UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

Serie Q

Suplemento 28
(12/1999)

SERIE Q: CONMUTACIÓN Y SEÑALIZACIÓN

Informe técnico: Marco de señalización y protocolo para un entorno en evolución – Especificación para el acceso de servicio

Recomendaciones UIT-T de la serie Q – Suplemento 28

(Anteriormente Recomendaciones del CCITT)

RECOMENDACIONES UIT-T DE LA SERIE Q

CONMUTACIÓN Y SEÑALIZACIÓN

SEÑALIZACIÓN EN EL SERVICIO MANUAL INTERNACIONAL	Q.1–Q.3
EXPLOTACIÓN INTERNACIONAL SEMIAUTOMÁTICA Y AUTOMÁTICA	Q.4–Q.59
FUNCIONES Y FLUJOS DE INFORMACIÓN PARA SERVICIOS DE LA RDSI	Q.60–Q.99
CLÁUSULAS APLICABLES A TODOS LOS SISTEMAS NORMALIZADOS DEL UIT-T	Q.100–Q.119
ESPECIFICACIONES DE LOS SISTEMAS DE SEÑALIZACIÓN N.° 4 Y N.° 5	Q.120–Q.249
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 6	Q.250–Q.309
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN R1	Q.310–Q.399
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN R2	Q.400–Q.499
CENTRALES DIGITALES	Q.500–Q.599
INTERFUNCIONAMIENTO DE LOS SISTEMAS DE SEÑALIZACIÓN	Q.600–Q.699
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 7	Q.700–Q.849
SISTEMA DE SEÑALIZACIÓN DIGITAL DE ABONADO N.° 1	Q.850–Q.999
RED MÓVIL TERRESTRE PÚBLICA	Q.1000–Q.1099
INTERFUNCIONAMIENTO CON SISTEMAS MÓVILES POR SATÉLITE	Q.1100–Q.1199
RED INTELIGENTE	Q.1200–Q.1699
REQUISITOS Y PROTOCOLOS DE SEÑALIZACIÓN PARA IMT-2000	Q.1700–Q.1799
RED DIGITAL DE SERVICIOS INTEGRADOS DE BANDA ANCHA (RDSI-BA)	Q.2000–Q.2999

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

Suplemento 28 a las Recomendaciones UIT-T de la serie Q

Informe técnico: Marco de señalización y protocolo para un entorno en evolución – Especificación para el acceso de servicio

Resumen

Este Suplemento especifica el modelo de información y computación (nivel de sesión) del punto de referencia consumidor-detallista. Proporciona especificación de información y computación para el acceso de servicio definido en el marco de señalización y protocolo para un entorno en evolución (SPFEE). Las especificaciones de información y computación se describen en lenguajes de descripción informales y formales, tales como el lenguaje de definición de interfaz (IDL).

Orígenes

El Suplemento 28 a las Recomendaciones UIT-T de la serie Q, preparado por la Comisión de Estudio 11 (1997-2000) del UIT-T, fue aprobado por el procedimiento de la Resolución 5 de la CMNT el 3 de diciembre de 1999.

Palabras clave

Acceso, IDL, modelo computacional, modelo de información, punto de referencia, sesión.

PREFACIO

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

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

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

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

NOTA

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

PROPIEDAD INTELECTUAL

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

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

ÍNDICE

Suplemento 28 a las Recomendaciones UIT-T de la serie Q

Informe técnico: Marco de señalización y protocolo para un entorno en evolución – Especificación para el acceso de servicio

(Ginebra, 1999)

1 Alcance

Este Suplemento proporciona:

– especificación de información; y

– especificación de computación

para el acceso de servicio definido en el marco de señalización y protocolo para un entorno en evolución (SPFEE, *signalling and protocol framework for an evolving environment*), que incluyen las especificaciones del punto de referencia consumidor-detallista. Las especificaciones de información y computación se describen en lenguajes de descripción informales y formales, tales como el lenguaje de definición de interfaz (IDL, *interface definition language*).

2 Referencias

Los siguientes Informes técnicos y otras referencias contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones del presente Suplemento. Al efectuar esta publicación, estaban en vigor las ediciones indicadas. Todos los Suplementos u otras referencias son objeto de revisiones, por lo que se preconiza que los participantes en acuerdos basados en el presente Suplemento investiguen la posibilidad de aplicar las ediciones más recientes de los Suplementos y otras referencias citadas a continuación. Se publica periódicamente una lista de las Recomendaciones y Suplementos UIT-T actualmente vigentes.

[1] Recomendación UIT-T X.901 (1997) | ISO/CEI 10746-1:1998, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Visión de conjunto.*

[2] Recomendación UIT-T X.902 (1995) | ISO/CEI 10746-2:1996, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Fundamentos.*

[3] Recomendación UIT-T X.903 (1995) | ISO/CEI 10746-3:1996, *Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Arquitectura.*

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

[5] Recomendación UIT-T Z.130 (1999), *Lenguaje de definición de objetos de la UIT.*

[6] Recomendaciones UIT-T de la Serie Q – Suplemento 27 (1999), *Informe técnico: Visión general del marco de señalización y protocolo para un entorno en evolución.*

3 Definiciones

En este Suplemento se definen los términos siguientes además de los definidos en [6].

3.1 sesión de acceso de dominio (D_AS): Objeto abstracto que representa la información genérica requerida para establecer y soportar el acceso entre dos dominios.

3.2 sesión de acceso de dominio de usuario (UD_AS): Objeto gestionado por el usuario que representa el conjunto de capacidades y configuración que el usuario emplea para ponerse en contacto con un proveedor.

3.3 sesión de acceso de dominio de proveedor (PD_AS): Objeto gestionado por el proveedor y creado cuando el usuario se convierte en una entidad reconocida e identificable con capacidades específicas y datos dentro del dominio de proveedor.

3.4 sesión de servicio de proveedor (PSS): Visión central de la sesión de servicio, que incluye a todos los miembros y cualquier información adicional y lógica de proveedor necesarias para ejecutar peticiones de servicio y mantener la sesión.

3.5 sesión de servicio de utilización (USS): Visión personalizada de un servicio que tiene un miembro de sesión (por ejemplo, la visión de un usuario de extremo).

3.6 sesión de servicio de utilización de dominio (D_USS): Objeto abstracto que representa información genérica requerida para establecer y mantener un servicio entre dos dominios para un miembro de sesión asociado.

3.7 sesión de servicio de utilización de dominio de usuario (UD_USS): Funcionalidad e información presentes en un dominio de usuario de extremo, por ejemplo, un dominio de consumidor, para soportar la sesión de servicio de utilización y permitir que el usuario de extremo interactúe con el servicio.

3.8 sesión de servicio de utilización de dominio de proveedor (PD_USS): Funcionalidad e información presentes en un dominio de proveedor (por ejemplo, detallista, proveedor de terceros) para soportar la sesión de servicio de utilización en el cometido de proveedor de utilización.

3.9 vinculación de sesiones de servicio de utilización de dominio (D_USS_Binding): Información dinámica asociada con la vinculación de dos D_USS.

4 Abreviaturas

En este Suplemento se utilizan las siguientes siglas.

3Pty Punto de referencia entre dominios de terceros (*third-party inter-domain reference point*)

anonUA Agente de usuario anónimo (*anonymous user agent*)

as-UAP Aplicación de usuario (relacionada con sesión de acceso) [*user application (access session related*)]

AS Sesión de acceso (*access session*)

Bkr Punto de referencia entre dominios de corredor (*broker inter-domain reference point*)

CO Objeto computacional (*computational object*)

ConS Punto de referencia entre dominios de servicio de conectividad (*connectivity service inter-domain reference point*)

CS Sesión de comunicación (*communication session*)

CSLN Punto de referencia entre dominios de capa de cliente-servidor (*client-server layer inter-domain reference point*)

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

FCAPS Averías, configuración contabilidad, calidad de funcionamiento, seguridad (*fault, configuration, accounting, performance, security*)

IA Agente inicial (*initial agent*)

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

LNFed Punto de referencia entre dominios de federación de redes de capa (*layer network federation inter-domain reference point*)

NamedUA Agente de usuario denominado (*named user agent*)

ODL Lenguaje de definición de objetos (*object definition language*)

ODP Procesamiento distribuido abierto (*open distributed processing*)

OMT Técnica de modelado de objetos (*object modelling technique*)

PA Agente de proveedor (*provider agent*)

Ret Punto de referencia entre dominios de detallista (*retailer inter-domain reference point*)

RP Punto de referencia (*reference point*)

RtR Punto de referencia entre dominios de detallista a detallista (*retailer to retailer inter-domain reference point*)

SC Componente de servicio (*service component*)

SF Fábrica de servicios (*service factory*)

SPFEE Marco de señalización y protocolo para un entorno en evolución (*signalling and protocol framework for an evolving environment*)

SS Sesión de servicio

SSM Gestor de sesión de servicio (*service session manager*)

ss-UAP Aplicación de usuario (relacionada con sesión de servicio) [*user application (service session related*)]

Tcon Punto de referencia entre dominios de conexión de terminal (*terminal connection inter-domain reference point*)

UA Agente de usuario (*user agent*)

UAP Aplicación de usuario (*user application*)

USM Gestor de sesión de servicio de usuario (*user service session manager*)

5 Modelo de información del nivel de sesión y recurso

El modelo de información se describe en la notación de modelo de objetos de OMT. Los símbolos típicos se ilustran en la figura 5-1. Las clases abstractas (para los cuales no hay objetos) se muestran en el estilo "corchetes ondulados"[1], cuya única finalidad es simplificar el modelado orientado a objetos.

[1] Es decir, "{}" se insertan enfrente de la clase de nombre en los diagramas.

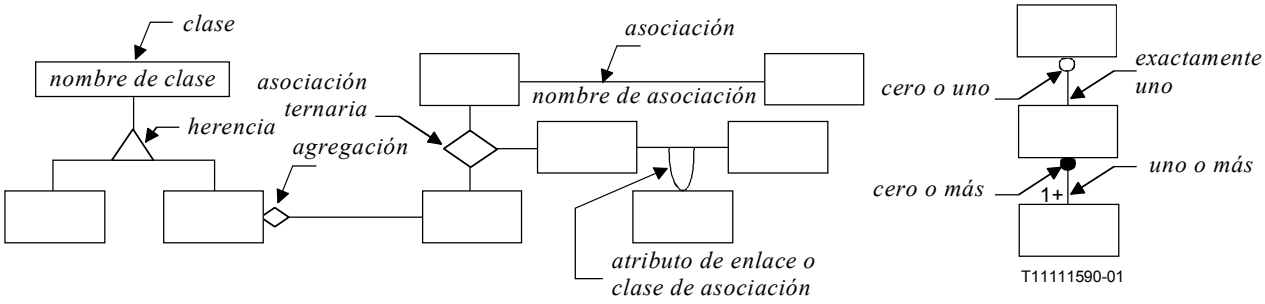


Figura 5-1 – Símbolos típicos en un diagrama OMT

5.1 Sesiones, servicios y dominios

Esta subcláusula presenta el modelo de información en lo que respecta a los servicios relacionados con sesión[2]. Como un servicio relacionado con sesión puede extenderse a múltiples dominios, que son tratados como dominios administrativos comerciales, es útil modelar el servicio como una agregación de uno o más servicios de dominio, donde cada servicio de dominio representa una parte de un servicio confinado a un solo dominio. Una sesión es un ejemplar de un servicio. Como una sesión puede representar un caso de un servicio, una sesión de dominio representa un ejemplar de un servicio de dominio. Las sesiones de dominio pueden interactuar para establecer servicios que abarcan múltiples dominios.

La figura 5-2 muestra la relación entre dominio, servicio y objetos de sesión. Un servicio relacionado con sesión es una agregación de otros servicios relacionados con sesión y servicios de dominio. Cada servicio de dominio es creado por una sesión de dominio, que es establecida en su dominio asociado y es gestionada por éste. Una sesión de dominio puede estar vinculada a otra sesión de dominio por una vinculación de sesiones de dominio, que representa la información dinámica utilizada para enlazar las dos sesiones.

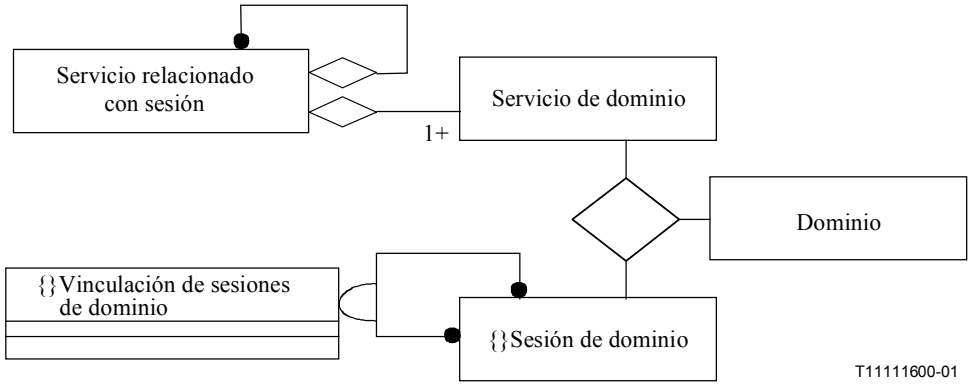


Figura 5-2 – Modelo de información de sesión

---

[2] Un servicio se clasifica en dos tipos: uno es "servicios relacionados con sesión" y el otro es "servicios no relacionados con sesión". Los "servicios no relacionados con sesión" están fuera del ámbito del presente Suplemento. Los servicios relacionados con sesión se definen como sigue: el servicio relacionado con sesión representa un servicio de telecomunicación en línea ofrecido a los usuarios **con** equipos y recursos de telecomunicación. Por ejemplo, todos los servicios de telecomunicación, desde el servicio telefónico ordinario hasta los servicios multimedios y/o los servicios relacionados con Internet, se clasifican como servicios relacionados con sesión.

5.2 Clasificación de sesiones

Esta subcláusula considera la clasificación de sesiones. Son posibles varios esquemas de clasificación, pero todos se basan en el objeto raíz de sesión. Este objeto genérico define las propiedades comunes de una sesión. Todos los objetos derivados, independientemente de su clasificación, heredan estos atributos y operaciones comunes, tales como: identificador de sesión, tipo de sesión, estado, terminación, suspensión y reanudación.

Una clasificación se basa en las separaciones de acceso de servicios (funcionales), servicio y comunicación. Las sesiones han sido identificadas para soportar cada una de estas separaciones. Aunque especializada para soportar los requisitos particulares de una zona funcional, cada sesión retiene las propiedades comunes del objeto raíz de sesión.

Es útil también utilizar la sesión de dominio y la vinculación de sesiones de dominio para facilitar la clasificación de sesiones. También en este caso, el objeto raíz de sesión es la raíz de la jerarquía[3], y la sesión de dominio y la vinculación de sesiones de dominio heredan sus propiedades. Este tipo de clasificación se puede combinar con el esquema anterior para clasificar todos los objetos de sesión para cada separación. A continuación se utilizará este esquema combinado y los siguientes aspectos para clasificar objetos:

– Aspectos de usuario: Representan las entidades, semántica, constricciones y reglas que rigen la disponibilidad y uso de capacidades de servicio con respecto a un usuario específico, y los recursos y mecanismos necesarios para soportar realmente las capacidades de servicio de acuerdo con un punto de vista de usuario específico.

– Aspectos de proveedor: Representan las entidades, semántica, constricciones y reglas que rigen la provisión y uso de capacidades de servicio a los usuarios, así como los recursos y mecanismo necesarios para soportar realmente estas capacidades.

5.3 Clasificación de sesiones de acceso

Las sesiones de acceso se pueden clasificar como una jerarquía de especialización según se muestra en la figura 5-3. La clasificación corresponde con la separación en aspectos de usuario y de proveedor.

Al relacionar esto con la figura 5-2, se observa que la *sesión de acceso de dominio* corresponde (por subclasificación) a la sesión de dominio, y la *sesión de acceso* corresponde (por subclasificación) a la vinculación de sesiones de dominio.

---

[3] El objeto raíz de sesión es un objeto abstracto, lo que significa que este objeto no es ejemplificable.

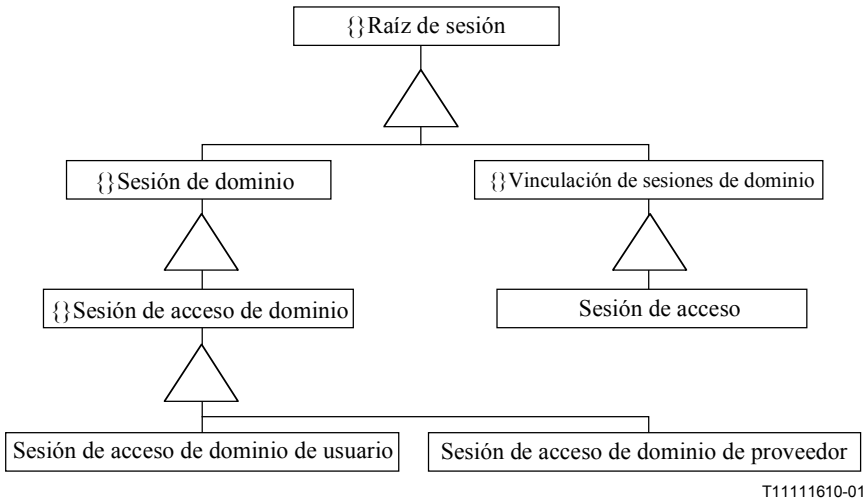


Figura 5-3 – Clasificación de la sesión de acceso

5.4 Sesión de acceso

Los objetos de información relacionados con acceso y sus relaciones se muestran en la figura 5-4.

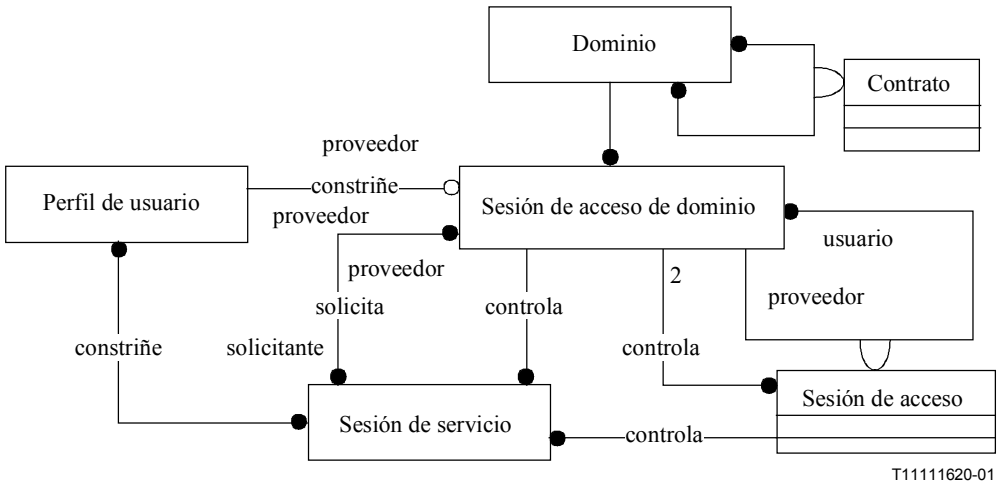


Figura 5-4 – Relaciones entre objetos de información relacionados con acceso

Cada sesión de acceso de dominio está asociada con un cometido de acceso particular. Se han identificado dos cometidos: usuario y proveedor. El cometido de usuario acepta invitaciones y hace peticiones en la sesión de acceso de dominio asociada. El cometido de proveedor acepta peticiones y envía invitaciones a la sesión de acceso de dominio asociada.

La sesión de acceso se clasifica por los cometidos soportados por cada sesión de acceso de dominio, y que dependen de la relación entre dominios. Se puede necesitar los siguientes tipos de sesión de acceso:

– **Sesión de acceso de tipo asimétrico**: Un dominio desempeña el cometido de usuario y el otro el cometido de proveedor.

– **Sesión de acceso de tipo simétrico**: Hay simetría en la sesión de acceso, ambos dominios soportan ambos cometidos.

Los objetos de información y sus relaciones para la sesión de acceso de tipo asimétrico se muestran en la figura 5-5, que presenta una versión más detallada de la figura 5-4 para la sesión de acceso de tipo asimétrico. Una sesión asimétrica es soportada por la sesión de acceso de dominio de usuario y por la sesión de acceso de dominio de proveedor.

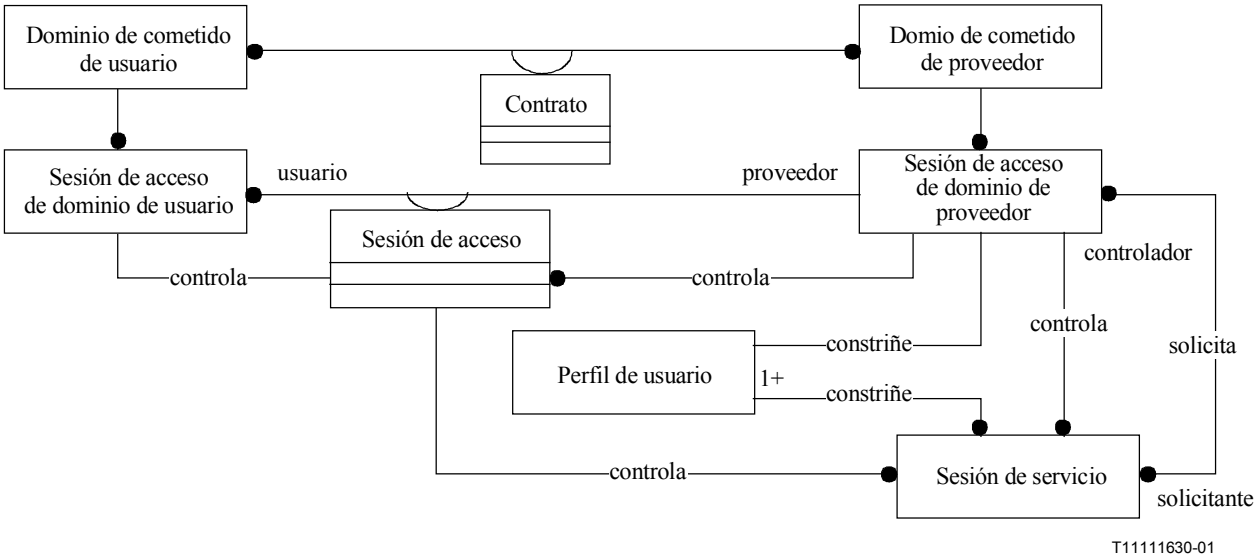


Figura 5-5 – Objetos de información y su relación para sesiones de acceso de tipo asimétrico

5.4.1 Sesión de acceso de dominio (D_AS, *domain access session*)

Éste es un objeto abstracto que representa la información genérica requerida para establecer y soportar acceso entre dos dominios. Cada sesión de acceso de dominio está asociada con un determinado dominio. Sin embargo, un dominio puede tener muchas sesiones de acceso de dominio. Una sesión de acceso de dominio suele estar asociada con una (o posiblemente más) relaciones contractuales con otro dominio. Puede haber múltiples sesiones de acceso de dominio dentro del dominio para cada relación contractual.

