId  serviceId;
        t_AccountNumber                     accountNumber;
        short                               maxNumOfServiceProfiles;
        t_DateTime                          actualStart;
        t_DateTime                          requestedStart;
        t_Person                            requester;
        t_Person                            technicalContactPoint;
        t_AuthLimit                         authorityLimit;
        t_SubscriptionProfile               subscriptionProfile;
        t_SagServiceProfileList             sagServiceProfileList;
};
```

```
// Notification Type, used in "i_SubscriptionNotify::notify"
enum t_subNotificationType
{NEW_SERVICES,PROFILE_MODIFIED,SERVICES_WITHDRAWN};
// Notification Type, used in "i_ServiceNotify::notify"
enum t_slcmNotificationType
{NEW_SERVICE,TEMPLATE_MODIFIED,SERVICE_WITHDRAWN};
};
#endif // File SPFEESubCommonTypes.idl
```

9.4.2 SPFEERetSubscriberSubscriptionMgmt.idl

```
// File: SPFEERetSubscriberSubscriptionMgmt.idl
// Based on: SPFEEScsSubscriptionService.idl
// Contents: Definition of the online subscription management
// service specific interface.
//

#ifndef SPFEERetSubscriberSubscriptionMgmt_IDL
#define SPFEERetSubscriberSubscriptionMgmt_IDL
#include "SPFEESubCommonTypes.idl"

module SPFEERetSubscriberSubscriptionMgmt {

interface i_SubscriberSubscriptionMgmt {

enum t_errorType { NameTooLong, AddressTooLong, OtherErrors };
exception e_invalidSubscriberInfo{
      t_errorType errorType;
};
exception e_unknownSubscriber{
      SPFEESubCommonTypes::t_AccountNumber       subscriberId;
};
exception e_applicationError{};
exception e_invalidEntityInfo{};
exception e_unknownSAE{
      SPFEESubCommonTypes::t_entityId    entityId;
};
exception e_unknownSAG{
      SPFEESubCommonTypes::t_SagId       sagId;
};
exception e_invalidSAG{
      SPFEESubCommonTypes::t_SagId       sagId;
};
exception e_invalidContractInfo{};
exception e_invalidSubscriptionProfile{};
exception e_invalidSAGServiceProfile{
          SPFEESubCommonTypes::t_ServiceProfileId     spId;
};
exception e_unknownServiceProfile{
          SPFEESubCommonTypes::t_ServiceProfileId     spId;
};
exception e_unknownServiceId{
      SPFEEAccessCommonTypes::t_ServiceId serviceId;
};
exception e_invalidSearchCriteria{};
exception e_notSubscribedService{
          SPFEEAccessCommonTypes::t_ServiceId serviceId;
};
```

```
 // Operations for Subscription and Service Contract handling

 // This operation returns the list of services available for
 subscription and use.
 void listServices (
        out SPFEEAccessCommonTypes::t_ServiceList      serviceList
 ) raises (e_applicationError);

 // This operation creates a subscription for a new customer.
 // The initial list of services the subscriber wants to contract c
 an be specified.
 // It returns:
 // – a unique identifier for the subscriber.
 // – a list of service templates
 void subscribe (
        in SPFEESubCommonTypes::t_Subscriber       subscriberInfo,
        in SPFEESubCommonTypes::t_ServiceIdList       serviceList,
        out SPFEESubCommonTypes::t_AccountNumber     subscriberId,
   out SPFEESubCommonTypes::t_ServiceTemplateList   svcTemplateList
 ) raises (e_invalidSubscriberInfo,
          e_unknownServiceId,
          e_applicationError);

 // This operation creates a (set of) new service contract(s) for
 an existing customer.
 // A list of services the subscriber wants to contract is
 specified.
 // It returns a list of service contract management interfaces,
 //  one for each of the services requested.
 void contractService (
       in SPFEESubCommonTypes::t_ServiceIdList         serviceList,
  out SPFEESubCommonTypes::t_ServiceTemplateList    svcTemplateList
 ) raises (e_unknownSubscriber,
          e_unknownServiceId,
          e_applicationError);