La D_AS está especializada en objetos de información de tipo sesión de acceso de dominio de usuario y de sesión de acceso de dominio de proveedor.

5.4.1.1 Sesión de acceso de dominio de usuario (UD_AS, *user domain access session*)

Este objeto es gestionado por el usuario y representa el conjunto de capacidades y la configuración que el usuario emplea para ponerse en contacto con un proveedor. La UD_AS mantiene políticas definidas por el usuario, que determinan los términos de la interacción con un proveedor, tales como política de seguridad y contabilidad; constituyen la base de la negociación con un proveedor en el establecimiento de una sesión de acceso mutuamente aceptable. La UD_AS puede comprender de uno a muchos a terminales o nodos DPE. Cualquiera que sea la configuración UD_AS, el usuario y el proveedor tendrán una perspectiva sobre la fiabilidad de la UD_AS. Este nivel de confianza (por ejemplo, confidencialidad, protección de contraseña, implementaciones de cifrado) se reflejará en las restricciones de gestión impuestas a la sesión de acceso (por ejemplo, rechazo de servicios de alto valor). Una UD_AS soportada en un terminal resistente a las manipulaciones suministrado por el proveedor es más fiable que una pequeña LAN de PC multiusuarios.

5.4.1.2 Sesión de acceso de dominio de proveedor (PD_AS, *provider domain access session*)

Este objeto es gestionado por el proveedor y cabe considerar que es creado cuando el usuario se convierte en una entidad reconocida e identificable con capacidades específicas y datos dentro del dominio del proveedor. La sesión termina cuando el usuario cesa de tener cualquier relación de usuario con el proveedor (de modo que el proveedor deja de mantener información permanente sobre el usuario en el perfil de usuario). Este objeto conoce información persistente sobre el usuario. Parte de esta información especifica políticas que determinan los términos de interacción con el usuario, tales como política de seguridad y contabilidad, que constituyen la base de la negociación con el usuario en el establecimiento de una sesión de acceso mutuamente aceptable. El usuario y el detallista tendrán una perspectiva de la fiabilidad de la PD_AS. Este nivel de confianza (por ejemplo, confiabilidad, protección de contraseña, implementaciones de cifrado) se reflejará en las restricciones de gestión impuestas a la sesión de acceso.

5.4.2 Sesión de acceso (AS, *access session*)

Este objeto es gestionado por el usuario o el proveedor por medio de la D_AS, y representa el conjunto de información dinámica para una vinculación entre las D_AS, tales como política de seguridad, contabilidad y descripción de sesión para esta vinculación. Este objeto termina cuando el usuario o el proveedor termina la relación (vinculación dinámica entre las D_AS) con el proveedor o con el usuario.

5.4.3 Perfil de usuario

El perfil de usuario contiene toda la información usada directamente por la D_AS para decisiones de autorización, restricciones y personalización de las D_AS, sesiones de acceso (dentro de vinculaciones de sesiones de acceso) y sesiones de servicio.

5.5 Clasificación de sesiones de servicio

La figura 5-6 muestra la jerarquía de clasificación de sesiones de servicio con la separación en aspectos de usuario y de proveedor.

Al relacionar esto con la figura 5-2, se observa que la sesión de servicio de proveedor (PSS, *provider service session*) y la sesión de servicio de utilización de dominio (D_USS, *domain usage service session*) corresponden (por subclasificación) a la sesión de dominio, y las vinculaciones de sesiones de servicio de utilización de dominio corresponden (por subclasificación) a la vinculación de sesiones de dominio.

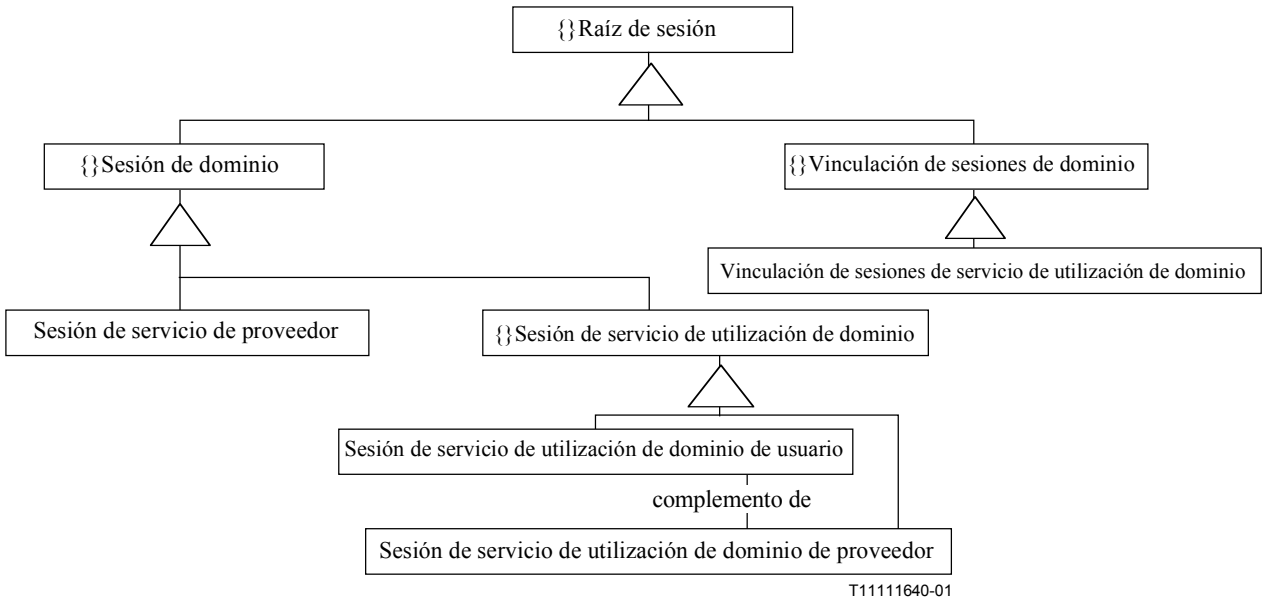


Figura 5-6 – Clasificación de la sesión de servicio

5.6 Sesión de servicio

Los objetos relacionados con la sesión de servicio y sus relaciones se muestran en la figura 5-7.

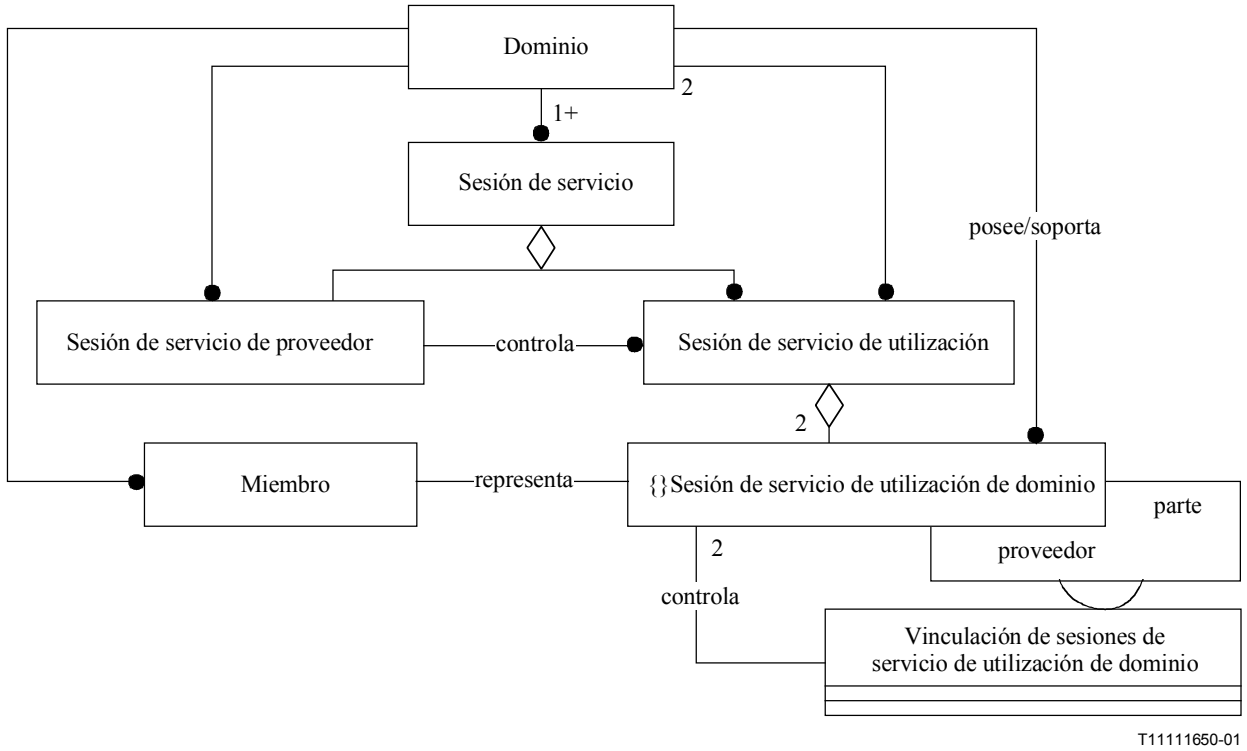


Figura 5-7 – Objetos relacionados con la sesión de servicio y sus relaciones

La sesión de servicio consta de sesiones de servicio de utilización y de proveedor. Cada sesión de servicio de utilización puede abarcar más de dos dominios y se compone de dos sesiones de servicio de utilización de dominio complementarias (D_USS), donde una D_USS complementaria es la que

puede interactuar con otra. Cada miembro de una sesión, es decir, un usuario de extremo, recurso de servicio o sesión asociada, está asociado con una sesión de servicio de utilización.

El tipo de D_USS soportada depende del cometido percibido de ese miembro en la sesión de servicio. La figura 5-6 muestra las posibles D_USS. Se soportan los siguientes cometidos de utilización genéricos:

– Parte de utilización: Se asemeja a un "usuario de extremo" de un servicio, un cometido activo que puede hacer peticiones y recibe notificaciones de cambios de sesión.

– Proveedor de utilización: Proporciona un servicio o actúa como un recurso de servicio para otra entidad. Es un cometido pasivo en el sentido de que no puede iniciar acciones, sino que responde a peticiones o notificaciones de cambios.

Son posibles más cometidos, entre los que cabe citar cometidos específicos de servicio, que están fuera del ámbito de este Suplemento y los cometidos de control y gestión para soportar relaciones en servicios compuestos.

Cuando comienza una sesión de servicio, o cuando un nuevo miembro se incorpora a la sesión de servicio, ésta adquiere la información de perfil de usuario pertinente de la sesión de acceso o de la sesión de acceso de dominio (por ejemplo, descripción de servicios) para el miembro. Esto restringe la sesión de servicio de utilización y potencialmente la sesión de servicio de proveedor. En el caso de sesiones de servicio de múltiples miembros, el perfil de usuario de un individuo o la configuración de utilización vigente pueden afectar a toda la sesión de servicio, dependiendo de la naturaleza del servicio y de sus políticas de gestión.

Una sesión de servicio puede ser ejemplificada por una sesión de acceso, sesión de acceso de dominio u otra sesión de servicio. El iniciador de una sesión de servicio la asocia con su(s) miembro(s). Los miembros pueden tener diferentes responsabilidades dentro de la sesión (por ejemplo, gestión o puramente interacción con el contenido de servicio y el control de sesión general). Si una sesión de servicio es responsabilidad de una sesión de acceso, la sesión de servicio puede permanecer activa mientras esas sesión de acceso esté activa. Cuando termina una sesión de acceso, las sesiones de servicio de utilización conexas deben ser terminadas, suspendidas, o transferidas a otra sesión de acceso o sesión de acceso de dominio.

5.6.1 Sesión de servicio de proveedor (PSS)

Contiene una visión central de la sesión de servicio, incluidos todos los miembros y la información adicional y lógica de proveedor necesarias para ejecutar peticiones de servicio y mantener la sesión. El soporte de esta sesión es responsabilidad del proveedor. Una sesión de servicio de proveedor representa las capacidades de servicio comunes a múltiples miembros. Generalmente, la sesión de servicio de proveedor mantiene objetos de información relacionados con la gestión del servicio (por ejemplo, contabilidad) o información del servicio relacionada con el sistema.

5.6.2 Sesión de servicio de utilización (USS, *usage service session*)

Ésta es la visión de un servicio personalizada que tiene el miembro (por ejemplo, la visión de un usuario de extremo). Oculta la complejidad del servicio al miembro y asegura que las preferencias del miembro y el entorno son soportados por el servicio. Oculta la heterogeneidad de cada configuración de utilización a la sesión de servicio de proveedor. Se descompondrá además en partes de utilización de dominio.

5.6.3 Sesión de servicio de utilización de dominio (D_USS)

Éste es un objeto abstracto que representa información genérica requerida para establecer y soportar un servicio entre dos dominios para un miembro de sesión asociada. Cada sesión de servicio de utilización de dominio está asociada con un dominio particular y una sesión de servicio particular. Como es un objeto abstracto, no es ejemplificable. La D_USS está especializada en cuatro tipos de nuevos objetos de información como se describe a continuación.

5.6.3.1 Sesión de servicio de utilización de dominio de usuario (UD_USS, *user domain usage service session*)

Ésta es la funcionalidad e información presentes en un dominio de usuario de extremo, por ejemplo, un dominio de consumidor, para soportar la sesión de servicio de utilización y permitir que el usuario de extremo interactúe con el servicio. En todos los casos, la UD_USS está asociada con el cometido de parte de utilización. Es responsabilidad del dominio de usuario de extremo realizar y gestionar esta sesión. Sin embargo, algunas responsabilidades de realización y gestión de recursos en la UD_USS pueden ser asignados al dominio de proveedor mediante acuerdo.

5.6.3.2 Sesión de servicio de utilización de dominio de proveedor (PD_USS, *provider domain usage service session*)

Ésta es la funcionalidad e información presentes en un dominio de proveedor (por ejemplo, detallista, proveedor de terceros) para soportar la sesión de servicio de utilización en el cometido de proveedor de utilización. Oculta la complejidad y especificidad del otro dominio a la PSS y aísla la actividad específica de la parte de la actividad general de la PSS, que es común a todos los participantes en la sesión de servicio. El dominio con el cometido de proveedor de utilización es responsable de realizar y gestionar la sesión de servicio de utilización de dominio de proveedor.

5.6.4 Vinculación de sesiones de servicio de utilización de dominio (D_USS_Binding, *domain usage service session binding*)

Representa la información dinámica asociada con la vinculación de dos D_USS. Las D_USS controlan esta información. La información contenida es determinada por el tipo de D_USS que participa en la vinculación, y los modelos de sesión.

5.7 Recursos [Sesión de comunicación (CS)]

El recurso [sesión de comunicación (CS, *communication session*)] representa una visión de servicio general de conexiones de trenes y una visión independiente de la tecnología de red de los recursos de comunicación requeridos para establecer conexiones de extremo a extremo. Un recursos (sesión de comunicación) puede tratar múltiples conexiones que pueden ser multipunto y multimedios.

Un recurso (sesión de comunicación) puede acordar la calidad de servicio, el establecimiento, modificación y supresión de múltiples conexiones.

La adopción del concepto "sesión" para controlar la capacidad de comunicación tiene la ventaja de que los servicios pueden crear dinámicamente, retener, reanudar y mantener una configuración adecuada de recursos de comunicación que satisfaga sus necesidades.

Un recurso (sesión de comunicación) es controlada por una sesión de servicio de la PSS o PD_USS. Sólo una sesión de servicio puede estar asociada con un recurso (sesión de comunicación) en un momento dado.

6 Modelo computacional del nivel de sesión y recurso

6.1 Componentes relacionados con la sesión de acceso

Los componentes relacionados con la sesión de acceso soportan las sesiones relacionadas con acceso[4]. Soportan la funcionalidad y operaciones de sesión comunes definidas para los conceptos de sesión. En 6.4 "Relación con el modelo de información" se indica una correspondencia entre los conceptos de sesión y los componentes de servicio.

---

[4] Sesión de acceso (AS), sesión de acceso de dominio de proveedor (PD_AS) y sesión de acceso de dominio de usuario (UD_AS).

Las sesiones de acceso pueden ser simétricas o asimétricas. El tipo de sección de acceso es determinado por el punto de referencia entre los dominios. A continuación se consideran los componentes necesarios para soportar una sesión de acceso asimétrica. El cometido de usuario es desempeñado por el agente de proveedor mientras que el cometido de proveedor es desempeñado por el agente inicial y el agente de usuario. Se considera también una aplicación de usuario, que satisface las necesidades de los usuarios de extremo.

6.1.1 Aplicación de usuario

La aplicación de usuario (UAP, *user application*) se define para modelar una variedad de aplicaciones y programas en el dominio. Una UAP representa una o más de estas aplicaciones y programas. La UAP puede ser utilizada por usuarios humanos y/o otras aplicaciones en el dominio de usuario. La UAP puede ser un componente relacionado con sesión de acceso o un componente relacionado con sesión de servicio, o ambos. A continuación se define la UAP relacionada con la sesión de acceso. La UAP relacionada con la sesión de servicio se define en 6.2.

Como un componente relacionado con la sesión de acceso, la UAP permite a un usuario humano o a otra aplicación, utilizar las capacidades de un PA a través de una interfaz apropiada (de usuario). Una UAP relacionada con la sesión de acceso sustenta parte de la sesión de acceso de dominio. La UAP proporciona las capacidades para:

- pedir información de autenticación del usuario, requerida por el PA para establecer una sesión de acceso con un UA;
- que el usuario pida la creación de nuevas sesiones de servicio;
- que el usuario pida incorporarse a una sesión de servicio existente;
- avisar al usuario cuando llegan invitaciones al PA.

Una UAP relacionada con la sesión de acceso puede soportar también las siguientes capacidades facultativas, cuando son soportadas también por el PA:

- permitir que el usuario busque un proveedor, y se registre como un usuario de los servicios del proveedor;
- permitir que el usuario busque servicios e identifique proveedores que suministren estos servicios.

Una UAP puede tener ninguna o más interfaces de trenes. Estas interfaces pueden estar limitadas a las de los sistemas de usuario y/o a las de los dominios de proveedores.

Un dominio de usuario contiene una o más UAP relacionadas con sesión de acceso. Cualquier UAP relacionada con sesión de acceso puede pedir a un PA que establezca una sesión de acceso. Una o más UAP interactúan con un PA para utilizar sus capacidades relacionadas con sesión de acceso dentro de una sesión de acceso.

Una UAP puede soportar capacidades relacionadas solamente con sesión de acceso o capacidades relacionadas solamente con sesión de servicio, o puede soportar ambas. Las UAP relacionadas con sesión de acceso pueden estar especializadas por un dominio para interactuar con un PA especializado.

6.1.2 Agente de proveedor

El agente de proveedor (PA, *provider agent*) es un componente independiente del servicio, definido como el punto extremo del usuario de una sesión de acceso. El PA es soportado en un dominio, que desempeña el cometido de usuario de acceso. El PA soporta a un usuario que accede a su UA y utiliza servicios, a través de una sesión de acceso. El PA soporta la sesión de acceso de dominio de usuario, junto con las UAP relacionadas con sesión de acceso, y otras infraestructuras del dominio de usuario.

Las capacidades soportadas por un PA:

– establecen una relación fiable entre el usuario y el proveedor (una sesión de acceso), interactuando con un agente inicial[5] y obteniendo una referencia a un UA[6];

– dentro de una sesión:

  • transportan peticiones (de un usuario a un UA) para crear nuevas sesiones de servicio;

  • transportan peticiones de incorporación a una sesión de servicio existente;

  • reciben invitaciones de incorporación a sesiones de servicio existentes (de un UA) y avisar al usuario[7];

  • utilizan anónimamente los servicios del proveedor;

  • despliegan nuevos componentes en el dominio del usuario;

  • soportan el acceso a información de configuración de terminal de un dominio de proveedor;

  • registran la recepción de invitaciones enviadas cuando no existe sesión de acceso.

Las operaciones soportadas por un PA son independientes del servicio.

Para cada sesión de acceso concurrente que un usuario tiene con un proveedor, hay un PA en el dominio del usuario. Cada PA puede estar asociado (a través de una sesión de acceso) con el mismo UA, o con UA distintos. Un solo PA está asociado únicamente con un UA a través de una sesión de acceso. (Cuando no existen sesiones de acceso, el dominio de usuario puede aún soportar un PA. Puede ser utilizado para iniciar una sesión de acceso, y puede recibir invitaciones, si está registrado.)

6.1.3 Agente inicial

Un agente inicial (IA, *initial agent*) es un componente independiente del usuario y del servicio que es el punto de acceso inicial a un dominio. Un IA es soportado por dominios que desempeñan el cometido proveedor. Se devuelve una referencia de IA a un PA cuando éste desea ponerse en contacto con el dominio. El IA soporta capacidades[8] para:

– autenticar el dominio solicitante y establecer una relación fiable entre los dominios (una sesión de acceso) interactuando con el PA;

– establecer una sesión de acceso, pero permitiendo que el dominio solicitante permanezca anónimo. El tipo de UA accedido de esta manera es un agente de usuario anónimo.

---

[5] El PA puede utilizar un servicio de localización para hallar una referencia de interfaz para el IA, o algún otro medio. El PA proporcionará el nombre de detallista, y posiblemente otra información para la búsqueda del servicio de localización. La capacidad del servicio de localización de devolver esta referencia de interfaz es una parte importante para permitir el acceso con independencia de la ubicación, que es una característica importante de la movilidad personal. Esto supone que es posible ponerse en contacto verdaderamente con el servicio de localización, con independencia de la ubicación. Asimismo, puede suponer que se requiere interfuncionamiento entre servicios de localización en diferentes dominios. No se ha definido cómo el servicio de localización obtiene una referencia de interfaz de un agente inicial. Es probable que el servicio de localización tenga que interactuar con un objeto en el dominio del proveedor para obtener la referencia. Esta interacción no está definida actualmente.

[6] Esta capacidad es un elemento importante para soportar la movilidad personal. Porque permite que un usuario acceda a un dominio de proveedor desde diversas ubicaciones.

[7] Utilizando una UAP relacionada con sesión de acceso.

[8] Estas capacidades están disponibles con independencia de la ubicación del PA con el cual interactúa el IA, y por tanto son una parte importante del soporte a la movilidad personal, es decir, permiten el acceso de usuario con independencia de la ubicación.

Las operaciones soportadas por un IA son independientes del servicio.

Un IA soporta peticiones de un PA a la vez. El PA solicita ponerse en contacto con el dominio y recibe una referencia a un IA. Cuando el PA ha interactuado con el IA para establecer una sesión de acceso con un UA, la referencia al IA será no válida[9]. A continuación, el otro PA puede ponerse en contacto con el IA.

6.1.4 Agente de usuario

Un agente de usuario (UA, *user agent*) es un componente independiente del servicio que representa a un usuario en el dominio del proveedor. Es soportado por un dominio que desempeña el cometido proveedor de acceso. Es el punto extremo del dominio de proveedor de una sesión de acceso con un usuario. Soporta la sesión de acceso de dominio de proveedor. Es accesible desde el dominio del usuario, prescindiendo de la ubicación del dominio.

Un UA soporta capacidades para:

– establecer una relación fiable entre el usuario y el proveedor (una sesión de acceso) haciendo referencia al PA del usuario;

– dentro de una sesión de acceso:

  • actuar como un solo punto de contacto del usuario para controlar y gestionar (crear/suspender/reanudar) el ciclo de vida de sesiones de servicio y sesiones de servicio de usuario;

  • crear una nueva sesión de servicio (pidiendo que la fábrica de servicios cree un USM y SSM);

  • incorporarse a una sesión de servicio existente creando una nueva sesión de servicio de usuario (mediante una fábrica de servicios que crea un USM);

  • resolver el entorno de ejecución del servicio del usuario, permitiéndole utilizar servicios de muchos tipos de terminales diferentes. Esto requiere información de configuración de recursos del sistema del usuario (que comprende terminales y sus puntos de acceso que son utilizados por el usuario o están disponibles para éste). El acceso a esta información puede ser restringida por el usuario/PA;

  • proporcionar acceso a la información del contrato del usuario con el proveedor;

  • resolver problemas de interacción entre peticiones de utilización de servicio.

El UA se define como un tipo de componente abstracto (es decir, no ejemplificable). Se definen dos subtipos ejemplificables:

– agente de usuario denominado (namedUA);

– agente de usuario anónimo (anonUA).

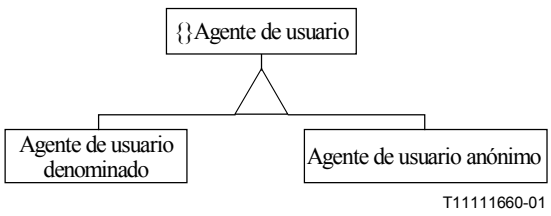


Figura 6-1 – Jerarquía de herencia para el agente de usuario

[9] Es decir, el PA no debe retener una referencia a un IA después que ha establecido una sesión de acceso. Una implementación puede imponer que la referencia al IA no sea utilizable una vez establecida la sesión de acceso.

Los subtipos se crean para representar diferentes tipos de usuario. Los subtipos de UA soportan todas las capacidades que son definidas para el UA.

El UA denominado es un UA especializado para un usuario que es un usuario de extremo o abonado del proveedor.

El UA anónimo es un UA especializado para un usuario que no desea revelar su identidad al proveedor.

A continuación se definen estos subtipos.

6.1.5 Agente de usuario denominado

El agente de usuario denominado (namedUA, *named user agent*) es un componente independiente del servicio que representa a un usuario en el dominio del proveedor. Es una especialización de UA para un usuario que es un usuario de extremo o abonado del proveedor. Es el punto extremo del dominio del proveedor de una sesión de acceso con un usuario. Es accesible desde el dominio del usuario, prescindiendo de la ubicación de dominio.

El UA denominado soporta todas las capacidades que se definen para UA y además las siguientes:

- dentro de una sesión de acceso:
  - actuar como un solo punto de contacto para controlar y gestionar (crear/suspender/reanudar) el ciclo de vida de sesiones de servicio y sesiones de servicio de usuario, teniendo en cuenta las restricciones impuestas por los abonados y el usuario;
  - suspender/reanudar sesiones de servicio de usuario y sesiones de servicio existentes, lo que incluye soporte para la movilidad de sesión;
  - gestionar las preferencias del usuario (opciones o constricciones) en cuanto al acceso de servicio y a la ejecución del servicio (esto se efectúa comenzando una sesión de servicio específica de proveedor);
  - resolver el entorno de ejecución de servicio para el usuario, permitiéndole utilizar servicios de muchos tipos diferentes de terminales. Esto requiere información de configuración de recursos del sistema de usuario (que comprende terminales y sus puntos de acceso que son utilizados por el usuario o están disponibles para éste. El acceso a esta información puede ser restringida por el usuario/PA). Esto incluye soporte de la movilidad personal;
  - registrar al usuario un terminal para recibir invitaciones, lo que incluye soporte para la movilidad personal;
  - permitir que el usuario defina políticas privadas/públicas de usuario. (Esto se hace comenzando una sesión de servicio específica de proveedor.)
  - negociar los modelos de sesión y conjuntos de características soportados por una sesión de servicio, para interactuar con una UAP en el dominio del usuario;
- aceptar invitaciones de los usuarios para incorporarse a una sesión de servicio;
- entregar invitaciones a un terminal, previamente registrado por el usuario con el UA denominado. No se requiere una sesión de acceso para permitir esta entrega de invitaciones.

El UA denominado puede soportar las siguientes capacidades facultativas:

- ejecutar acciones en nombre de los usuarios, cuando éstos no están en una sesión de acceso con el UA denominado;
- iniciar una sesión de acceso con un PA;
- soportar la autenticación adicional del usuario. Esto se puede adaptar al contexto del usuario y de la utilización.

Las operaciones soportadas por un UA denominado son independientes del servicio.

Un UA denominado puede soportar una o más sesiones de acceso concurrentemente[10]. Cada sesión de acceso es con un PA distinto.

6.1.6 Agente de usuario anónimo

Un agente de usuario anónimo (anonUA, *anonymous user agent*) es un componente independiente del servicio que representa a un usuario en el dominio del proveedor. Es una especialización de UA para usuarios que no desean revelar su identidad al proveedor. Es el punto extremo del dominio del proveedor de una sesión de acceso con el usuario anónimo.

El UA anónimo soporta todas las capacidades que se definen para el UA y además las siguientes:

– Soportar una relación fiable entre el usuario y el proveedor (una sesión de acceso) haciendo referencia al PA del usuario. El proveedor no conoce la identidad del usuario (la "fiabilidad" no está garantizada identificando el usuario, como para el UA denominado, pero puede ser asegurada, por ejemplo, mediante pago previo).

– dentro de una sesión de acceso:
  - Suspender/reanudar sesiones de servicio de usuario y sesiones de servicio existentes dentro de una sesión de acceso. (Las sesiones suspendidas no pueden ser reanudadas en una sesión de acceso diferente.[11])
  - Gestionar las preferencias del usuario (opciones o constricciones) en el acceso de servicio y la ejecución del servicio. (Éstas tendrían que ser determinadas durante la sesión de acceso y no podrán ser reutilizadas en una sesión de acceso distinta.)
  - Proporcionar acceso a información del contrato del usuario con el proveedor. (Este contrato se definiría al comienzo de la sesión de acceso y terminaría al final de la sesión de acceso.)
  - Definir políticas privadas/públicas de usuario. (Esto se puede hacer comenzando una sesión de servicio específica de proveedor. Esta información se mantendría solamente durante esta sesión de acceso.)
  - Permitir que el usuario anónimo se registre como un usuario del proveedor (es decir, establecer un contrato con el proveedor para más de una sesión de acceso).
  - Negociar los modelos de sesión y conjuntos de características soportados por una sesión de servicio, para interactuar con una UAP en el dominio de usuario.

El UA anónimo no soporta la movilidad personal o de sesión.

6.2 Componentes relacionados con sesión de servicio

Los componentes relacionados con sesión de servicio definidos a continuación siguen los conceptos de sesión indicados en [6]. Las sesiones de servicio pueden ser soportados en múltiples dominios. Las sesiones relacionadas con servicio[12] son sustentadas por estos componentes.

Las interacciones entre los dominios dependen del cometido que desempeña el dominio en la sesión de servicio. Un dominio que desempeña el cometido de proveedor de utilización soportará un gestor

---

[10] Los UA denominados deben ser capaces de soportar una sesión de acceso y facultativamente múltiples sesiones de acceso concurrentes. Los UA denominados continúan existiendo cuando no hay sesión de acceso.

[11] Se supone que las sesiones suspendidas son terminadas por el proveedor si la sesión de acceso es terminada.

[12] Sesión de servicio (SS), sesión de servicio de proveedor (PSS), sesión de servicio de utilización (USS), sesión de servicio de utilización de dominio de proveedor (PD_USS) y sesión de servicio de utilización de dominio de usuario (UD_USS).

de sesión de servicio y también gestores de sesión de servicio de usuario para cada dominio que desempeña el cometido parte de utilización. El dominio de parte soporta una UAP relacionada con sesión de servicio para interactuar con el USM.

Un dominio puede ser percibido como un dominio de parte por el dominio de proveedor, mientras que está realmente componiendo esta sesión de servicio por sí mismo.

Los modelos de sesión definen cómo los componentes de sesión de servicio en cada dominio pueden interactuar de una manera genérica. Estos modelos de sesión permiten que componentes que han sido diseñados e implementados separadamente, interactúen para soportar la sesión de servicio.

El modelo de sesión permite que los componentes de sesión de servicio soliciten, por ejemplo, la terminación y suspensión de la sesión, los participantes, el establecimiento y modificación de vinculaciones de trenes entre partes.

Los componentes relacionados con sesión de servicio pueden soportar uno o más de una variedad de modelos de sesión, que pueden ser definidos por una variedad de organizaciones. Cada sesión puede soportar varios modelos de sesión, o sólo puede soportar un modelo. Los servicios pueden decidir no soportar el modelo de sesión propuesto. Esto es aceptable porque la parte de acceso incluye la posibilidad de negociar interfaces de utilización alternativa.

6.2.1 Aplicación de usuario

La aplicación de usuario (UAP) se define para modelar una variedad de aplicaciones y programas en el dominio de usuario. Una UAP representa una o más de estas aplicaciones y programas en el modelo computacional. Una UAP puede ser utilizada por usuario humanos y/u otras aplicaciones en el dominio de usuario. Una UAP puede ser un componente relacionada con sesión de acceso o un componente relacionado con sesión de servicio, o ambos. La UAP relacionada con sesión de acceso se define en 6.1.1. La UAP relacionada con la sesión de servicio se define a continuación.

Como un componente relacionado con sesión de servicio, la UAP permite que un usuario utilice las capacidades de una sesión de servicio, mediante una interfaz de usuario apropiada. Actúa como un punto extremo de una sesión de servicio, soportando la sesión de servicio de utilización de dominio de usuario (UD_USS). Las capacidades proporcionadas por determinadas UAP son específicas de la UAP, y cualquier sesión de servicio es una parte de la misma. Las UAP pueden proporcionar algunas de las siguientes capacidades genéricas a los usuarios:

- comienzo/terminación de la sesión;
- invitación a otros usuarios para incorporarse a la sesión;
- incorporación a una sesión de servicio existente;
- adición/supresión/modificación de vinculaciones de trenes y la participación de los usuarios en las mismas;
- establecimiento de relaciones de sesión de control y otros cambios en la sesión de servicio;
- suspensión de la participación del usuario en la sesión, o toda la sesión;
- reanudación de la participación del usuario en la sesión, o toda la sesión.

Es posible incorporar ninguna[13] o más interfaces de trenes a una UAP. Las interfaces de trenes pueden estar vinculadas a las de otros sistemas de usuario y/o a las del dominio del proveedor.

El dominio de usuario contiene una o más UAP relacionadas con sesión de servicio. Una UAP relacionada con sesión de servicio puede participar en una o más sesiones de servicio. Para cada

[13] Algunos servicios no requieren interfaces de trenes en la UAP.

sesión de servicio en la cual la UAP participa, interactúa con un gestor de sesión de servicio de usuario, o con un gestor de sesión de servicio[14].

La UAP puede soportar también un modelo de sesión particular, tal como el modelo de sesión propuesto. El USM/SSM utiliza estas interfaces para compartir información con la UAP sobre las partes y recursos de servicio en la sesión de servicio.

Una UAP puede soportar capacidades relacionadas solamente con sesión de acceso, capacidades relacionadas solamente con sesión de servicio, o ambas. Las UAP relacionadas con sesión de servicio estarán especializadas para las sesiones de servicio con las que interactúa. Para utilizar una sesión de servicio, una UAP especializada para el tipo de servicio debe estar presente en el dominio del usuario (o ser instalada, por ejemplo, por telecarga) antes de la sesión.

6.2.2 Fábrica de servicios

Una fábrica de servicios (SF, *service factory*) es un componente específico de servicio que crea los componentes de sesión de servicio para un tipo de servicio.

La petición de crear una sesión de servicio de un tipo de servicio determinado resultará en la creación de uno o más componentes relacionados con servicios[15]. La SF creará e inicializará los casos de acuerdo con las reglas impuestas por su implementación y devolverá al cliente una o más referencias de interfaz a estos componentes, (La SF se utiliza para crear todos los componentes relacionados con sesión de servicio definidos en este documento: USM y SSM.)

Las peticiones suelen ser hechas por los UA. Otros clientes pueden también pedir la creación de una sesión de servicio. El cliente debe tener una referencia de interfaz con la SF y emitir una petición apropiada. Una SF que soporta más de un tipo de servicio proporcionará normalmente interfaces separadas para cada tipo de servicio.

La SF soporta capacidades para:

– Crear componentes relacionados con servicio para uno o más tipos de servicio a petición. (Esto incluye la elección de los modelos de sesión soportados por la sesión de servicio, aunque esto puede ser fijado por el tipo de servicio.)

La SF puede soportar capacidades facultativas para:

– Crear un componente relacionado con el servicio (típicamente USM) que se ha de utilizar junto con otros componentes relacionados con servicio (típicamente un SSM y varios USM) creados por una fábrica diferente.
– Continuar la gestión de los componentes creados. Puede proporcionar una lista de sesiones gestionadas, y puede "limpiar" algunas sesiones, si así se solicita.
– Puede incluir mecanismos para programar la activación de una sesión en una fecha y hora específicas. (Este mecanismo incluye la reserva de recursos.)
– Soportar la suspensión/reanudación de una sesión de servicio.

La SF reúne los recursos necesarios para la existencia del componente que crea. Por tanto, la SF representa un ámbito de asignación de recursos, que es el conjunto de recursos disponibles para la SF. Una SF puede soportar una interfaz que permita a los clientes restringir el alcance.

---

[14] Los servicios que sólo soportan un usuario en una sesión de servicio sólo pueden proporcionar un SSM (ningún USM) y permiten que la UAP interactúe directamente con el SSM. Estos servicios estarán restringidos a tener sólo un usuario en una sesión de servicio, y no deben poder invitar a otros usuarios a que se incorporen a la sesión, pues ningún USM podrá ponerse a disposición del usuario invitado.

[15] Generalmente, el USM y el SSM.

6.2.3 Gestor de sesión de servicio

El gestor de sesión de servicio (SSM, *service session manager*) es un componente que comprende segmentos de control de sesión específicos del servicio y genéricos de la sesión de servicio de proveedor (PSS). Un SSM soporta capacidades de servicio que son compartidas entre miembros (partes, recursos de servicio, etc.) en una sesión de servicio. La información relacionada con un miembro particular de la sesión de servicio está encapsulada en gestores de sesión de servicio de utilización de miembro (MUSM, *member usage service session managers*). Los SSM soportan (algunas o todas) las siguientes capacidades:

- seguir la pista y controlar los diversos recursos compartidos por múltiples usuarios en una sesión de servicio. Esto se puede hacer teniendo referencias a otros objetos (como un CSM) que mantiene realmente el contexto de utilización para una clase específica de recursos;
- mantener el estado de una sesión de servicio y soportar su suspensión/reanudación;
- soportar la adición/invitación/supresión de usuarios a/de la sesión de servicio interactuando con los UA correspondientes;
- soportar la adición/supresión/modificación de vinculaciones de trenes y la participación de los usuarios en los mismos;
- soportar la negociación de capacidades entre los usuarios que interactúan con los USM. El SSM servirá como un centro de control de construcción de consenso (como los procedimientos de votación);
- soportar capacidades de gestión asociadas con la sesión de servicio (por ejemplo, contabilidad).

Es posible incorporar ninguna[16] o más interfaces de trenes a un SSM. Las interfaces de trenes pueden estar vinculadas a otras interfaces de trenes en este u otros dominios.

Un SSM es creado por una SF, uno por petición para el correspondiente tipo de servicio. Es suprimido cuando todos los usuarios dejan la sesión de servicio, o cuando la sesión es abandonada por un usuario o SF. La duración de un SSM es igual que la correspondiente sesión de servicio de proveedor.

6.2.4 Gestor de sesión de servicio de utilización de miembro

El gestor de sesión de servicio de utilización de miembro (MUSM) es un componente abstracto, que comprende los segmentos de control de sesión específico del servicio y genéricos de la sesión de servicio de utilización de dominio (D_USS) que interactúa con el PSS. Está especializado de acuerdo con el cometido de miembro de sesión desempeñado por la D_USS:

- El gestor de sesión de usuario de servicio (USM) representa la UD_USS.

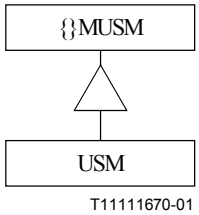


Figura 6-2 – Jerarquía de herencia para el gestor de sesión de servicio de utilización de miembro

---

[16] Típicamente las interfaces de trenes son ofrecidas por componentes de soporte de servicio (SSC, *service support components*) asociados con un SSM.

El MUSM representa y mantiene el contexto de un miembro (parte, recurso de servicio o proveedor) en una sesión de servicio. Sus principales características son las siguientes:

– Contiene la información y capacidades de servicio locales del miembro. Si una operación conlleva actividades que son puramente locales para el miembro, el MUSM controla y gestiona las actividades por sí mismo. Si no, el MUSM interactúa con el SSM para ejecutar la operación. El SSM puede interactuar con el MUSM en respuesta a operaciones de otros miembros (o debido a la lógica de servicio) que afectan a este miembro. Estas interacciones dependen del cometido del miembro.

– Mantiene el seguimiento de los recursos exclusivos (no compartidos) utilizados por el miembro y los controla en una sesión de servicio. Esto se puede hacer teniendo referencias a otros objetos (por ejemplo, un gestor de sesión de comunicación) que mantiene realmente el contexto de un miembro para una clase de recurso específica.

– Puede ser configurado según las preferencias del miembro. Esto se puede hacer durante la inicialización y dinámicamente durante la sesión.

Como el MUSM es un componente de servicio abstracto, no se crean ejemplares del mismo. Se crean componentes especializados apropiados para representar miembros de sesión específico.

6.2.5 Gestor de sesión de servicio de usuario

El gestor de sesión de usuario (USM, *user service session manager*) es un componente que comprende segmentos de control de sesión específicos de servicio y genéricos de la sesión de servicio de usuario de dominio de proveedor (PD_USS). Es una especialización del MUSM que representa y mantiene el contexto de una parte[17], o recurso en una sesión de servicio. Tiene las mismas características que el MUSM (sustituyendo el miembro por la parte o recurso de servicio, según proceda):

– mantiene el estado de la PD_USS y soporta la suspensión/reanudación de la participación de cada parte en la sesión de servicio;

– desempeña los diferentes cometidos de la parte en el servicio. El cometido de una parte será independiente del servicio (por ejemplo, presidente en una conferencia).

Es posible incorporar ninguna[18] o más interfaces de trenes a un USM. Las interfaces de trenes pueden estar vinculadas a otras interfaces de trenes en éste u otros dominios.

Un USM es creado por la SF, uno por petición para el correspondiente tipo de servicio (por cada PD_USS). Se suprime cuando la parte deja la sesión de servicio. La duración de USM es igual que la correspondiente PD_USS.

6.3 Componentes relacionados con recursos (sesión de comunicación)

Obsérvese que los servicios que no utilizan vinculaciones de trenes no tendrán recursos (sesiones de comunicación) y no necesitarán estos componentes. Sin embargo, como los componentes son independientes del servicio, es probable que un CSM o TCSM esté presente en la mayoría de los dominios.

6.3.1 Gestor de sesión de comunicación

El gestor de sesión de comunicación (CSM, *communication session manager*) es un componente independiente del servicio que gestiona vinculaciones de extremo a extremo a nivel de aplicación

---

[17] Una parte puede ser un usuario de extremo, o una sesión de servicio que desempeña un cometido de parte de utilización.

[18] Normalmente las interfaces de trenes son ofrecidas por componentes de soporte de servicio (SSC) asociadas con un USM.

entre interfaces (conexiones de flujo de trenes). Un flujo de trenes es una abstracción de una conexión. El CSM proporciona interfaces para que los USM/SSM puedan establecer, modificar y suprimir flujos de trenes. El CSM descompone la conexión solicitada en dos partes, la parte nodal y la parte de transporte. Pide al TCSM que se ocupe de la parte nodal y pide a otros objetos de gestión de conexión que se ocupen de la parte transporte.

El CSM proporciona las siguientes capacidades:

– Creación y control de conexiones de flujo de trenes (SFC, *stream flow connections*) de extremo a extremo.

6.3.2 Gestor de sesión de comunicación de terminal

El gestor de sesión de comunicación de terminal (TCSM, *terminal communication session manager*) es un componente independiente del servicio que gestiona las conexiones de flujo de terminal (TFC, *terminal flow connections*) (conexiones de flujo intranodales) dentro del dominio del usuario. El TCSM proporciona una interfaz al CSM, para que éste pueda pedir al TCSM que establezca, modifique y suprima conexiones en el dominio del usuario.

El TCSM proporciona la siguiente capacidad:

– creación y control de conexiones de flujo (TFC) dentro del dominio del usuario.

6.4 Relación con el modelo de información

Los cuadros 6-1 y 6-2 proporcionan una correspondencia entre conceptos y objetos de sesión en el modelo de información y componentes en el modelo computacional.

La correspondencia de los conceptos de sesión con un componente significa que el componente soporta la funcionalidad y estado de la sesión y controla los recursos de servicio que forman parte de la sesión. Si un concepto de sesión corresponde con varios componentes, cada uno de éstos soporta parte de la funcionalidad y estado, y controla algunos de los recursos de servicio de la sesión.

La correspondencia de los objetos de información con un componente significa que la información en el objeto de información está contenida dentro del componente, y que el componente puede proporcionar acceso a esa información a otros componentes/objetos.

Cuadro 6-1 – Correspondencia entre componentes relacionados con sesión de acceso

Concepto de sesión/ Objetos de información	Componentes
Sesión de acceso (AS) con cometidos de proveedor de usuario	PA y UA
Sesión de acceso de dominio de usuario (UD_AS)	PA
Sesión de acceso de dominio de proveedor (PD_AS)	UA
Perfil de usuario con cometidos de proveedor-usuario	UA
Contrato con cometidos de proveedor-usuario	PA y UA

Cuadro 6-2 – Correspondencia entre componentes relacionados con sesión de servicio

Concepto de sesión/ Objetos de información	Componentes
Sesión de servicio (SS)	UAP, USM y SSM relacionados con sesión de servicio
Sesión de servicio de usuario (USS)	UAP y USM relacionados con sesión de servicio
Sesión de servicio de utilización de dominio de usuario (UD_USS)	UAP relacionada con sesión de servicio
Sesión de servicio de utilización de dominio de proveedor (PD_USS)	USM
Sesión de servicio de proveedor (PSS)	SSM

6.5 Ejemplos

A continuación se describen varios escenarios de ejemplo para ilustrar cómo los componentes pueden interactuar en los siguientes casos:

- puesta en contacto con un proveedor;
- registro con un proveedor como un usuario conocido;
- comienzo de una nueva sesión de servicio;
- invitación a un usuario a incorporarse a una sesión de servicio existente;
- incorporación a una sesión de servicio existente;
- creación de una vinculación de trenes en una sesión de servicio existente.