// This operation withdraws a subscription or a list of service
contracts.
 // The list of services the subscriber wants to unsubscribe
 // is an input parameter. If this list is empty, that means
 //  the withdrawal of all the services, and thus the subscription.
 void unsubscribe (
       in SPFEESubCommonTypes::t_ServiceIdList         serviceList
 ) raises (e_unknownSubscriber,
          e_unknownServiceId,
          e_applicationError);

// Operations for Subscriber Information Management.
// ------------------------------------------------------------

// This operation creates a set of entities.
// Sub generates a unique identifier for every entity.
//
 void createSAEs (
        in SPFEESubCommonTypes::t_EntityList     entityList,
        out SPFEESubCommonTypes::t_entityIdList  entityIdList
 ) raises (e_applicationError,
          e_invalidEntityInfo);

 // This operation deletes a set of entities. The entity is
 // removed from all the SAGs it could be assigned to and then
 deleted.
```

```
  void deleteSAEs (
          in SPFEESubCommonTypes::t_EntityList     entityList,
          in SPFEESubCommonTypes::t_entityIdList entityIdList
  ) raises (e_applicationError,
            e_unknownSAE);

  // This operation creates a set of SAGs.
  // It returns a set of unique SAG identifiers.
  void createSAGs (
          in SPFEESubCommonTypes::t_SagList    sagList,
          out SPFEESubCommonTypes::t_SagIdList sagIdList
  ) raises (e_applicationError,
            e_invalidSAG,
            e_unknownSAG);

  // This operation deletes a SAG. The entities belonging to that
   SAG are not deleted.
  void deleteSAGs (
          in SPFEESubCommonTypes::t_SagIdList sagIdList
  ) raises (e_applicationError,
            e_unknownSAG);

  // This operation assigns a list of entities to a SAG.
  void assignSAEs (
          in SPFEESubCommonTypes::t_entityIdList entityList,
          in SPFEESubCommonTypes::t_SagId        sagId
  ) raises (e_unknownSAE,
            e_unknownSAG,
            e_applicationError);

  // This operation removes a list of entities from a SAG.
  void removeSAEs (
          in SPFEESubCommonTypes::t_entityIdList entityList,
          in SPFEESubCommonTypes::t_SagId        sagId
  ) raises (e_unknownSAE,
            e_unknownSAG,
            e_applicationError);

  // This operation returns the list of entities assigned to a SAG.
  // If a SAG is not specified, it returns all the entities for that
   subscriber.
  void listSAEs (
          in SPFEESubCommonTypes::t_SagId         sagId,
          out SPFEESubCommonTypes::t_entityIdList entityList
  ) raises (e_unknownSAG,
            e_applicationError);

  // This operation returns the list of SAGs for that subscriber.
  void listSAGs (
          out SPFEESubCommonTypes::t_SagIdList     sagIdList
  ) raises (e_applicationError);

  // This  operation  returns  the  information  about  a  specific
   subscriber
  void getSubscriberInfo (
          out SPFEESubCommonTypes::t_Subscriber     subscriberInfo
  ) raises (e_applicationError,
            e_unknownSubscriber);

  // This  operation  modifies  the  information  about  a  specific
   subscriber
// Only name and address fields are modifiable. The rest are updated
  // only by Sub as a result of other operations -createSAGs,...-
```

```
    void setSubscriberInfo (
            in SPFEESubCommonTypes::t_Subscriber     subscriberInfo
    ) raises (e_unknownSubscriber,
              e_invalidSubscriberInfo,
              e_applicationError);

    // This operation returns the list of services subscribed by
    //  a specific subscriber.
    //
    void  listSubscribedServices (
           out SPFEEAccessCommonTypes::t_ServiceList      service_list
    ) raises (e_applicationError,
              e_unknownSubscriber);

   // Operations for Service Contract Management.
   // -------------------------------------------------

    // This operation returns the template for the service.
    void getServiceTemplate (
            in SPFEEAccessCommonTypes::t_ServiceId           serviceId,
            out SPFEESubCommonTypes::t_ServiceTemplate  template
    ) raises (e_applicationError);

    // This operation creates a service contract.
    // This contract can include a set of service profiles.
    void defineServiceContract (
        in SPFEESubCommonTypes::t_ServiceContract     serviceContract,
            out SPFEESubCommonTypes::t_ServiceProfileIdList  spIdList
    )
    raises (e_applicationError,
            e_invalidContractInfo,
            e_invalidSubscriptionProfile,
            e_invalidSAGServiceProfile);