Obsérvese que los escenarios son ejemplos y que en aras de la sencillez se suponen que todas las operaciones se completan satisfactoriamente (sin errores, sin averías y sin rechazo).

6.5.1 Puesta en contacto con un proveedor

Este ejemplo muestra al usuario A que se pone en contacto con un proveedor (véase la figura 6-3). Este escenario soporta la movilidad del usuario permitiendo que éste se ponga en contacto con un proveedor específico desde cualquier terminal.

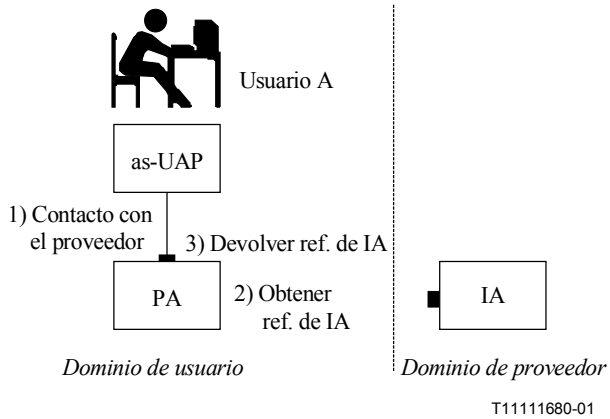


Figura 6-3 – Puesta en contacto con un proveedor

Condiciones previas:

Un PA debe estar presente en el dominio del usuario.

Escenario:

1) El usuario comienza una UAP relacionada con sesión de acceso. Proporciona el nombre del detallista con el cual desea ponerse en contacto. La UAP solicita al PA que se ponga en contacto con el proveedor, dando el nombre de detallista.
2) El PA obtiene una referencia a una interfaz de un agente inicial del proveedor[19].
3) El PA responde satisfactoriamente a la UAP.

Condiciones posteriores:

El PA tiene una interfaz de referencia con el IA. El usuario no ha establecido una sesión de acceso entre el PA y el IA. El IA no soporta el uso de servicios de usuario, sólo operaciones para establecer una sesión de acceso con un usuario conocido (véase 6.5.2) o un usuario anónimo.

El IA no conoce ninguna interfaz en el PA ni en el dominio del usuario.

Es posible que el proveedor telecargue un PA específico de proveedor al dominio del usuario, una vez que el PA ha obtenido una referencia de interfaz para el IA. Esto facilita la movilidad del usuario. Actualmente no se ha definido ningún escenario que describa cómo se logra esto.

6.5.2 Registro con un proveedor como un usuario conocido

Este ejemplo muestra al usuario A que establece una sesión de acceso con su agente de usuario denominado del proveedor (véase la figura 6-4). El usuario desea utilizar los servicios del proveedor a los cuales se ha abonado previamente (véase 6.5.3). Este escenario apoya la movilidad de usuario permitiendo que éste establezca una sesión de acceso con un proveedor desde cualquier terminal.

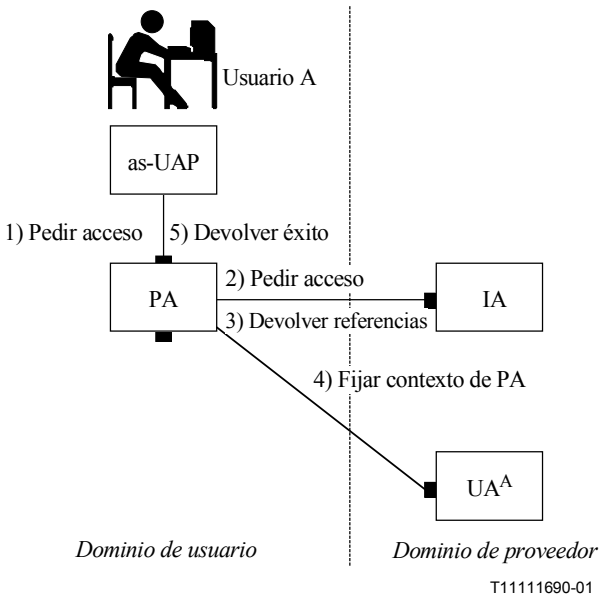


Figura 6-4 – Registro con un proveedor como un usuario conocido

[19] El PA puede utilizar un servicio de localización para hallar una referencia de interfaz para el IA, o algún otro medio.

Condiciones previas:

El usuario se pone en contacto con el proveedor (como en 6.5.1) y el PA tiene una referencia de interfaz a un agente inicial del proveedor.

Escenario:

1) El usuario A utiliza una UAP relacionada con sesión de acceso para registrarse con el proveedor como un usuario conocido[20]. El usuario solicita al PA que los registre con el proveedor como un usuario conocido. La UAP suministra la información de seguridad al PA.

2) El PA solicita que se establezca una sesión de acceso con el UA denominado del usuario. El PA proporciona el nombre de usuario al IA. El PA ha pasado la información de seguridad de usuario a los servicios de seguridad soportados por el DPE. Los servicios de seguridad interactúan con el dominio de proveedor para autenticar al usuario[21].

3) El IA ya ha autenticado al usuario a través de los servicios de seguridad de DPE, y se ha establecido una sesión de acceso. Devuelve la referencia de interfaz del UA del usuario.

4) El PA envía información sobre el dominio de usuario al UA. Esta información figura en el contexto del PA, que incluye referencias a interfaces en el PA, y posiblemente información de terminal.

5) El PA comunica a la UAP que la operación ha tenido éxito.

Condiciones posteriores:

El usuario ha establecido una sesión de acceso entre el PA y el UA denominado. El UA denominado está personalizado para el usuario, y conoce las interfaces del PA.

Todas las referencia de interfaz del IA mantenidas por el PA serán no válidas.

Obsérvese que en la secuencia de eventos, se permite la movilidad personal debido a las siguientes capacidades:

a) la capacidad de ponerse en contacto con el servicio de denominación prescindiendo de la ubicación y devolver una referencia al IA;

b) el IA establece una relación fiable con el usuario, que es independiente de la ubicación física de dicho usuario;

c) el IA devuelve una referencia del UA denominado propio del usuario al usuario (PA que actúa en nombre de éste).

Es posible que una vez establecida la sesión de acceso, el proveedor telecargue un PA específico de proveedor al dominio de usuario, lo que facilita la movilidad de usuario. Actualmente no se ha definido ningún escenario que describa cómo se logra esto.

6.5.3 Comienzo de una nueva sesión de servicio

Este ejemplo muestra a un usuario que comienza una nueva sesión de servicio (en este ejemplo, un servicio de videoconferencia, pero la interacción sería igual para todos los tipos de servicio). Se supone que el usuario esté en una sesión de acceso con el proveedor y que tenga un abono válido al servicio (el tipo de servicio es videoConference234). Se supone que la UAP relacionada con la sesión de servicio esté presente en el terminal del usuario (véase la figura 6-5).

---

[20] La UAP puede pedir al usuario su nombre de usuario y otra información relacionada con la seguridad, por ejemplo, contraseña.

[21] Si el DPE no soportase servicios de seguridad, el PA tendría que enviar primero la información de autenticación al IA.

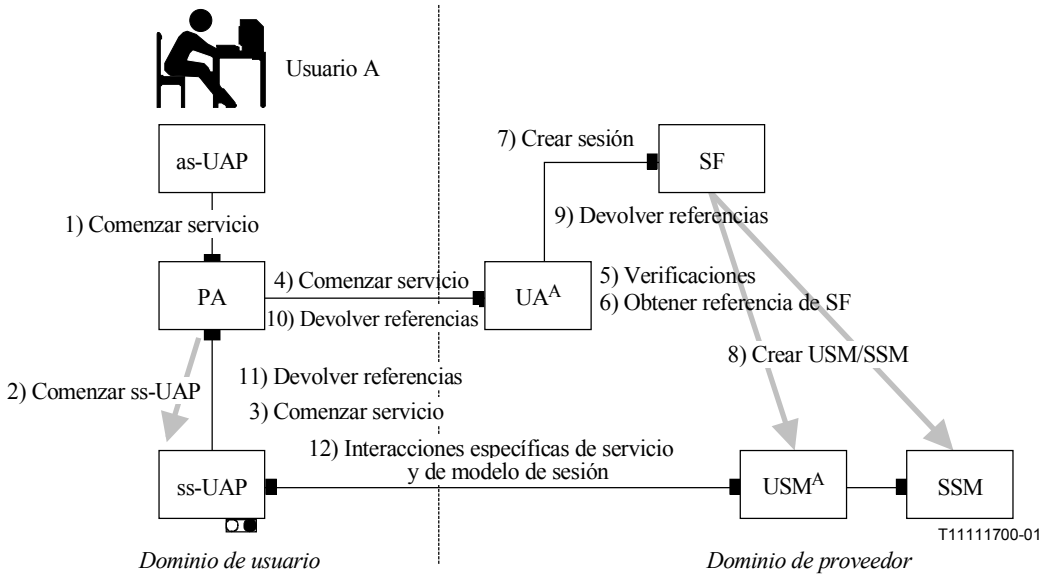


Figura 6-5 – Comienzo de una nueva sesión de servicio

Condiciones previas:

Existe una sesión de acceso entre el PA (usuario A) y el UA (en el dominio de proveedor). Una UAP relacionada con sesión de acceso muestra al usuario los servicios que puede comenzar.

Escenario:

1) La UAP relacionada con sesión de acceso pide al PA una lista de los servicios a los que el usuario está abonado. El PA hace la misma petición al UA, que devuelve la lista. El UAP visualiza la lista al usuario. El usuario selecciona un servicio para comenzar[22]. El UAP pide al PA que comience el servicio.

2) El PA comienza la UAP relacionada con sesión de servicio[23] asociada con esta sesión de servicio y le informa el tipo de servicio que debe comenzar (videoConference234).

3) La UAP relacionada con sesión de servicio pide una nueva sesión de servicio del tipo de servicio videoConference234 al PA. (La UAP puede pasar información sobre sí misma al PA, incluidos modelos de sesión y conjuntos de características soportados y referencias a sus interfaces operacionales y de trenes.)

4) El PA pide al UA (del usuario A) que comience una nueva sesión de servicio del tipo de servicio (videoConference234). (Puede también pasar la información sobre la UAP.)

5) El UA puede ejecutar algunas acciones, que no están prescritas aquí, antes de continuar. Por ejemplo, el UA comprueba la petición de nueva sesión contra el perfil de abono del usuario[24] para verificar que el usuario está abonado a este servicio y que éste puede ser utilizado con la configuración de terminal del usuario. Se pueden tomar también otras decisiones. El UA plantea una excepción al PA, si el UA declina comenzar la sesión de servicio.

---

[22] Las interacciones precedentes no se muestran en la figura.

[23] Si la UAP relacionada con sesión de servicio no está disponible en el dominio del usuario, el PA puede tratar de telecargar la UAP y continuar.

[24] El perfil de abono del usuario puede definir preferencias y constricciones en la invocación de un servicio, que pueden depender de la ubicación del usuario en ese momento. Esto proporciona soporte para la movilidad personal.

6) El UA obtiene una referencia a una fábrica de servicios que puede crear componentes de sesión de servicio para el tipo de servicio (videoConference234).[25]

7) El UA pide que la fábrica de servicios cree una nueva sesión del tipo de servicio (videoConference234).

8) La fábrica de servicio crea un SSM y un USM[26] y los inicializa.

9) La fábrica de servicios devuelve referencias de interfaz del USM y del SSM al UA.

10) El UA devuelve referencias del USM y del SSM al PA.

11) El PA devuelve referencias del USM y del SSM a la UAP relacionada con sesión de servicio.

12) La UAP relacionada con sesión de servicio y el USM (y SSM) pueden interactuar utilizando interfaces específicas de servicios o interfaces definidas por modelos de sesión, que incluyen el modelo de sesión propuesto. Pueden ser necesarias algunas interacciones entre estos componentes antes de que el usuario pueda utilizar el servicio.

En este punto, el usuario A es el único participante en la sesión de servicio. Algunos servicios pueden ser servicios de un solo usuario, o pueden ser utilizados por un solo usuario. Como éste es un ejemplo de un servicio de videoconferencia, probablemente el usuario A desea invitar a algunos otros usuarios a incorporarse en la sesión.

6.5.4 Invitación a usuario a incorporarse a una sesión de servicio de sistema

Este ejemplo muestra el usuario A que invita a otro usuario (B) a incorporarse a la sesión de servicio (véase la figura 6-6). El ejemplo termina cuando la invitación ha sido entregada al usuario B. En 6.5.5 se muestra el ejemplo del usuario B que se incorpora a la sesión.

Este ejemplo supone que el usuario B invitado es un usuario denominado, y está representado en el proveedor por un agente de usuario denominado. Los usuarios anónimos, representados por un agente de usuario anónimo, no pueden ser invitados a incorporarse a una sesión de servicio porque no es posible localizar al usuario específico, puesto que los agentes de usuarios anónimos no publican la identidad del usuario (e incluso pueden no conocer la identidad del usuario).

Este escenario soporta la movilidad de usuario permitiendo que un usuario sea invitado a incorporarse a una sesión, prescindiendo de su ubicación. (Esto no significa que podrá incorporarse automáticamente a la sesión.)

---

[25] El UA pude utilizar un servicio de localización para hallar una fábrica de servicios apropiada, o algún otro medio. El UA puede también proporcionar otra información para la búsqueda de la fábrica de servicio, tal como información de configuración de terminal. Otros medios comprenden: la información de abono podrá contener una referencia de interfaz a la fábrica de servicio que se ha de utilizar.

[26] La fábrica de servicios crea los objetos computacionales que comprenden la sesión de servicio y que pueden incluir el USM y el SSM. Son posibles también otros objetos computacionales.

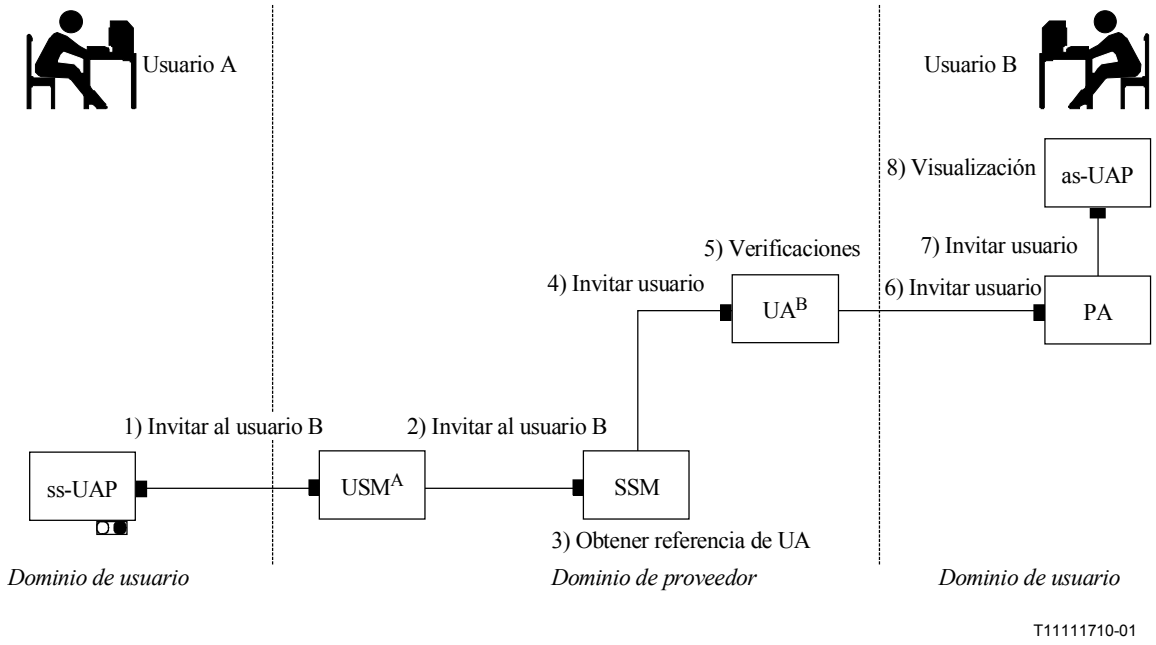


Figura 6-6 – Invitación a un usuario a incorporarse a una sesión de servicio existente

Condiciones previas:

Existe una sesión de acceso entre el PA y el UA del usuario que envía la invitación (usuario A). NO es necesario que exista una sesión de acceso entre un PA y el UA del usuario que recibe la invitación (usuario B) pero para este ejemplo se supone que existe una sesión de acceso para el usuario B.

El usuario A está utilizando una UAP relacionada con sesión de servicio y tiene una sesión de servicio establecida con un USM y un SSM. El usuario A desea invitar al usuario B a incorporarse a esta sesión de servicio. El usuario A está "activo" en la sesión de servicio, es decir, no se ha suspendido su participación.

Escenario:

1) El usuario A utiliza la UAP para invitar a otro usuario (invitado) a incorporarse a una sesión. (El usuario A suministra el nombre de usuario del invitado o una alias definido por el usuario que puede ser resuelto por el UA del invitante.) La UAP pide al USM que invite al usuario B a incorporarse a la sesión.
2) El USM pide al SSM que invite al usuario a incorporarse a la sesión. (El USM y el SSM pueden comprobar que el usuario A puede invitar al usuario B. Estas comprobaciones no se definen aquí.)
3) El SSM obtiene una referencia a una interfaz de invitación del UA del usuario B[27].
4) El SSM envía una invitación utilizando la interfaz de invitación del UA del usuario B.
5) El UA del invitado puede ejecutar algunas acciones, que no se indican aquí, antes de continuar. Por ejemplo, el UA puede verificar el perfil de usuario dentro del UA para una política de invitación. La política determinará las acciones e interacciones del UA con otros objetos. El UA puede plantear una excepción al SSM, si el UA declina entregar la invitación.

[27] El UA puede utilizar un servicio de localización para localizar al UA del usuario B, o algún otro medio.

6) En este ejemplo, existe una sesión de servicio entre el UA del usuario B y el PA en el terminal del usuario B. La invitación es entregada al PA, utilizando una interfaz de invitación en el PA.

7) El PA envía la invitación a la UAP relacionada con sesión de acceso.

8) La UAP visualiza la invitación al usuario B.

La invitación de incorporarse a la sesión de servicio ha sido entregada al UA, PA del usuario B, y es visualizada por la UAP. La invitación contiene suficiente información para que el UA localice la sesión de servicio y permita que el usuario se incorpore a la misma (como se describe en 6.5.5). Sólo parte de esta información será transferida al PA y a la UAP.

En el ejemplo anterior existe ya una sesión de acceso entre el PA y el UA del usuario B. Si el usuario B NO está actualmente en una sesión de acceso con el UA, pueden darse varias alternativas. Actualmente no se ha definido cuáles de estas alternativas deben ser soportadas como capacidades obligatorias y facultativas. Las alternativas son:

- El UA almacena la invitación hasta que el usuario invitado establece una sesión de acceso. Cuando la establece, se entrega la invitación como se indica anteriormente.
- El UA entrega la invitación a un terminal registrado. (El terminal habría sido seleccionado por el usuario para recibir invitaciones cuando no hay una sesión de acceso.)[28]
- El UA devuelve la dirección de un terminal registrado al SSM.
- El UA envía la invitación a otro UA. (Este UA habría sido seleccionado por el usuario para recibir invitaciones cuando no hay sesión de acceso. El UA puede estar en un dominio de proveedor diferente.)
- El UA devuelve la dirección de otro UA.
- El UA comienza una sesión de servicio de un tipo especificado. (La invitación puede ser enviada a la sesión de servicio, como parte de su configuración, o ulteriormente.)

6.5.5 Incorporación a una sesión de servicio existente

Este ejemplo muestra al usuario B que se incorpora a una sesión existente, tras recibir una invitación (véase la figura 6-7).

Se supone que el usuario B esté en una sesión de acceso con el proveedor y que tenga un abono válido al servicio (el tipo de servicio es videoConference234). Se supone que la UAP relacionada con sesión de servicio esté presente en el terminal del usuario B.

El usuario B puede incorporarse a esta sesión desde cualquier terminal, desde el cual ha establecido una sesión de acceso. Cuando esta sesión de acceso está establecida, el PA pide una lista de las invitaciones recibidas por el UA. El PA puede pedir después la incorporación a cualquiera de las sesiones. Sin embargo, en este ejemplo se supone que las invitaciones han sido entregadas al PA y as-UAP del usuario B, como se describe en 6.5.4.

[28] Este caso se requiere para soportar la movilidad personal.

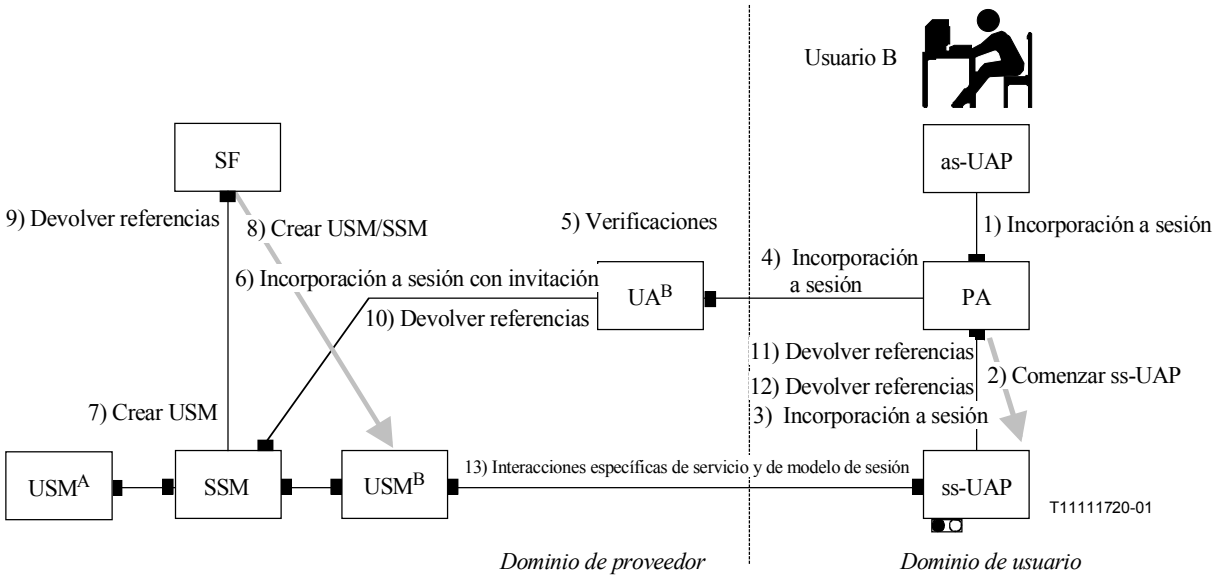


Figura 6-7 – Incorporación a una sesión de servicio existente

Condiciones previas:

Existe una sesión de acceso entre el PA (usuario B) y el UA (en el dominio del proveedor). El UA y el PA del usuario B han recibido la invitación de incorporarse a la sesión de servicio, y una UAP relacionada con sesión de acceso muestra al usuario la invitación que ha recibido.

Escenario:

1) La UAP relacionada con sesión de acceso visualiza una lista de invitaciones de incorporación a sesiones de servicio. El usuario selecciona una invitación. La UAP pide al PA la incorporación a la sesión de servicio, dando el ID de invitación.

2) El PA comienza una UAP relacionada con sesión de servicio[29], asociada con este tipo de sesión de servicio y le informa el identificador de invitación que debe pedir para la incorporación.

3) La UAP relacionada con sesión de servicio pide la incorporación a la sesión de servicio, dando la identificación de invitación del PA. (La UAP puede pasar información sobre sí misma al PA, incluidos los modelos de sesión soportados y las referencias a sus interfaces operacionales y de trenes.)

4) El PA solicita la incorporación a la sesión de servicio, dando el identificador de invitación al UA (del usuario B). (Puede también pasar la información sobre la UAP.)

5) El UAP puede ejecutar algunas acciones, que no se indican aquí, antes de continuar. Por ejemplo, el UA puede verificar el identificador de invitación contra las invitaciones vigentes del usuario, así como el perfil de abono del usuario[30] para verificar que está abonado a este servicio, y que puede ser utilizado con la configuración de terminal vigente, etc. El UA puede declinar que el usuario se incorpore a la sesión.

---

[29] Si la UAP relacionada con sesión de servicio no está disponible en el dominio del usuario, el PA puede intentar telecargar la UAP.

[30] El perfil de abono del usuario puede definir preferencias y constricciones en la invocación de un servicio, que pueden depender de la ubicación del usuario en ese momento. Esto proporciona soporte a la movilidad personal.

6) El UA obtiene una referencia al SSM[31] y pide la incorporación a la sesión. (Puede pasar alguna información en la invitación para confirmar que el SSM invitó a este usuario a incorporarse a la sesión.)

7) El SSM pide a su fábrica de servicios que cree un USM para el usuario B.

8) La fábrica de servicios crea un USM y lo inicializa.

9) La fábrica de servicios devuelve referencias de interfaz del USM al SSM.

10) El SSM devuelve referencias del USM y de sí mismo al UA.

11) El UA devuelve referencias del USM y del SSM al PA.

12) El PA devuelve referencias del USM y del SSM a la UAP relacionada con sesión de servicio.

13) La UAP relacionada con sesión de servicio y el USM (y SSM) pueden interactuar utilizando interfaces específicas de servicio o definidas por modelos de sesión, que incluyen el modelo de sesión propuesto. Puede ser necesario efectuar algunas interacciones entre estos componentes antes de que el usuario pueda utilizar el servicio.

En este punto, los usuarios A y B participan en la sesión de servicio. Como éste es un ejemplo de un servicio de videoconferencia, el usuario puede invitar a otros usuario a incorporarse a la sesión. Puede haber alguna política específica del servicio para decidir si un determinado usuario en la sesión puede invitar a otros usuarios o no.

6.5.6 Petición y establecimiento de una vinculación de trenes

Este ejemplo muestra el establecimiento de una vinculación de trenes entre las UAP en los dominios de los consumidores 1 y 2 (véase la figura 6-8). El consumidor 3 pide el establecimiento de una vinculación de trenes en la cual participen los consumidores 1 y 2, pero sin participar él mismo en la vinculación de trenes[32].

Condición previa:

Los tres consumidores ya han sido invitados a la sesión de servicio y participan en la misma.

---

[31] La invitación puede contener una referencia a una interfaz en el SSM para utilizar la incorporación a la sesión, o el UA puede utilizar información en la invitación junto con un servicio de localización para hallar el SSM, o algún otro medio. El UA puede proporcionar también otra información al SSM, tal como información de configuración de terminal e información de aplicación.

[32] Este ejemplo se muestra para ilustrar la separación entre sesión de servicio y sesión de comunicación. Naturalmente, son posibles otros ejemplos, a saber, uno (más similar a la telefonía ordinaria) donde el consumidor 1 actúa como el iniciador de la sesión de servicio y también como el iniciador de la sesión de comunicación. La separación de acceso y utilización y de sesión de servicio y sesión de comunicación tiene aún significado y añade valor (por ejemplo, apoya los aspectos de movilidad).

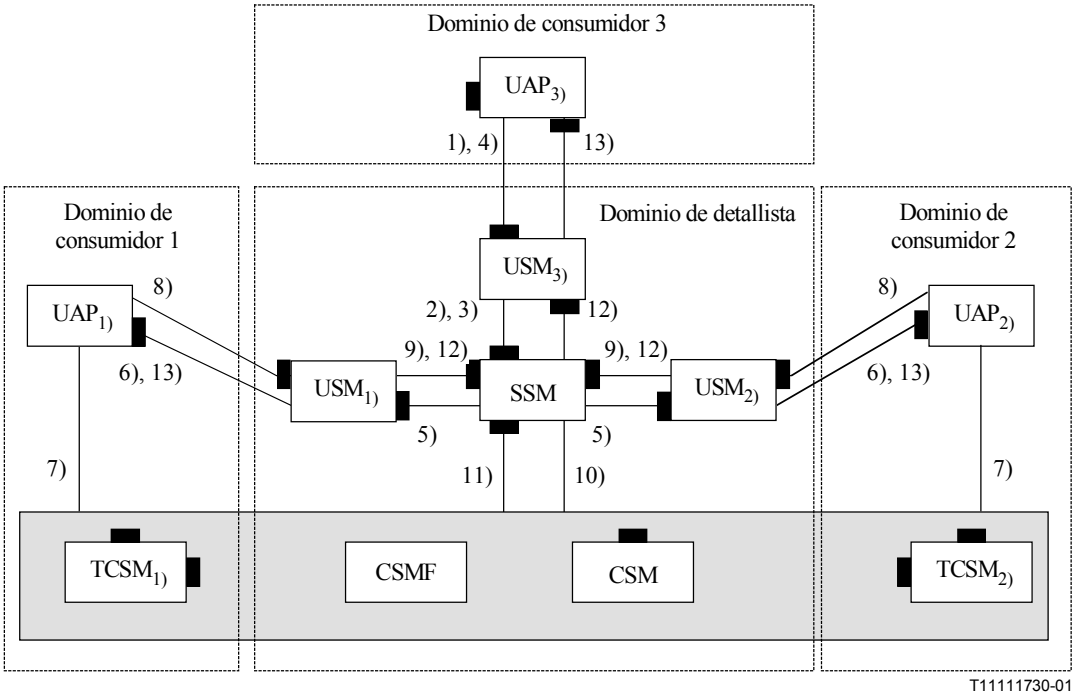


Figura 6-8 – Petición y establecimiento de vinculación de trenes [tres partes en la sesión de servicio, el recurso (sesión de comunicación) comprende dos de ellas]

Condición posterior:

Se establece una vinculación de trenes entre los dos participantes, parte 1 y parte 2, y la parte 3 tiene una referencia a la vinculación de trenes para controlarlo.

Escenario:

El escenario muestra un establecimiento satisfactorio, pero en algunos puntos es evidente que podrían haberse tomado decisiones diferentes.

1) La UAP3 pide el establecimiento de una vinculación de trenes con las partes 1 y 2 como participantes. El USM3 puede facultativamente efectuar las comprobaciones necesarias para asegurar que el consumidor 3 puede establecer la vinculación de trenes solicitada.

2) El USM3 envía la petición al SSM de la sesión de servicio.

   Facultativamente, el SSM puede verificar el permiso de establecer esta vinculación de trenes; si es necesario, puede negociar el permiso con los otros miembros en la sesión. [Esto conllevará la característica de votación (no mostrada).]

3) Si se obtiene el permiso, el SSM devuelve un identificador de vinculación de trenes, así como un identificador de petición para ulteriores confirmaciones al USM3.

4) El USM devuelve un identificador de vinculación de trenes, así como un identificador de petición, para ulteriores confirmaciones a la UAP3.

   (Lo siguiente se puede hacer en paralelo con "1" y "2", denominado en adelante "i".)

5) El SSM pide al USMi la incorporación a la vinculación de trenes. El USMi puede tomar decisiones facultativas, por ejemplo, sobre la no participación en nombre del consumidor i.

6) El USMi envía la invitación a la UAPi.

7) La UAPi comienza un escenario de establecimiento de aplicación para obtener el NFEP relacionado con el usuario de la interfaz de trenes por el consumidor i en este servicio. Esto

puede haber sido hecho ya[33], o debe hacerse para que el consumidor participe en esta vinculación de trenes.

8) La UAPi acepta y devuelve esta aceptación al USMi, junto con una descripción de las condiciones de participación del consumidor i en la vinculación de trenes, así como el descriptor de interfaz de trenes.

9) El USMi envía esta aceptación y la información asociada al SSM.

Dependiendo de la respuesta de los participantes (y de la lógica específica para este servicio), el SSM puede elegir abandonar la vinculación de trenes, o la petición de vinculación de trenes resultará en una petición de comunicaciones (esto es lo que se muestra).

10) El SSM pide establecer un recurso (sesión de comunicación) (si no existe ya), y solicita después el establecimiento de los flujos de trenes asociados con la vinculación de trenes.

11) Se devuelven las notificaciones finales al SSM.

12) Se devuelven las notificaciones finales a los USM de los consumidores 1, 2 y 3 [o a alguno de ellos, dependiendo de su respuesta en el paso 9 (para cada consumidor i)].

13) Se devuelven las notificaciones finales a los consumidores 1, 2 y 3 (o a alguno de ellos, dependiendo de la respuesta de la UAP en el paso 9 y/o del comportamiento del USMi[34].

7 Visión general de la especificación RET

En esta cláusula se presenta la especificación, que da la funcionalidad y el alcance general del punto de referencia en esta especificación y lo define brevemente.

La cláusula 8 define cómo un consumidor accede a un detallista para utilizar servicio accesibles. Trata del establecimiento y uso de una asociación segura entre los dominios, denominada una sesión de acceso (definida y descrita más detalladamente en la cláusula 8). Dentro de la sesión de acceso, trata del control de servicios y sesiones de servicio y de la gestión de abono. Consta de un conjunto de interfaces operacionales, ofrecidas por los cometidos comerciales consumidor y detallista. Las interfaces se definen primero informalmente utilizando texto claro y diagramas, y después por medio de especificaciones IDL semiformales; el comportamiento se describe en texto claro. Se describe también una interfaz especializada de abono, así como especificaciones completas de las interfaces IDL.

La cláusula 9 contiene las especificaciones completas de interfaces IDL para Ret-RP incluida las interfaces para la parte de abono.

Este Suplemento consta de especificaciones no formales, en texto claro y diagramas, y de especificaciones semiformales utilizando el lenguaje de definición de interfaz.

La finalidad de este Suplemento es proporcionar especificaciones que están listas para ser utilizadas en la implementación de múltiples vendedores interoperables de las interfaces computacionales

---

[33] Dependiendo del tipo de servicio y de las capacidades del terminal, pueden darse varios casos: si la aplicación es la única que utiliza trenes, la parte nodal (terminal) de la vinculación de trenes puede efectuarse en el soporte físico. Pudiera ser también que, al recibir la invitación, el consumidor 1 ya conoce (o considera probable) que se le pida que participe en una vinculación de trenes, y que comience a preparar las acciones internas del terminal necesarias para obtener una interfaz de trenes (por ejemplo, pedir a otra aplicación que libere una interfaz de trenes o suprimir alguna aplicación para aumentar la capacidad de funcionamiento). Esto muestra la especialización de comportamiento de la UAP.

[34] Es posible que el USM obtenga la notificación, pero que no la envíe a la UAP; esto es similar a lo explicado en el paso 5), donde el USM toma decisiones (por ejemplo, "pantallas") en nombre del usuario/UAP.

requeridas entre los dominios descritos en este Suplemento: el dominio detallista y el dominio consumidor.

7.1 Funcionalidad general y alcance de los puntos de referencia

La definición de un punto de referencia en el SPFEE es que define las interacciones entre partes interesadas (por medio de interfaces que se proporcionan entre sí). En este Suplemento:

– El punto de referencia detallista (Ret-RP, *retailer reference point*): entre el consumidor y el detallista.

Obsérvese que el cometido usuario de extremo está generalizado en este Suplemento como cometido consumidor. El cometido parte interesada de consumidor modela dos partes interesadas: el abonado y el usuario de extremo. El abonado es la entidad que tiene una relación comercial con el detallista, mientras que el usuario de extremo es la persona que utiliza realmente las capacidades proporcionadas por el proveedor de servicio a través del detallista.

El Ret-RP soporta un consumidor que accede a un detallista para utilizar los servicios que pone a disposición a nombre de uno o más proveedores de servicios. Trata el establecimiento y uso de una asociación segura entre los dominios, denominada una sesión de acceso. Dentro de la sesión de acceso, trata del control del ciclo de vida del uso de los servicios del proveedor. Corresponde con la funcionalidad, interfaces y objetos relacionados con la sesión de acceso. Define interfaces para sustentar el uso de las siguiente funcionalidad:

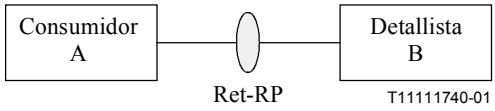
– iniciación de diálogo entre los dominios de consumidor (abonado y usuario de extremo) y de detallista;
– identificación de los dominios entre sí (cada dominio puede permanecer anónimo dependiendo de la interacción solicitada);
– establecimiento de una asociación segura entre los dominios, en una sesión de acceso;
– establecimiento del contexto por defecto para el control y gestión de la funcionalidad de utilización;
– descubrimiento de ofertas de servicio[35];
– lista de sesiones de acceso, sesiones de servicio y servicios abonados;
– iniciación de utilización entre los dominios de usuario de extremo y proveedor de servicio;
– control y gestión de sesiones (por ejemplo, terminación, suspensión, reanudación, incorporación, notificación de cambios, etc.).

En este Suplemento se utilizan los siguientes principios:

– movilidad personal y de sesión, que proporciona la descripción de cómo transferir y gestionar entornos personales entre puntos de acceso de usuario dentro de una sesión;
– gestión, que proporciona los mecanismos para gestionar la información administrativa (por ejemplo, abonados) y FCAPS (por ejemplo, gestión de averías para un servicio).

---

[35] Estos servicios pueden ser primarios [por ejemplo, vídeo a la carta (VoD, *video on demand*)], auxiliares de los primarios (por ejemplo, gestión de configuración para VoD) o administrativos (por ejemplo, gestión de abonado para VoD).



Utilización de Ret-RP con respecto a los cometidos comerciales TINA

7.1.1 Ciclo de vida del cometido comercial Ret-RP

El Ret-RP soporta todo el ciclo de vida de la relación entre consumidor y detallista, que se describe como ciclo de vida de abonado y usuario de extremo.

El ciclo de vida de abonado describe los procesos por los cuales un abonado establece una relación con un detallista, y la modifica o termina. La relación comprende abono, personalización, y la asociación entre abonado y usuario de extremo.

El ciclo de vida de usuario de extremo describe el proceso por el cual los usuarios de extremo pueden acceder a servicios y utilizarlos, lo que comprende establecimiento del sistema de usuario de extremo, contacto con el detallista y personalización del servicio.

7.2 Hipótesis principales

Dos hipótesis principales son:

– se supone un DPE con capacidad de penetración, interoperable que proporciona servicios de seguridad;

– se supone la existencia de una dirección de denominación y marco de resolución.

7.3 Definición del punto de referencia Ret

La definición del punto de referencia Ret es una especificación semiformal de la relación comercial entre el consumidor y el detallista. De conformidad con el modelo comercial y puntos de referencia del SPFEE [6], el Ret-RP se divide en una parte *acceso* y una parte *utilización*. Para el Ret-RP o parte acceso describe cómo un cometido comercial consumidor accede a un cometido comercial detallista para utilizar servicios proporcionados por proveedores de servicio; la parte utilización está fuera del ámbito de este Suplemento. Como cada parte es tratada independientemente en las especificaciones del SPFEE, pueden ser usadas también independientemente. Este Suplemento sólo describe la parte de acceso.

7.3.1 Cometidos comerciales y cometidos de sesión

Como se indica en [6], un cometido comercial puede desempeñar diferentes cometidos de sesión. Se definen dos cometidos de sesión básicos: *usuario* y *proveedor*. Se hace una especialización cuando se tratan interacciones relacionadas con acceso. De este modo, los cometidos se convierten en usuario de acceso y proveedor de acceso. Los cometidos de sesión y dominios comerciales que denominan convenciones se reflejan en la denominación de la estructura de módulos para las especificaciones IDL (cláusula 9).

Aunque las especificaciones del Ret-RP se relacionan con cometidos comerciales (de conformidad con [6]), la aplicabilidad de las especificaciones puede ampliarse a relaciones en las que participan los mismos cometidos de sesión, prescindiendo de los cometidos comerciales en cuestión. Por ejemplo, siempre que se puede definir un usuario de acceso y un proveedor de acceso, se pueden aplicar las especificaciones de la parte acceso del Ret-RP. No obstante, los medios para ampliar las especificaciones Ret-RP a contextos distintos de la relación consumidor/detallista (por ejemplo, para federación de detallista a detallista) están fuera del ámbito de este Suplemento.

7.3.2 Conformidad con las especificaciones de punto de referencia red

Este Suplemento proporciona las directrices necesarias para identificar lo que significa la conformidad con Ret-RP. La conformidad no significa el soporte de todas las características sino el soporte de todas las características obligatorias[36] de conformidad con las especificaciones del SPFEE, y la conformidad de las especificaciones del SPFEE para características facultativas, si son soportadas.

El Ret-RP se perfila como interfaces y operaciones obligatorias y facultativas.

El lado consumidor y el lado detallista alegan separadamente conformidad con Ret-RP.

Por tanto, para un sistema SPFEE, el nivel mínimo de conformidad con Ret-RP significa soportar por lo menos las interfaces y operaciones obligatorias de uno de los dos lados (consumidor o detallista).

Para que dos sistemas SPFEE interactúen por el Ret-RP, se requiere que un sistema se conforme con el lado consumidor y el otro con el lado detallista.

8 Especificación de Ret-RP

El Ret-RP ofrece las siguientes capacidades:

- iniciación de diálogo entre los dominios de consumidor y detallista;
- identificación de los dominios entre sí (cada dominio puede permanecer anónimo dependiendo de la interacción solicitada);
- establecimiento de una asociación segura entre los dominios (una sesión de acceso);
- establecimiento del contexto por defecto para el control y gestión de la funcionalidad de utilización (sesiones de servicio);
- descubrimiento de ofertas de servicio[37];
- listas de sesiones de acceso, sesiones de servicio y servicios abonados;
- iniciación de utilización entre los dominios (comienzo de una sesión de servicio);
- control y gestión de sesiones de servicio (por ejemplo, terminación, suspensión, reanudación, incorporación, notificación de cambios, etc.

Cabe señalar que el Ret-RP trata dos tipos de funcionalidad de acceso:

- funcionalidad dedicada a la sesión de acceso entre el consumidor y el detallista;
- funcionalidad relacionada con acceso a servicios, para los cuales el detallista debe invocar uno o más proveedores de servicio que soportan los servicios reales (el detallista es un puro detallista, dedicado solamente a la funcionalidad de acceso).

  Una de las funcionalidades del detallista es invocar posiblemente más de un proveedor de servicio para satisfacer una petición de consumidor. Esto se hace de manera totalmente transparente para el consumidor. El detallista hace una o más invocaciones a uno o más proveedores de servicio y devuelve un resultado fusionado. Esto es transparente también para el proveedor de servicio, que no debe saber que el detallista está posiblemente haciendo invocaciones similares a otros proveedores de servicio. Se ha de señalar que la especificación del Ret-RP y Ret-SP-RP son suficientemente flexibles y genéricas para

---

[36] En este caso, característica significa interfaz u operación.

[37] Estos servicios pueden ser primarios [por ejemplo, vídeo a la carta (VoD)], auxiliares de los primarios (por ejemplo, gestión de configuración para VoD) o administrativos (por ejemplo, gestión de abonado para VoD).

asegurar que el cometido multiplexor del detallista puede ser desempeñado. En consecuencia, se considera que una especificación más detallada de este asunto está fuera del ámbito de la especificación de los puntos de referencia Ret y Ret-SP.

Las funciones dedicadas al acceso a servicios son:

– descubrimiento de ofertas de servicios[38];
– listas de sesiones de acceso, sesiones de servicio y servicios abonados;
– iniciación de utilización entre los dominios (comienzo de una sesión de servicio);
– control y gestión de sesiones de servicio (por ejemplo, terminación, suspensión, reanudación, incorporación, notificación de cambios, etc.).

El Ret-RP trata ampliamente el establecimiento y uso de una asociación segura entre los dominios, denominada una sesión de acceso.

La sesión de acceso Ret-RP se define a continuación. Muchas de las interfaces y operaciones definidas serán aplicables a las partes de acceso de otros puntos de referencia entre dominios (tales como Ret-SP y Retailer_to_Retailer). Para facilitar esta reutilización, se ha definido un conjunto de interfaces que pueden ser reutilizadas en otros puntos de referencia. Estas interfaces pueden ser reconocidas por el prefijo `i_User` o `i_Provider`. Estos tipos de interfaces corresponden con los cometidos usuario de acceso y proveedor de acceso.

Las interfaces para el Ret-RP se designan con los prefijos `i_Consumer` e `i_Retailer`, que corresponden con los dominios administrativos comerciales de consumidor y de detallista del modelo comercial SPFEE [6]. Para Ret-RP, estos dominios desempeñan los cometidos usuario de acceso y proveedor de acceso. Todas las interfaces `i_Consumer` e `i_Retailer` son heredadas de las correspondientes interfaces `i_User` e `i_Provider`. Las especializaciones para Ret-RP se definen en las interfaces `i_Consumer`/`i_Retailer`, aunque actualmente no se definen especializaciones.

En resumen, las interfaces Ret-RP se heredan de interfaces genéricas usuario-proveedor que pueden ser reutilizadas en muchos otros puntos de referencia.

NOTA – El texto principal de este Suplemento describe solamente interfaces y operaciones en interfaces. En la cláusula 9 figura una lista completa de las definiciones IDL, y cómo los interfaces se agrupan en módulos.

El resto de la sesión de acceso del Ret-RP se ha estructurado como sigue:

La subcláusula 8.1 "Visión general de interfaces de acceso para Ret-RP" describe las interfaces de acceso de Ret-RP, junto con una breve explicación de cada operación e identifica las interfaces que son exportadas por Ret-RP.