    // This operation creates a set of service profiles.
    void defineServiceProfiles (
in SPFEESubCommonTypes::t_SubscriptionProfile   subscriptionProfile,
in SPFEESubCommonTypes::t_SagServiceProfileList  sagServiceProfiles,
           out SPFEESubCommonTypes::t_ServiceProfileIdList  spIdList
    )
    ras      e_invalidSubscriptionProfile,
            e_invalidSAGServiceProfile);

    //  his operation  deletes  a  set  of  service  profiles  and  their
     associated SAGs.
    void deleteServiceProfiles (
            in SPFEESubCommonTypes::t_ServiceProfileIdList  spIdList
    )
    raises (e_applicationError,
            e_unknownServiceProfile);

    // This operation returns the list of service profiles identifiers
    void listServiceProfiles (
            in SPFEEAccessCommonTypes::t_ServiceId          serviceId,
            out SPFEESubCommonTypes::t_ServiceProfileIdList  spIdList
    )
    raises (e_applicationError);

    // This operation returns the service contract information.
    // If a (list of) SAG(s) is specified it returns the set of
    // SAG service profile for that(those) SAG(s).
    void getServiceContractInfo (
         in  SPFEEAccessCommonTypes::t_ServiceId          serviceId,
         in  SPFEESubCommonTypes::t_ServiceProfileIdList     spIdList,
```

```
                out SPFEESubCommonTypes::t_ServiceContract    serviceContract
        )
        raises (e_applicationError,
                e_unknownServiceProfile);

        // This operation assigns a service profile to a list of SAGs and
        SAEs.
        void assignServiceProfile (
                in SPFEESubCommonTypes::t_ServiceProfileId  spId,
                in SPFEESubCommonTypes::t_SagIdList  sagIdList,
                in SPFEESubCommonTypes::t_entityIdList  saeIdList
        )
        raises (e_applicationError,
                e_unknownSAG,
                e_unknownSAE,
                e_unknownServiceProfile);

        // This operation removes a service profile assignment to a list
       of SAGs and SAEs.
        void removeServiceProfile (
                in SPFEESubCommonTypes::t_ServiceProfileId  spId,
                in SPFEESubCommonTypes::t_SagIdList  sagIdList,
                in SPFEESubCommonTypes::t_entityIdList  saeIdList
        )
        raises (e_applicationError,
                e_unknownSAG,
                e_unknownSAE,
                e_unknownServiceProfile);

        // This operation activates a set of service profiles
        void activateServiceProfiles (
                in SPFEESubCommonTypes::t_ServiceProfileIdList  spIdList
        )
        raises (e_applicationError,
                e_unknownServiceProfile);

        // This operation deactivates a set of service profiles
        void deactivateServiceProfiles (
                in SPFEESubCommonTypes::t_ServiceProfileIdList  spIdList
        )
        raises (e_applicationError,
                e_unknownServiceProfile);

}; // i_SubscriberSubscriptionMgmt
};
#endif // SPFEERetSubscriberSubscriptionMgmt_IDL
```

9.4.3 SPFEERetRetailerSubscriptionMgmt.idl

```
#ifndef SPFEERetRetailerSubscriptionMgmt_IDL
#define SPFEERetRetailerSubscriptionMgmt_IDL
#include "SPFEERetSubscriberSubscriptionMgmt.idl"

module SPFEERetRetailerSubscriptionMgmt {

interface i_RetailerSubscriptionMgmt:
SPFEERetSubscriberSubscriptionMgmt::i_SubscriberSubscriptionMgmt
{
// This interface inherits from  i_SubscriberSubscriptionMgmt, and adds a
few
//  operations  for  the  retailer  operator's  access  to  subscription
functionality
// through the Ret-RP.
```

```
        // TBD.
        void listSubscribers (
        );

        // TBD.
        void listServiceContracts (
        );

        // TBD.
        void listUsers (
        );

}; // i_RetailerSubscriptionMgmt

};

#endif // SPFEERetRetailerSubscriptionMgmt_IDL
```

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 de données et communication entre systèmes ouverts
Série Y	Infrastructure mondiale de l'information et protocole Internet
Série Z	Langages et aspects informatiques généraux des systèmes de télécommunication

Genève, 2001