La subcláusula 8.2 "Interfaces de usuario-proveedor" identifica las interfaces genéricas usuario-proveedor que no son exportadas por el Ret-RP. Muestra la jerarquía de herencia para las interfaces exportadas por Ret-RP y describe también las interfaces genéricas, de modo que puedan ser reutilizadas en otros puntos de referencia entre dominios.

La subcláusula 8.3 "Visión de información de acceso" da una visión de información del Ret-RP y describe los tipos de información pasados por el Ret-RP.

Las subcláusulas 8.5 "Definiciones de interfaz de acceso: Interfaces de dominio de consumidor" y 8.6 "Definiciones de interfaz de acceso: Interfaces de dominio de detallista" describen las operaciones de interfaces Ret-RP soportadas por los dominios de consumidor y de detallista, incluidos los parámetros y la dinámica.

---

[38] Estos servicios pueden ser primarios [por ejemplo, vídeo a la carta (VoD)], auxiliares de los primarios (por ejemplo, gestión de configuración para VoD) o administrativos (por ejemplo, gestión de abonado para VoD).

Las definiciones IDL de cada una de las interfaces figuran en la cláusula 9.

8.1 Visión general de interfaces de acceso para Ret-RP

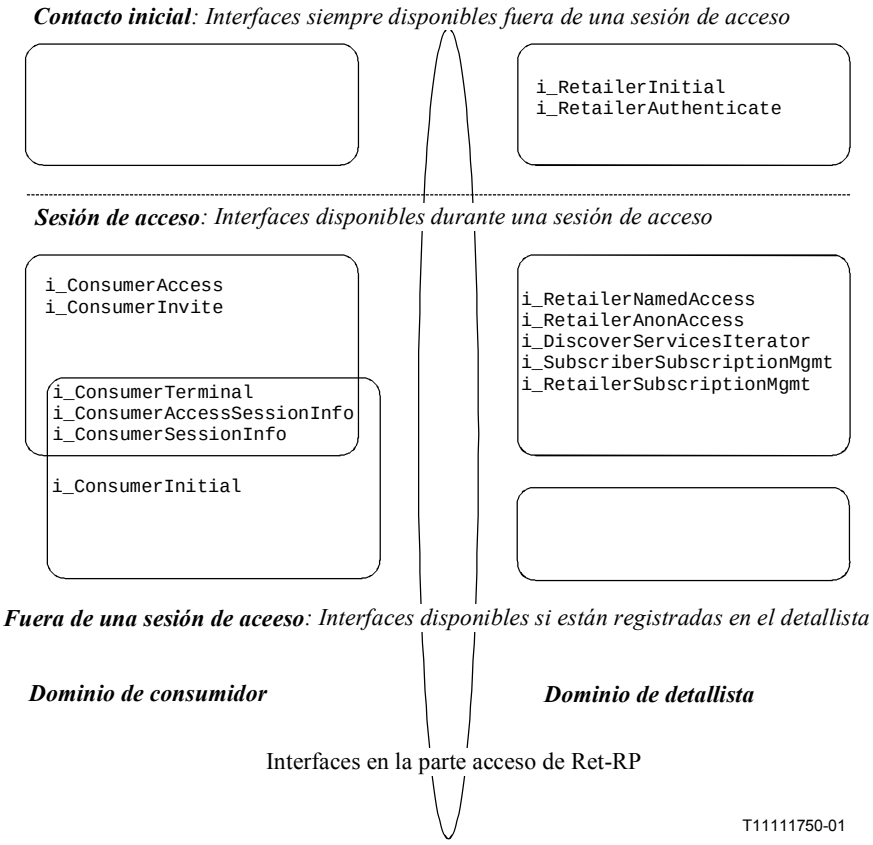
La parte acceso del Ret-RP se define por un conjunto de interfaces que son ofrecidas en el punto de referencia. Todas las interfaces en la parte acceso están clasificadas según el lado del RP que ofrece la interfaz: el consumidor o el detallista.

Las interfaces son clasificadas también según sean accesibles durante una sesión de acceso o estén siempre disponibles fuera de una sesión de acceso, o puedan estar registradas para estas disponibles fuera de una sesión de acceso.

El registro de interfaces sólo puede ser efectuado por el consumidor en el dominio detallista durante una sesión de acceso. La duración del registro depende de cómo el consumidor registra sus interfaces, es decir, sólo mientras exista la sesión de acceso o permanente.

El siguiente diagrama denomina todas las interfaces definidas por la parte acceso y las clasifica como se indica anteriormente.

Las interfaces "siempre disponibles fuera de una sesión de acceso" son soportadas por el detallista para permitir que un consumidor pida el establecimiento de una sesión de acceso. Son el punto de contacto inicial para el consumidor y le permiten autenticarse y autenticar al detallista, establecer la sesión de acceso y obtener una referencia a una interfaz `i_RetailerNamedAccess`, o `i_RetailerAnonAccess`.



Las interfaces "disponibles durante una sesión de acceso" permiten que el consumidor y el detallista interactúen durante una sesión de acceso. Las interfaces en el detallista permiten que el consumidor descubra servicios, inicie el uso de estos servicios, controle y gestione los servicios (por ejemplo, terminación, suspensión, reanudación, etc.) y registre el contexto e interfaces del consumidor con el detallista. Las interfaces soportadas por el consumidor permiten que el detallista descubra interfaces y configuración del terminal del consumidor, notifique cambios en las sesiones de acceso y de

servicio y envíe invitaciones de incorporación a sesiones de servicio. Estas capacidades sólo son posibles durante la sesión de acceso.

Las interfaces "disponibles si están registradas en el detallista" son soportadas por el consumidor. Deben estar registradas en el detallista para que sean accesibles para uso fuera de la sesión de acceso. Cuando la interfaz apropiada está registrada, el detallista puede ejecutar todas las operaciones "disponibles" durante una sesión de acceso, así como pedir al consumidor que inicie una sesión de acceso con ellas.

A continuación se describirán globalmente las interfaces y sus operaciones. En 8.5 y 8.6 figura información detallada sobre operaciones, sus listas de parámetros y su dinámica.

Las interfaces dedicadas al abono (i_SubscriberSubscriptionMgmt e i_RetailerSubscriptionMgmt) se describen en 8.7 "Gestión de abono".

8.1.1 Ejemplo de escenario de parte acceso de Ret-RP

En esta subcláusula se muestra un ejemplo del uso de las interfaces Ret-RP de acceso. Describe un consumidor que utiliza interfaces de detallista para establecer una sesión de acceso, utiliza facilidades de detallista y se registra para recibir invitaciones fuera de una sesión de acceso.

1) Un dominio de consumidor se pone en contacto con el detallista obteniendo una referencia a una interfaz i_RetailerInitial[39].

2) El dominio de consumidor invoca la operación `requestNamedAccess`() en `i_RetailerInitial`, porque desea establecer una sesión de acceso con el detallista como un usuario denominado. (En cambio, si el consumidor desea permanecer anónimo, podrá utilizar la operación `requestAnonymousAccess`() en esa interfaz.)

   2a) Si se han utilizado servicios de seguridad CORBA, se habrán intercambiado las credenciales de ambos dominios y otra información de autenticación, y el consumidor y el detallista habrán sido autenticados entre sí. La invocación a `requestNamedAccess`() devuelve una referencia a una interfaz `i_RetailerNamedAccess`. (Se ha establecido una sesión de acceso entre el consumidor y el detallista.)

   2b) Si no se utilizan servicios de seguridad CORBA, la invocación a `requestNamedAccess`() falla y se plantea una excepción e_AuthenticationError. Esta excepción contiene una referencia a una interfaz `i_RetailerAuthenticate`, que el consumidor puede utilizar para autenticarse. Después de esto, el consumidor invoca `requestNamedAccess`() en `i_RetailerInitial` para obtener una referencia a la interfaz `i_RetailerNamedAccess`.

3) En este punto, se ha establecido una sesión de acceso, y el consumidor tiene una referencia a la interfaz `i_RetailerNamedAccess`.

4) El dominio de consumidor informa al dominio de detallista sus interfaces y configuración de terminar invocando la operación **setUserCtxt**() en `i_RetailerNamedAccess`. El detallista obtiene referencias de las interfaces `i_ConsumerAccess`, `i_ConsumerInvite`, `i_ConsumerTerminal`, e `i_ConsumerSessionInfo` para uso dentro de esta sesión de acceso.

5) Ahora el consumidor puede, invocando la operación apropiada en la interfaz `i_RetailerNamedAccess`:

   – descubrir servicios ofrecidos por el detallista (`discoverServices`());

   – abonarse a estos servicios (comenzando un servicio de abono);

   – enumerar las sesiones de acceso y sesiones de servicio que existen en ese momento con (`listAccessSessions`(), `listServiceSessions`())

---

[39] El mecanismo por el cual el consumidor obtiene esa interfaz no es indicado por Ret-RP.

– comenzar una nueva sesión de servicio (`startService`());
– suspender, reanudar, incorporarse a y terminar sesiones existentes (`suspendSession`(), `resumeSession`(), `endSession`());
– obtener referencias a interfaces específicas de detallista (`getInterfaces`());
– registrar interfaces para uso fuera de una sesión de acceso (`registerInterfaceOutsideAccessSession`());
– y mucho más ...

6) El detallista puede:
– obtener referencias a interfaces específicas de detallista (mediante la interfaz `i_ConsumerAccess`);
– invitar al consumidor a incorporarse a una sesión (mediante la interfaz `i_ConsumerInvite`);
– descubrir la configuración de terminal (mediante la interfaz `i_ConsumerTerminal`);
– informar al consumidor los cambios de sus sesiones de acceso y de servicio (mediante las interfaces `i_UserAccessSessionInfo`, e `i_UserSessionInfo`);

7) El consumidor registra la interfaz `i_ConsumerInitial` para uso fuera de una sesión de acceso (mediante `registerInterfaceOutsideAccessSession`() en `i_RetailerNamedAccess`). Después termina la sesión de acceso (), y no puede hacer más peticiones al detallista.

8) El detallista puede invitar al consumidor a incorporarse a una sesión de acceso utilizando `inviteUserOutsideAccessSession`() en la interfaz `i_ConsumerInitial`.

9) Si el consumidor desea incorporarse a la sesión a la que ha sido invitado, tendría que establecer otra sesión de acceso (como en el paso 1).

8.1.2 Siempre disponible fuera de una sesión de acceso

Sólo las interfaces de detallista están siempre disponibles fuera de una sesión de acceso.

Las siguientes interfaces son proporcionadas por el detallista para que el consumidor y/o el detallista puedan autenticarse y establecer una sesión de acceso.

- `i_RetailerInitial` – Esta interfaz es el punto de contacto inicial del consumidor con el detallista. Se puede utilizar para pedir el establecimiento de una sesión de acceso. La sesión de acceso proporciona un acceso de consumidor para utilizar sus servicios abonados, etc., a través de una interfaz i_RetailerNamedAccess, o i_RetailerAnonAccess, si el consumidor está autenticado como un usuario denominado o anónimo. Si el consumidor no está autenticado, devuelve una referencia a la interfaz i_RetailerAuthenticate, para poder efectuar esta autenticación.
- **i_RetailerAuthenticate** – Esta interfaz es utilizada por el consumidor para autenticarse a sí mismo y al detallista y para pasar credenciales que pueden ser utilizadas para establecer la sesión de acceso.

8.1.2.1 Interfaz i_RetailerInitial

La interfaz i_RetailerInitial permite que el consumidor pida el establecimiento de una sesión de acceso.

– `requestNamedAccess`() permite al consumidor identificarse ante el detallista, y establecer una sesión de acceso. Puede haber sido establecido ya un contexto seguro entre el consumidor y el detallista utilizando los servicios de seguridad CORBA. En este caso, esta operación devuelve una referencia a una interfaz i_RetailerNamedAccess. Si el consumidor no ha sido autenticado aún, se planteará una excepción e_AuthenticationError. Esto contiene

una referencia a una interfaz i_RetailerAuthenticate que puede ser utilizada para autenticar y establecer el contexto seguro. Esta operación puede ser invocada de nuevo para extraer la referencia a la interfaz i_RetailerNamedAccess.

– **`requestAnonymousAccess`**() permite al consumidor establecer una sesión de acceso con el detallista a fin de revelar de su identidad. La sesión de acceso permitirá el acceso a algunos servicios, aunque es posible que el consumidor tenga que negociar con el detallista los servicios que están disponibles. (Evidentemente, los servicios no son especializados para el consumidor.) El consumidor interactúa con el detallista a través de una interfaz i_RetailerAnonAccess. En los demás casos, esta operación es igual a requestNamedAccess().

8.1.2.2 Interfaz i_RetailerAuthenticate

La interfaz i_RetailerAuthenticate permite que el consumidor y/o el detallista se autentiquen y adquieran credenciales para establecer un contexto seguro. La interfaz proporciona un mecanismo genérico de autenticación que puede ser utilizado para apoyar un número de diferentes protocolos de autenticación.

La finalidad primaria de esa interfaz es verificar que el consumidor y el detallista están hablando realmente con el dominio que deseaba. No se pretende necesariamente identificar al consumidor. (requestNamedAccess() se utiliza para identificar al consumidor y proporcionarle acceso a sus servicios.)

– `getAuthenticationMethods`() proporciona una lista de los métodos de autenticación soportados por el detallista.

– **`authenticate`**() permite que el consumidor seleccione un método de autenticación, pase datos de autenticación y pida credenciales específicas que puedan ser utilizadas para mantener un contexto seguro. El detallista devuelve sus datos de autenticación (envía datos para que el consumidor responda utilizando continueAuthentication() (si es necesario) y las credenciales solicitadas (si es posible). Si se requiere otro protocolo de autenticación antes que las credenciales sean devueltas, éstas pueden ser devueltas por continueAuthentication()

– **`continueAuthentication`**() puede ser invocada una o más veces después de authenticate(). Permite que el consumidor responda a los datos devueltos de authenticate() o de la invocación anterior continueAuthentication(). En la primera de las siguientes invocaciones de continueAuthentication(), las credenciales solicitadas por el consumidor pueden ser devueltas de acuerdo con los requisitos del protocolo.

8.1.2.3 Disponible durante una sesión de acceso

Ambas interfaces de consumidor y de detallista están disponibles durante una sesión de acceso.

El consumidor soporta las siguientes interfaces para que el detallista las utilice durante la sesión de acceso:

– `i_ConsumerAccess` – El detallista puede encontrar las interfaces en el dominio de consumidor utilizando esta interfaz. Proporciona al detallista las referencias de interfaz a otras interfaces en el dominio de consumidor.

– `i_ConsumerInvite` – Esta interfaz es utilizada por el detallista para notificar al consumidor invitaciones de incorporación a sesiones de servicio. El consumidor puede registrar una interfaz i_ConsumerInitial para recibir invitaciones fuera de una sesión de acceso.

– `i_ConsumerTerminal` – Esta interfaz es utilizada por el detallista dentro de una sesión de acceso para acceder a información de configuración de terminal, por ejemplo, aplicaciones instaladas, configuración de soporte físico, (NAP), etc.

– `i_ConsumerAccessSessionInfo` – Esta interfaz es utilizada por el detallista para informar al consumidor los cambios de estado de otras sesiones de acceso que este consumidor tiene con este detallista.

– `i_ConsumerSessionInfo` – Esta interfaz es utilizada por el detallista para informar al consumidor los cambios de estado de sesiones de servicio que este consumidor tiene con este detallista. Se envía información sobre todas las sesiones de servicio utilizadas a través de todas las sesiones de acceso con este detallista.

Todas las interfaces del consumidor soportadas pueden estar registradas en el detallista para uso fuera o dentro de una sesión de acceso, mediante las operaciones en la interfaz i_RetailerNamedAccess. Es posible registrar también otras interfaces específicas de detallista no definidas por Ret-RP. El registro de las tres primeras mencionadas es obligatorio, utilizando la operación setUserCtxt() en la interfaz i_RetailerNamedAccess. La duración de este registro particular es igual que la duración de la sesión de acceso.

El detallista soporta dos interfaces para uso durante sesiones de acceso. Al consumidor sólo se le dará una referencia de una de estas interfaces. Si han sido autenticadas con un nombre de usuario y se ha invocado la operación requestNamedAccess(), se le dará una referencia a i_RetailerNamedAccess; en los demás casos, si ha sido autenticado como un usuario anónimo, y ha invocado requestAnonymousAccess(), se le dará una referencia a i_RetailerAnonAccess:

– `i_RetailerNamedAccess` – Esta interfaz permite que un consumidor conocido acceda a sus servicios abonados, comience y gestione sesiones de servicio, etc.

– `i_RetailerAnonAccess` – Esta interfaz es utilizada por el detallista para notificar al consumidor las invitaciones de incorporación a sesiones de servicio. El consumidor puede registrar una interfaz i_ConsumerInitial para recibir invitaciones fuera de una sesión de acceso.

Durante una sesión de acceso, el consumidor tendrá acceso a una de estas interfaces, dependiendo de si ha sido autenticado como un usuario denominado o anónimo. La definición actual de Ret-RP no permite cambiar de usuario anónimo a denominado en la misma sesión de acceso.

El detallista soporta también la siguiente interfaz:

• `i_DiscoverServicesIterator` – Se devuelve al consumidor una referencia a esta interfaz después de invocar la operación discoverServices() en cualquiera de las interfaces mencionadas anteriormente. Se utiliza para extraer las descripciones de servicios restantes, que no fueron devueltas directamente de discoverServices().

8.1.2.4 Interfaz i_ConsumerAccess

La interfaz i_ConsumerAccess permite al detallista acceder al dominio de consumidor durante una sesión de acceso y al detallista pedir referencias a interfaces soportadas por el dominio de consumidor. Estas interfaces incluyen las definidas por Ret-RP así como otras interfaces específicas de detallista.

– `cancelAccessSession`() – Permite que el detallista cancele esta sesión de acceso. Después que se ha invocado esta operación, ni el consumidor ni el detallista utilizarán las otras interfaces. (Es posible utilizar aún las interfaces registradas para uso fuera de la sesión de acceso o las interfaces dentro de otra sesión de acceso.)

Esta interfaz hereda las siguientes operaciones de la interfaz i_UserAccess, para que el detallista obtenga referencias a otras interfaces soportadas por el consumidor:

– `getInterfaceTypes`() – Permite al detallista descubrir todos los tipos de interfaces sustentados por el dominio de consumidor.

– **getInterface**() – Permite al detallista extraer una referencia de interfaz, dando el tipo de interfaz y las propiedades.

– **getInterfaces**() – Permite al detallista extraer una lista de todas las interfaces, soportadas por el consumidor.

Esta interfaz se registra en el detallista utilizando la operación setUserCtxt(), y está disponible durante la sesión de acceso vigente.

8.1.2.5 Interfaz i_ConsumerInvite

La interfaz i_ConsumerInvite permite que el detallista envíe invitaciones de incorporación a la sesión de servicio, durante una sesión de acceso. Sólo está disponible durante una sesión de acceso para recibir invitaciones. Si el consumidor desea recibir invitaciones fuera de una sesión de acceso, debe registrar la interfaz i_ConsumerInitial para uso fuera de una sesión de acceso.

– `inviteUser`() – Permite al detallista invitar al consumidor a incorporarse a una sesión de servicio. En la lista de parámetros se dispone de una descripción de sesión e información insuficiente para incorporarse a la sesión. Sólo es posible incorporarse a la sesión utilizando la operación joinSessionWithInvitation() en la interfaz i_RetailerNamedAccess.

– `cancelInviteUser`() – Permite que el detallista informe al consumidor que se ha cancelado una invitación enviada previamente al consumidor.

Esta interfaz se registra en el detallista utilizando la operación setUserCtxt() y está disponible para uso durante la sesión de acceso vigente.

8.1.2.6 Interfaz i_ConsumerTerminal

La interfaz i_ConsumerTerminal permite al detallista obtener información sobre la configuración de terminal y aplicaciones del dominio de consumidor.

– `getTerminalInfo`() – Permite al detallista obtener información sobre el terminal del dominio de consumidor. Se puede obtener información sobre el identificador y tipo de terminal, los puntos de acceso de red y las aplicaciones de usuario..

Esta interfaz se registra en el detallista utilizando la operación setUserCtxt(), y está disponible para uso durante la sesión de acceso vigente.

8.1.2.7 Interfaz i_ConsumerAccessSessionInfo

La interfaz i_ConsumerAccessSessionInfo permite que el detallista informe al consumidor los cambios de estado en otras sesiones de acceso con el consumidor (por ejemplo, sesiones de acceso con el mismo consumidor que son creadas o suprimidas). Sólo se informa al consumidor sobre las sesiones de acceso en las cuales participa.

– `newAccessSessionInfo`() – Esta operación (unidireccional) es utilizada por el detallista para informar al consumidor sobre una nueva sesión de acceso en la cual el consumidor participa.

– `endAccessSessionInfo`() – Esta operación (unidireccional) es utilizada por el detallista para informar al consumidor que otra sesión de acceso ha terminado.

– `cancelAccessSessionInfo`() – Esta operación (unidireccional) es utilizada para informar al consumidor que el detallista ha cancelado una sesión de acceso.

– `newSubscribedServicesInfo`() – Esta operación (unidireccional) es utilizada por el detallista para informar al consumidor que ha sido abonado a nuevos servicios.

Esta interfaz no se registra en el detallista utilizando la operación setUserCtxt(), sino que el dominio de consumidor debe registrar esta interfaz utilizando i_RetailerNamedAccess. Puede ser registrada dentro y fuera de una sesión de acceso.

8.1.2.8 Interfaz i_ConsumerSessionInfo

La interfaz i_ConsumerSessionInfo permite al detallista informar al consumidor los cambios de estado en las sesiones de servicios en las que el consumidor participa. Se invocan operaciones de información siempre que un cambio del servicio afecta al consumidor (es decir, la sesión es suspendida), pero no cuando el cambio no afecta al consumidor (es decir, otra parte en la sesión abandona). Esta interfaz es informada de los cambios en todas las sesiones de servicio en que participa el consumidor, y no sólo de las asociadas con esta sesión de servicio.

Las siguientes operaciones informan al consumidor que:

– `newSessionInfo`() – ha comenzado una nueva sesión de servicio;

– `endSessionInfo`() – ha terminado una sesión de servicio existente;

– `endMyParticipationInfo`() – ha terminado la participación del consumidor en la sesión;

– `suspendSessionInfo`() – ha sido suspendida una sesión de servicio existente;

– `suspendMyParticipationInfo`() – ha sido suspendida la participación del consumidor en la sesión de servicio;

– `resumeSessionInfo`() – se ha reanudado una sesión suspendida;

– `resumeMyParticipationInfo`() – se ha reanudado la participación del consumidor en la sesión;

– `joinSessionInfo`() – el consumidor se ha incorporado a una sesión de servicio.

Esta interfaz puede ser registrada en el detallista utilizando la operación setUserCtxt(). En este caso, la interfaz está disponible solamente durante la sesión de acceso vigente.

Puede ser registrada en cualquier otro momento en el detallista utilizando las operaciones de registro de interfaz en la interfaz i_RetailerAccess. Puede ser registrada para uso fuera y dentro de una sesión de acceso.

8.1.2.9 Interfaz i_RetailerNamedAccess

La interfaz i_RetailerNamedAccess permite que un consumidor conocido acceda a sus servicios abonados. El consumidor la utiliza para todas las operaciones dentro de una sesión de acceso con el detallista. Se devuelve una referencia a esta interfaz cuando el consumidor ha sido autenticado por el detallista y se ha establecido una sesión de acceso. Es devuelta invocando requestNamedAccess() en la interfaz i_RetailerInitial.

Proporciona las siguientes operaciones (que son heredadas de la interfaz i_ProviderNamedAccess):

– **setUserCtxt**() – permite al consumidor informar al detallista sobre interfaces en el dominio de consumidor y otra información de dicho dominio (por ejemplo, aplicaciones de usuario disponibles en el dominio de consumidor, sistema operativo utilizado, etc.). Debe ser invocada inmediatamente después de recibir la referencia a esta interfaz, o las operaciones subsiguientes pueden plantear una excepción.

– **listAccessSessions**() – permite al consumidor en esta sesión de acceso hallar otras sesiones de acceso que tiene con este detallista (por ejemplo, el consumidor está en el trabajo, pero en su casa tiene una sesión de acceso establecida que ejecuta una sesión de servicio de seguridad activa).

– **endAccessSession**() – permite que el consumidor termine una sesión de acceso especificada, sea la vigente u otra, encontrada utilizando listAccessSessions(). El consumidor puede especificar también algunas acciones si hay sesiones de servicio activas.

– **getUserInfo**() – obtiene el nombre de usuario del consumidor, y otras propiedades.

– **listSubscribedServices**() – enumera los servicios a los cuales está abonado el consumidor. El alcance de los servicios abonados se puede obtener utilizando listas de propiedades. La

operación devuelve información suficiente para que el consumidor comience un servicio determinado (abonado).

- **discoverServices**() – enumera todos los servicios disponibles del detallista. El consumidor puede conocer la lista suministrando algunas propiedades que el servicio debe tener y un número máximo que ha de devolver. Se puede utilizar una referencia a una interfaz i_DiscoverServicesIterator para extraer los servicios restantes.
- **getServiceInfo**() – devuelve la información de servicio para un servicio determinado (identificado en la invocación por su ID de servicio). Se puede obtener información similar (t_ServiceProperties) con listSubscribedServices o discoverServices, pero getServiceInfo es una versión simplificada, dirigida a un solo servicio, e independiente del estado del abono.
- **listRequiredServiceComponents**() – extrae información sobre cómo telecargar el soporte lógico de la aplicación en caso de Java applets. terminalInfo se incluye como un parámetro de red inteligente para evitar una invocación explícita de la operación getTerminalInfo. Por ejemplo, en el caso de telecarga de Java applet la lista de propiedades contendrá una entrada con un par nombre-valor que describe el URL de Java applet; el nombre será URL y el valor de cadena del URL.
- **listServiceSessions**() – enumera las sesiones de servicio del consumidor. La petición puede ser considerada por la sesión de acceso y las propiedades de sesión (por ejemplo, activa, suspendida, tipo de servicio, etc.).
- **getSession** (Models/InterfaceTypes/Interface/Interfaces)() – extrae toda la información sobre una determinada sesión.
- **listSessionInvitations**() – enumera las invitaciones de incorporación a una sesión de servicio que han sido enviadas al consumidor.
- **listSessionAnnouncements**() – enumera las sesiones de servicio que han sido anunciadas. Pueden ser indicadas por algunas propiedades de anuncio.
- **startService**() – permite al consumidor comenzar una nueva sesión.
- **endSession**() – permite al consumidor terminar una nueva sesión.
- **endMyParticipation**() – permite al consumidor terminar su participación en una sesión de servicio.
- **suspendSession**() – permite al consumidor suspender una sesión de servicio.
- **suspendMyParticipation**() – permite al consumidor suspender su participación en una sesión de servicio.
- **resumeSession**() – permite al consumidor reanudar una sesión de servicio.
- **resumeMyParticipation**() – permite al consumidor reanudar su participación en una sesión de servicio.
- **joinSessionWithInvitation**() – permite al consumidor incorporarse a una sesión de servicio a la cual ha sido invitado.
- **joinSessionWithAnnouncement**() – permite al consumidor incorporarse a una sesión de servicio que ha sido anunciada.
- **replyToInvitation**() – permite al consumidor responder a una invitación. Se puede utilizar para informar la sesión de servicio a la cual ha sido invitado, si se incorporará o no a la sesión, o para enviar la invitación a otro lugar. (No permite al consumidor incorporarse a la sesión.)

Soporta también las siguientes operaciones heredadas de la interfaz i_ProviderAccessInterfaces, que son útiles para acceder a interfaces específicas de detallista:

- **getInterfaceTypes**() – permite al consumidor descubrir todos los tipos de interfaz soportados por el dominio de detallista.

– **getInterface**() – permite al consumidor extraer una referencia de interfaz, dando el tipo de interfaz y propiedades.
– **getInterfaces**() – permite al consumidor extraer una lista de todas las interfaces soportadas por el detallista.
– **registerInterface**() – permite registrar una interfaz de consumidor para uso dentro de la sesión de acceso vigente. El registro termina cuando la sesión de acceso termina, o cuando se invoca la operación unregisterInterface(). Se devuelve un índice de interfaz para que la interfaz pueda ser desregistrada.
– **registerInterfaceOutsideAccessSession**() – permite a un consumidor registrar una interfaz para uso fuera de una sesión de acceso. (La interfaz registrada debe estar aún disponible cuando no exista la sesión de acceso entre el consumidor y el detallista.)
– **listRegisteredInterfaces**() – permite al consumidor enumerar las interfaces que han sido registradas en el detallista. La lista define las interfaces registradas para uso dentro y fuera de una sesión de acceso.
– **unregisterInterface**() – permite al consumidor desregistrar una interfaz, de modo que el detallista no intente utilizar esta interfaz (dentro o fuera de la sesión de acceso).

8.1.2.10 Interfaz i_RetailerAnonAccess

La interfaz i_RetailerAnonAccess permite a un consumidor anónimo acceder a los servicios del detallista. El consumidor anónimo la utiliza para todas las operaciones dentro de una sesión de acceso con el detallista. Esta interfaz es devuelta cuando el consumidor invoca la operación requestAnonymousAccess() en la interfaz i_RetailerInitial.

Actualmente las operaciones para esta interfaz no están definidas. Soportará operaciones similares a las de la interfaz i_RetailerNamedAccess.

8.1.2.11 Interfaz i_DiscoverServicesIterator

La interfaz i_DiscoverServicesIterator es devuelta por invocaciones a la operación discoverServices(). Esta operación se utiliza para extraer una lista de servicios soportados por el detallista que concuerdan con un conjunto de propiedades. La lista generada por esta operación puede ser demasiado larga para ser devuelta como un parámetro. Esta interfaz permite extraer la lista en partes digeribles por el consumidor. Cada invocación a discoverServices() devuelve un nuevo ejemplar de esta interfaz.

– **maxLeft**() – el consumidor puede hallar cómo se dejan servicios no previstos.
– **nextN**() – el consumidor puede indicar que desea obtener información sobre los siguientes n servicios.
– **destroy**() – el consumidor informa al detallista que la interfaz ya no es necesaria.

8.1.3 Disponible fuera de una sesión de acceso si está registrada

El consumidor puede registrar su interfaz para uso por el detallista fuera de la sesión de acceso vigente. Una interfaz puede ser registrada utilizando la operación registerInterfaceOutsideAccessSession() en la interfaz i_RetailerNamedAccess. Si está registrada, el detallista retendrá una referencia a la interfaz cuando el consumidor/detallista termina la sesión de acceso vigente. El detallista puede invocar operaciones en esta interfaz sin una sesión de acceso presente.

El detallista no utilizará la interfaz registrada hasta que haya terminado la sesión de acceso en la cual fue registrada. Continuará utilizando la interfaz hasta que la interfaz sea desregistrada. Si se establece otra sesión de acceso, el detallista invocará operaciones en la interfaz registrada, además de nuevas interfaces proporcionadas como parte de la nueva sesión de acceso.

– i_ConsumerInitial. Esta interfaz permite al detallista iniciar una sesión de acceso con el consumidor. Permite también al detallista enviar invitaciones al consumidor fuera de una sesión de acceso.
– i_ConsumerTerminal. El detallista utilizará esta interfaz para acceder a información de configuración de terminal, si es necesario. Véase la descripción anterior.
– i_ConsumerAccessSessionInfo. El detallista utilizará esta interfaz para informar al consumidor los cambios de cualquiera de sus sesiones de acceso. Véase la descripción anterior.
– i_ConsumerSessionInfo. El detallista utilizará esta interfaz para informar al consumidor los cambios de estado de cualquiera de sus sesiones de acceso. Véase la descripción anterior.

8.1.3.1 Interfaz i_ConsumerInitial

La interfaz i_ConsumerInitial permite al detallista ponerse en contacto con el consumidor fuera de una sesión de acceso. Se puede utilizar para pedir al consumidor que establezca una sesión de acceso con el detallista, e invitar a un usuario a incorporarse a una sesión de servicio.

Esta interfaz sólo está disponible para el detallista si el consumidor la ha registrado durante una sesión de acceso (registerInterfaceUntilUnregistered() en la interfaz i_RetailerNamedAccess). NO está disponible a través de un corredor, como la interfaz i_RetailerInitial.

Están disponibles las siguientes operaciones:

– `requestAccess`() – permite al detallista pedir al consumidor que establezca una sesión de acceso.
– `inviteUserWithoutAccessSession`() – permite al detallista enviar una invitación al consumidor mientras no participa en una sesión de acceso con el detallista.
– `cancelInviteUserWithoutAccessSession`() – permite al detallista cancelar una invitación enviada al consumidor.

8.2 Interfaces de usuario-proveedor

Las interfaces definidas anteriormente son para uso en la Ret-RP. Los nombres de interfaz utilizados incluyen los nombres consumidor y detallista para identificar que son para uso en el Ret-RP. Las descripciones anteriores incluyen también todas las operaciones utilizadas en el Ret-RP.

Otros puntos de referencia desearán utilizar interfaces similares a las definidas anteriormente para actividades relacionadas con acceso (por ejemplo, establecer una sesión de acceso, comenzar servicios, etc.). Para permitir que otros puntos de referencia reutilicen interfaces y operaciones, se define un conjunto de interfaces de acceso genéricas que soportan cometidos de sesión de acceso definidos en 5.4 del presente Suplemento. Los cometidos soportados son usuario de acceso y proveedor de acceso. Estas interfaces pueden ser reconocidas por el prefijo i_User o i_Provider.

Las interfaces para uso en el Ret-RP han sido definidas anteriormente, incluidas todas las de operaciones heredadas. Todas las interfaces i_Consumer e i_Retailer son heredadas de las correspondientes interfaces i_User e i_Provider. Cualesquiera especificaciones para el Ret-RP se definen en las interfaces i_Consumer/i_Retailer. Sin embargo, actualmente no se definen especializaciones.

Las siguientes figuras definen la jerarquía de herencia para las interfaces Ret-RP y las interfaces genéricas usuario-proveedor.

8.2.1 Interfaces de usuario

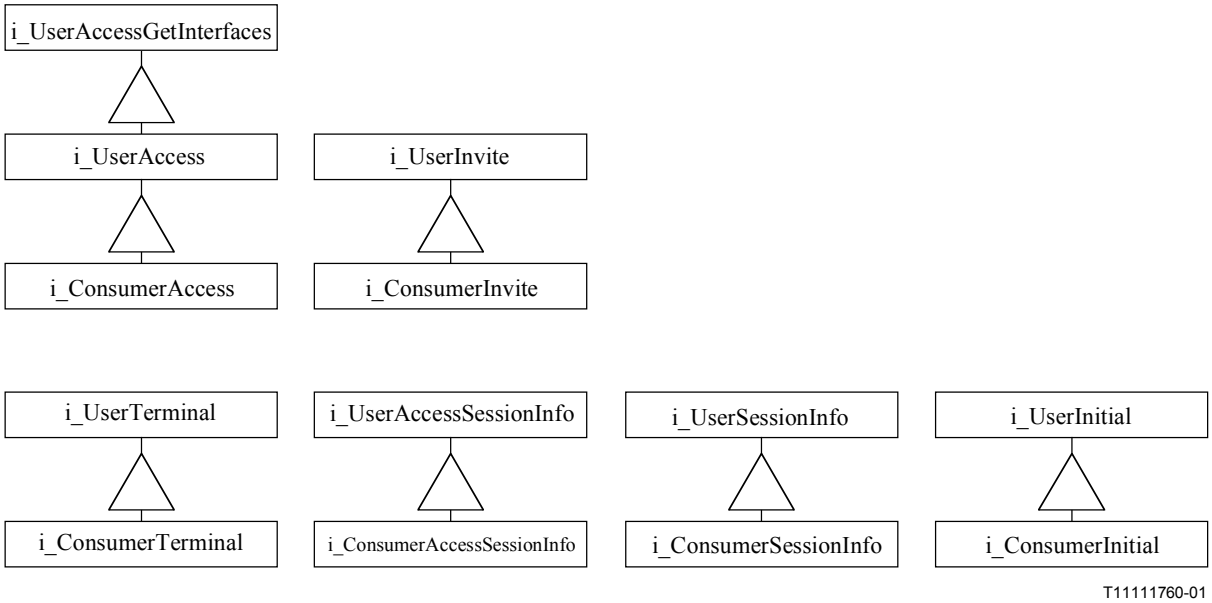


Figura 8-1 – Interfaces de consumidor heredadas de interfaces de usuario

La figura 8-1 muestra las interfaces de consumidor y las interfaces de usuario de las que hereda. Las interfaces de usuario y de consumidor tienen una correspondencia simple (todas las interfaces de consumidor heredan de una interfaz de usuario denominada correspondientemente). Todas las interfaces de usuario definen las operaciones descritas para las interfaces de consumidor en 8-1. La única excepción es la interfaz i_UserAccess que hereda esta operación de i_UserAccessGetInterfaces. Esto es para que otras interfaces puedan reutilizar las operaciones con miras a extraer interfaces.

8.2.1.1 i_UserAccess

Esta interfaz hereda de la interfaz abstracta i_UserAccessGetInterfaces, y define la siguiente operación:

– **cancelAccessSession**()

8.2.1.2 i_UserInvite

Esta interfaz define la siguiente operación:

– inviteUser()

– cancelInviteUser()

8.2.1.3 i_UserTerminal

Esta interfaz define la siguiente operación:

– getTerminalInfo()

8.2.1.4 i_UserAccessSessionInfo

Esta interfaz define las siguientes operaciones:

– newAccessSessionInfo()

– endAccessSessionInfo()

– cancelAccessSessionInfo()
– newSubscribedServicesInfo()

8.2.1.5 i_UserSessionInfo

Esta interfaz define las siguientes operaciones:

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

8.2.1.6 i_UserInitial

Esta interfaz define las siguientes operaciones:

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

8.2.2 Interfaces de proveedor

La figura 8-2 muestra las interfaces de detallista y las interfaces de proveedor de las que heredan.

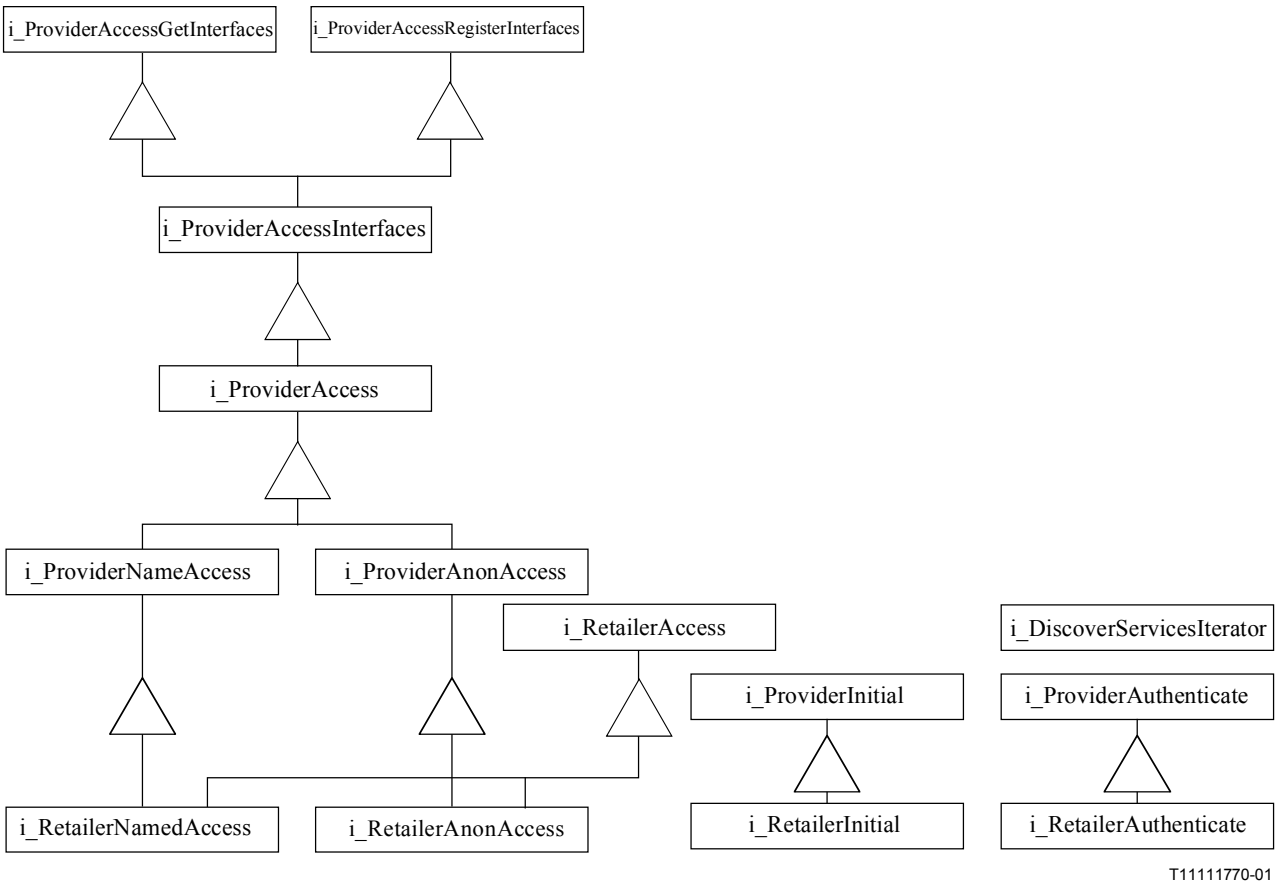


Figura 8-2 – Interfaces de detallista heredadas de interfaces de proveedor

La jerarquía de herencia para las interfaces i_RetailerNamedAccess e i_RetailerAnonAccess es compleja. Ambas heredan de i_RetailerAccess. Esta interfaz define operaciones específicas de Ret-RP que son comunes a ambas interfaces. (Actualmente no se definen operaciones.)

i_RetailerNamedAccess hereda también de i_ProviderNamedAccess, que definen las operaciones disponibles para el cometido proveedor de acceso genérico, cuando el dominio de usuario soporta un usuario conocido. i_ProviderNamedAccess define todas las operaciones ofrecidas por i_RetailerNamedAccess.

i_RetailerAnonAccess hereda de i_ProviderAnonAccess, que define las operaciones disponibles para el cometido proveedor de acceso genérico, cuando el dominio de usuario soporta un usuario anónimo. Actualmente i_ProviderAnonAccess sólo hereda operaciones de i_ProviderAccess.

La interfaz i_ProviderAccess define el cometido de proveedor de acceso genérico para reutilización en otros puntos de referencia. Es heredada por i_ProviderNamedAccess e i_ProviderAnonAccess. Actualmente, no se definen operaciones para esta interfaz. En el futuro, algunas de las operaciones definidas para i_ProviderNamedAccess se trasladarán aquí, porque son comunes a ambas interfaces, siendo apropiadas para usuarios conocidos y anónimos.

8.2.2.1 i_ProviderNamedAccess

Esta interfaz define las siguientes operaciones y hereda otras de 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 i_ProviderAnonAccess

Esta interfaz no define operaciones. Hereda de i_ProviderAccess.

8.2.2.3 i_ProviderAccess

Esta interfaz no define operaciones. Hereda de i_ProviderAccessInterfaces.

8.2.3 Interfaces abstractas

Esta subcláusula describe las interfaces abstractas heredadas en varias interfaces de detallista y de consumidor. No son exportadas por Ret. La finalidad principal de estas interfaces es proporcionar un mecanismo genérico de registro y extracción de interfaces en un determinado dominio.

– `i_UserAccessGetInterfaces` – permite al proveedor extraer todas las interfaces, sólo las interfaces que tienen ciertas propiedades o tipos de interfaz de la sesión de acceso vigente.
– `i_ProviderAccessGetInterfaces` – permite al usuario extraer todas las interfaces, sólo las interfaces que tienen ciertas propiedades o tipos de interfaz de la sesión de acceso vigente.
– `i_ProviderAccessRegisterInterfaces` – permite al usuario registrar todas las interfaces para la duración de una sesión de acceso o permanente. Ofrece también una operación para desregistrar interfaces.
– `i_ProviderAccessInterfaces` – hereda las dos anteriores y no ofrece funcionalidad adicional.

8.2.3.1 i_UserAccessGetInterfaces

Esta interfaz define las siguientes operaciones:

– `getInterfaceTypes()`
– **getInterface**()
– **getInterfaces**()

8.2.3.2 i_ProviderAccessGetInterfaces

Esta interfaz define las siguientes operaciones:

– `getInterfaceTypes()`
– **getInterface**()
– **getInterfaces**()

8.2.3.3 i_ProviderAccessRegisterInterfaces

Esta interfaz define las siguientes operaciones:

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

8.2.3.4 i_ProviderAccessInterfaces

Esta interfaz hereda de `i_ProviderAccessGetInterfaces` e `i_ProviderAccessRegisterInterfaces` sin operaciones adicionales.

8.3 Visión de información común

En esta subcláusula se describen tipos comunes de información que tienen un alto potencial de reutilización (en otros puntos de referencia).

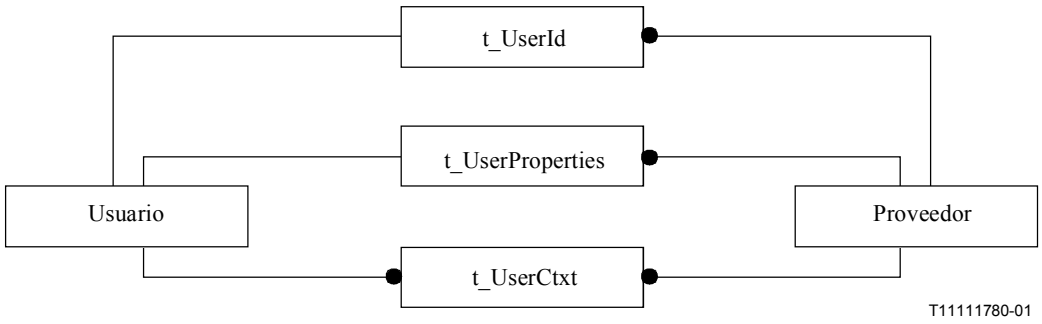


Figura 8-3 – Relación y cardinalidad de tipos comunes entre usuario y proveedor

8.3.1 Propiedades y listas de propiedades

Las propiedades son atributos o cualidades de algo. En Ret-RP, las propiedades se utilizan para asignar una cualidad a algo, o buscar algo que tiene esa cualidad particular.

Este algo para Ret-RP pueden ser usuarios, servicios, sesiones, interfaces, etc., cada uno de los cuales tendrá diferentes propiedades y cada propiedad puede tener una gama de diferentes valores y estructuras. (Asimismo para algunos ahora están claras las propiedades que se definirán, y algunas propiedades serán específicas del detallista.)

Habida cuenta de esto, se ha elegido el tipo t_Property para representar una propiedad. Su definición IDL se toma del servicio de objetos para comercio CORBA, y se copia en el módulo 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;
```

Como puede observarse, t_Property es una estructura que consta de un nombre y un valor. El nombre es una cadena internacional y el valor es any. Este formato permite al recibiente de la propiedad leer la cadena y compararla con las propiedades que conoce. Si es una propiedad que conoce, conocerá también el formato del valor. Si no conoce la propiedad, no debe leer el valor. (El valor any contiene un typecode (código de tipo) que puede ser visto en el depósito de interfaces para hallar el tipo del valor, pero esto debe ser innecesario la mayor parte del tiempo.)

t_Property y t_PropertyList se utilizan para atribuir cualidades a entidades, cuando no se desea definir todo lo que estas cualidades son actualmente. (Alguna de estas cualidades pueden ser también específicas de detallista, por lo que se puede utilizar también estos tipos para ampliar el Ret-RP.)

Por ejemplo, algunas características sobre el terminal son enviadas al detallista después que se establece una sesión de acceso. El tipo t_TerminalProperties se define como una lista de propiedades que permite enviar estas características al detallista. No están muy claras las características que tienen que ser enviadas al detallista para todos los tipos de terminal y puede ser necesario enviar alguna información diferente de otras.

Ret-RP define una propiedad particular para t_TerminalProperties denominada "TERMINAL INFO" que tiene un valor de tipo t_TerminalInfo. t_TerminalInfo es una estructura que mantiene alguna información sobre características de terminal. Cuando el detallista lee t_TerminalProperties y encuentra t_Property con el nombre "TERMINAL INFO", puede mirar el valor para hallar la característica de terminal. El valor será del tipo any, pero formatado con la información en la estructura t_TerminalInfo.

Sin embargo, t_TerminalInfo puede no estar completa, ni ser pertinente para todos los tipos de terminal. Si no contiene suficiente información, una futura versión de Ret-RP, o un detallista, puede definir otra propiedad, por ejemplo, "ADDITIONAL TERMINAL INFO", con un formato de valor apropiado, para contener la característica suplementaria. El detallista recibirá ambas propiedades en la lista t_TerminalProperties.

Si t_TerminalInfo contiene información que no es pertinente, el detallista puede definir una propiedad enteramente diferente y el consumidor debe enviarla en vez de la propiedad "TERMINAL INFO".

Ret-RP define nombres de propiedades y valores cuando es posible hacerlo. Para algunas listas de propiedades, por ejemplo, t_InterfaceProperties, corresponde al consumidor/detallista determinar las propiedades que pueden ser asociadas con ella.

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

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

t_MatchProperties se utiliza para abarcar los valores de retorno de algunas operaciones. Estas operaciones devuelven listas de algo. t_MatchProperties se utiliza para identificar lo que se ha de devolver, basado en las propiedades de ese algo (por ejemplo, para la operación listSubscribedServices, este algo son los servicios a que está abonado el consumidor. El parámetro t_MatchProperties define las propiedades de los servicios abonados que han de ser devueltos en la lista.)

t_MatchProperties contiene t_PropertyList y un tipo enumerado t_WhichProperties. t_PropertyList contiene las propiedades que tienen que concordar. t_WhichProperties identifica si alguna, todas o ninguna de las propiedades deben concordar y si debe concordar el nombre y valor de propiedad o sólo el nombre de propiedad.

Por ejemplo, en la operación listSubscriberServices, si t_WhichProperties es:

NoProperties, los servicios abonados no tienen que concordar con ninguna propiedad y así todos los servicios abonados son devueltos.

SomeProperties, los servicios abonados deben concordar por lo menos con una propiedad en la t_PropertyList (deben concordar el nombre y el valor de propiedad), que se han de incluir en la lista devuelta.

SomePropertiesNamesOnly, los servicios abonados deben concordar por lo menos con un nombre de propiedad en la t_PropertyList que se ha de devolver. Los valores de las propiedades en t_PropertyList pueden no ser significativos y no deben ser utilizados.

AllProperties, los servicios abonados deben concordar con todas las propiedades de t_PropertyList (deben concordar el nombre y el valor de propiedad) que se han de incluir en la lista devuelta.

AllPropertiesNamesOnly, los servicios abonados deben concordar con los nombres de propiedad en la t_PropertyList que se ha de devolver. Los valores de las propiedades en t_PropertyList pueden no ser significativos y no deben ser utilizados.

8.3.2 Información de usuario

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

t_UserId identifica el usuario ante el detallista. Es único de este usuario dentro del ámbito de este detallista. Se utiliza en requestNamedAccess() y es devuelto por getUserInfo(). t_UserId NO contiene el nombre del detallista, por lo que no puede ser utilizado para ponerse en contacto con él.

Puede ser enviado a un servicio de corredor/denominación cuando se intenta ponerse en contacto con un detallista junto con el nombre del detallista.

t_UserProperties es una secuencia de t_UserProperty. Contiene información sobre el usuario que se desea pasar al detallista. Se definen los siguientes nombres de propiedad para t_UserProperty. Se permiten otros nombres de propiedad, pero son específicos del detallista.

```
// 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 Excepción e_UserDetailsError

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

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

La excepción e_UserDetailsError se define para operaciones que requieren un parámetro t_UserDetails (por ejemplo inviteUserReq() en la parte utilización de Ret-RP). La excepción se plantea si t_UserName o t_UserProperties son no válidos.

Los siguientes códigos de error se pueden utilizar para definir el problema encontrado:

– InvalidUserName:

  El parámetro t_UserName no contiene un identificador de parte válido. (Esto puede deberse a que t_UserName esté formatado erróneamente, o t_UserName no se refiera a ningún usuario conocido.)

  La variable t_UserName en la excepción contiene el valor del parámetro t_UserName pasado en la invocación de operación.

– InvalidUserProperty:

  El parámetro t_UserProperties es erróneo. El elemento propertyError de la excepción describe el tipo de error en la propiedad de usuario. Si propertyError contiene InvalidPropertyName, el nombre de propiedad no es legal para esta operación. Si contiene InvalidPropertyValue, el valor no es un valor legal para el nombre de propiedad. Si propertyError contiene UnknownPropertyName, la sesión no reconoce el nombre de propiedad. Algunas sesiones pueden pasar por alto t_PropertyName's que no reconocen. No deben procesar t_PropertyValue asociado con t_PropertyName pero pueden procesar las otras t_Property's en el parámetro t_UserProperties. Estas sesiones no tienen que plantear la excepción con este código de error.

8.3.3 Información de contexto de usuario

```
       // 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;
};
```

`t_UserCtxt` informa al detallista sobre el dominio de consumidor, incluido el nombre del contexto, interfaces disponibles durante esta sesión de acceso e información de configuración de terminal. `t_UserCtxt` se utiliza solamente dentro de la parte acceso del Ret-RP, pero se incluye aquí para facilitar la lectura en la comprensión de `t_UserCtxtName`. En 8.4 figura una descripción completa.

`t_UserCtxtName` es un nombre dado a este contexto de consumidor. Es generado por el dominio de consumidor y se utiliza para distinguir entre sesiones de acceso a diferentes dominios/terminales de consumidor. Cuando enumera las sesiones de acceso, `t_UserCtxtName` es devuelto (junto con t_AccessSessionId), pues el primero debe ser un nombre legible por seres humanos para el "terminal" al cual está conectada la sesión de acceso.

8.3.4 Tipos relacionados con utilización

8.3.4.1 t_SessionId

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

Todas las operaciones en las interfaces de dominio de parte (incluidas todas las de ejecución y de información) incluyen un parámetro `t_SessionId`.

Esto permite al dominio de parte identificar la sesión de servicio que envía cada petición de operación. Su longitud es 32 bits. El `t_SessionId` es igual que el sessionId proporcionado por la operación `startService`() o `joinSession`() para esta sesión. (Es decir, el id que aparece en la lista listSessions en la parte de acceso que concuerda con este `t_SessionId` hará referencia a la misma sesión.) Si el dominio de parte no reconoce el `t_SessionId`, puede plantear un código de error `PD_InvalidSessionId` en la excepción `e_PartyDomainError`.

8.3.4.2 t_ParticipantSecretId

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

Todas las operaciones en interfaces de dominio de proveedor de la sesión de servicio (incluidas todas las peticiones) comprenden un parámetro `t_ParticipantSecretId`. Este tipo es devuelto también por peticiones para comenzar una sesión de servicio e incorporarse a la misma.

Permite a una sesión de servicio identificar el emisor de cada petición de operación. Es una clave de 128 bits cuyo formato no está definido, salvo que todos ceros supone que el participante no conoce o no requiere una clave. La sesión puede plantear un código de error `InvalidParticipantSecretId` en la excepción `e_UsageError`, si se necesita una clave para hacer una petición.

`t_ParticipantSecretId` se proporciona para que las sesiones puedan implementarse utilizando solamente una interfaz para todos los participantes. La sesión puede tener una seguridad razonable de que la petición proviene del usuario identificado y no de un usuario diferente.

No se prevé que t_ParticipantSecretId se utilice como el mecanismo de seguridad primario. Se debe utilizar la seguridad CORBA u otros contextos de seguridad para soportar las interacciones de dominio de parte-dominio de proveedor.

8.3.5 Invitaciones y anuncios

Las invitaciones permiten a una sesión pedir a un usuario de extremo específico que se incorpore a una sesión "en curso". Las invitaciones son entregadas al dominio de consumidor para el usuario de extremo, si existe una sesión de acceso. Si no existe una sesión de acceso con el dominio del consumidor, la invitación puede ser entregada a una interfaz "registrada previamente", o almacenada hasta que se establezca una sesión de acceso. Contiene información suficiente para que el usuario identifique al usuario que pidió enviar la invitación, halle la sesión y se incorpore a la misma o indique su rechazo. (Todas estas operaciones se definen a través de la parte acceso de Ret-RP y el detallista participa siempre para que el consumidor pueda hallar la sesión e incorporarse a la misma.)

```
// 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;
};
```

t_SessionInvitation describe la sesión de servicio a la cual el consumidor ha sido invitado, y proporciona un t_InvitationId para identificar esta invitación en la incorporación. (No debe dar referencias de interfaz a la sesión, ni ninguna información que permita al consumidor incorporarse a la sesión sin establecer primero una sesión de acceso con este detallista.) Proporciona también un t_UserId con el ID del usuario invitado. El dominio de consumidor puede verificar que la invitación es para un "usuario de extremo" conocido de este dominio.

t_SessionPurpose es una cadena que describe la finalidad de la sesión. Dicha finalidad se puede definir cuando comienza la sesión (mediante t_StartServiceSSProperties), o durante la sesión.

t_ServiceInfo es el servicio abonado que el consumidor puede utilizar para incorporarse a la sesión. Se describe en 8.4.

t_InvitationReason es una cadena que describe el motivo por el cual esta invitación fue enviada al usuario invitado y puede ser definida por la parte que solicitó la invitación, o por la sesión.

t_InvitationOrigin es una estructura que define dónde se generó la invitación y contiene el userId del usuario que comenzó la sesión y su sessionId para la sesión.

Se devuelve una t_InvitationReply que permite al consumidor informar al detallista sobre la acción que ejecutará en relación con la invitación. Se definen los siguientes códigos de respuesta:

– SUCCESS – el consumidor está de acuerdo en incorporarse a la sesión de servicio (el consumidor no tiene que establecer una sesión de acceso antes de poder incorporarse a la sesión de servicio. (Ésta NO tiene que ser establecida desde el terminal que recibió la invitación.) Utilizará joinSessionWithInvitation() en la interfaz i_RetailerNamedAccess para incorporarse a la sesión.) El consumidor puede utilizar replyToInvitation() para "cambiar de opinión" y no incorporarse a la sesión, pero debe haber respondido realmente con RINGING, u otro código de respuesta en vez de SUCCESS.

– UNSUCCESSFUL – no ha sido posible ponerse en contacto con el consumidor a través de esta operación. (No se incorporará a la sesión debido a esta invitación. Sin embargo, si se envió la misma invitación a múltiples interfaces, una respuesta de otra interfaz puede indicar que el consumidor se incorporará a la sesión.)

– DECLINE – el consumidor declina incorporarse a la sesión.

– UNKNOWN – el consumidor al que se envió la invitación no es conocido por esta interfaz. (t_SessionInvitation contiene un t_UserId para que el dominio de consumidor pueda verificar que la invitación es para un usuario conocido de este dominio.)

– FAILED – el consumidor no puede incorporarse a la sesión de servicio. (No se indica el motivo. La invitación puede estar formatada erróneamente, o el consumidor no puede incorporarse a la sesión.)

– FORBIDDEN – el dominio de consumidor no está autorizado a aceptar la petición.

– RINGING – el consumidor es conocido de este dominio y está siendo alcanzado. El detallista no debe suponer que el consumidor se incorporará a la sesión. (Si el consumidor desea incorporarse a la sesión, puede hacerlo como se describe en SUCCESS anterior. Si desea informar al detallista sobre su estado en relación con esta invitación, puede utilizar replyToInvitation() en la interfaz i_RetailerNamedAccess.)

– TRYING – el consumidor es conocido de este dominio pero no puede ser alcanzado directamente. El dominio de consumidor está ejecutando alguna acción para tratar de ponerse en contacto con el consumidor. El detallista puede tratar esto como RINGING.

– STORED – el consumidor es conocido de este dominio, pero por el momento no se está poniendo en contacto con él. La invitación ha sido almacenada para extracción por el consumidor. (El detallista puede tratar esto como RINGING, aunque puede pasar un rato antes de que el consumidor responda.)

– REDIRECT – el consumidor es conocido por este dominio pero no está disponible a través de esta interfaz. El detallista debe utilizar la dirección indicada en t_InvitationReplyProperies para ponerse en contacto con el consumidor.

– NEGOTIATE – el consumidor es conocido por este dominio, pero por el momento no se está poniendo en contacto con él. t_InvitationReplyProperties contiene un conjunto de alternativas que el detallista podrá aplicar para ponerse en contacto con el consumidor. (Estas alternativas no son definidas por Ret-RP y actualmente son específicas del detallista.)

– BUSY – el consumidor no puede ser alcanzado porque está "ocupado". Este código debe ser tratado como en el caso de UNSUCCESSFUL.

– TIMEOUT – el consumidor no puede ser alcanzado, porque ha expirado una temporización de dominio de consumidor mientras se trataba de alcanzarlo, es decir, el dominio de consumidor tiene un valor de temporización para ponerse en contacto con él utilizando un método (por ejemplo, ventaja, timbre telefónico) y esta temporización ha expirado. Este código debe tratarse como el caso de UNSUCCESSFUL.

Estos códigos de respuesta de invitación han sido tomados del proyecto de norma "Protocolo de iniciación de sesión" del Grupo de Trabajo de Ingeniería de Internet, Multimedia Multiparty Session Control (MMUSIC).

Los anuncios permiten que una sesión se anuncie a sí misma como un "grupo" de usuarios de extremo. Los anuncios no están dirigidos a un usuario específico ni son "entregados" al usuario de extremo. Son almacenados por el dominio de detallista hasta que el dominio de consumidor solicita una lista de anuncios. Los anuncios son devueltos al consumidor, dependiendo de los "grupos" a los cuales pertenece el usuario. (Éstos son definidos por propiedades de usuario, pero el Ret-RP no ha especificado mecanismos concretos para definir grupos de anuncios. Los anuncios contienen información suficiente para que el usuario se incorpore a la sesión. Esta operación se define a través de la parte de acceso del Ret-RP y el detallista participa siempre para que el consumidor pueda hallar la sesión e incorporarse a ella.)

Proyecto de definición: La estructura para anuncios es sólo un proyecto.

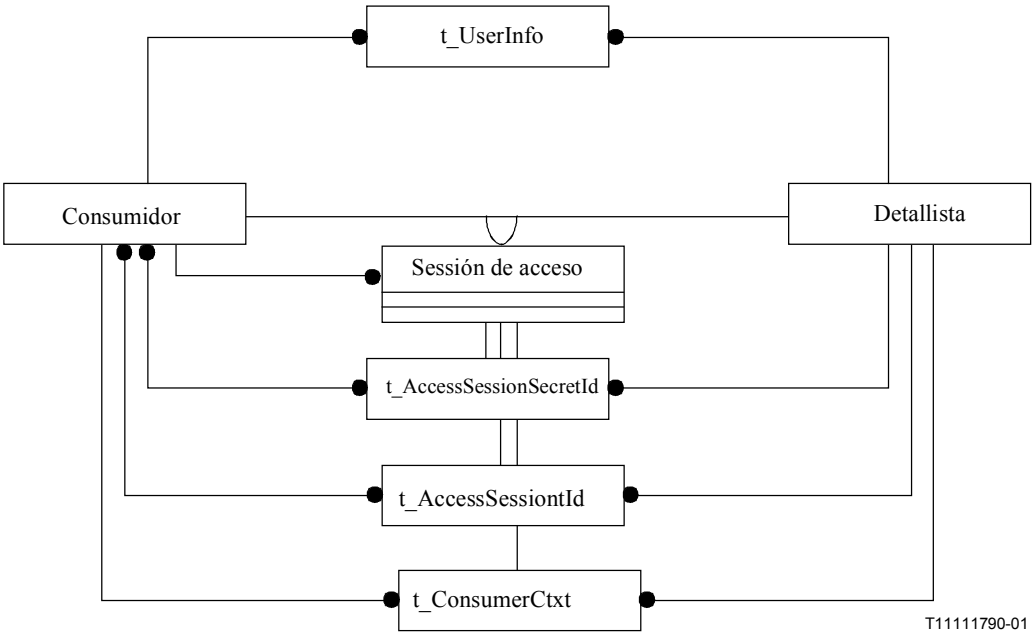
```
// module SPFEECommonTypes
typedef t_PropertyList t_AnnouncementProperties;

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;
```

t_SessionAnnouncement describe la sesión que se está anunciando y el "grupo" de usuarios al cual se difunde el anuncio. Es una estructura que contiene announcementId, sessionPurpose, serviceInfo y una lista de propiedades de anuncio. El Ret-RP no define nombres ni valores de propiedades para anuncios. Las propiedades de anuncio permiten a los detallistas definir sus propios tipos de anuncio que pueden ser pasados utilizando las operaciones de anuncio definidas por Ret-RP.

t_AnnouncementId identifica un anuncio al dominio de consumidor, que puede solicitar una lista de anuncios que están asociados con este usuario de extremo. El t_AnnouncementId es utilizado por el dominio de consumidor para distinguir los anuncios que recibe. Los identificadores para cada anuncio sólo pueden ser utilizados por este usuario y no identifican de manera única el anuncio para todos los consumidores de un detallista.

8.4 Visión de información de acceso

Esta subcláusula describe los tipos de información pasados a través del Ret-RP. Los tipos se pasan en operaciones definidas en las interfaces de acceso.



8.4.1 Información de sesión de acceso

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

t_AccessSessionId se utiliza para identificar una sesión de acceso. El t_AccessSessionId para la sesión de acceso vigente del consumidor es devuelto por requestNamedAccess() o requestAnonAccess(). Se puede encontrar el t_AccessSessionId para otras sesiones de acceso utilizando listAccessSessions() en la interfaz i_RetailerNamedAccess. (Los usuarios anónimos sólo pueden tener una sesión de acceso, y por tanto sólo un t_AccessSessionId.) El t_AccessSessionId es utilizado por el consumidor, es decir, para un consumidor (t_UserId) todos los t_AccessSessionId's son únicos.

t_AccessSessionSecretId se utiliza para identificar la sesión de acceso en la que se ha hecho una petición en requestNamedAccess(). Cada sesión de acceso de un consumidor tiene un t_AccessSessionSecretId único. Es devuelto por requestNamedAccess().

Todas las operaciones en requestNamedAccess() toman un t_AccessSessionSecretId como su primer parámetro. Este parámetro puede ser verificado por el detallista para determinar la sesión de acceso dentro de la cual el consumidor originó la petición. Esto es útil cuando el comportamiento de la petición depende del contexto del consumidor (por ejemplo, startService() verifica el contexto del consumidor para determinar si este servicio puede ser utilizado).

t_AccessSessionSecretId es sólo conocido dentro de la sesión de acceso en que es creado. No es conocido de otras sesiones de acceso del mismo consumidor, y no está disponible a través de listAccessSessions(). Esto se debe a que es utilizado para determinar la sesión de acceso dentro de la cual se hace la petición. Si otra sesión de acceso obtiene el t_AccessSessionSecretId de esta sesión de acceso, podría utilizarlo para pretender que la petición vino de esta sesión de acceso. Por este motivo, no debe ser visualizado a un consumidor humano, o a otra aplicación en el dominio de consumidor. t_AccessSessionSecretId no es por sí mismo un mecanismo de seguridad, por lo que se necesita la seguridad CORBA para establecer contextos de seguridad entre los dominios de

consumidor y detallista. Sin embargo, permite que el detallista descubra fácilmente el emisor de una petición.

8.4.2 Información de usuario

La mayor parte de la información relacionada con el usuario se describe en 8.3 sobre tipos comunes.

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

`t_UserInfo` describe al usuario y contiene el `t_UserId` del nombre de usuario y `t_UserProperties` es devuelto por `getUserInfo()` en la interfaz `i_ProviderAccess`.

8.4.3 Información de contexto de usuario

```
// 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;
};
```

`t_UserCtxt` informa al detallista sobre el usuario y el dominio de consumidor, incluido el nombre de contexto, las interfaces disponibles durante esta sesión de acceso e información de configuración de terminal.

`t_UserCtxtName` es un nombre dado a este contexto de consumidor. Es generado por el dominio de consumidor y se utiliza para distinguir entre sesiones de acceso a diferentes dominios/terminales de consumidor. Cuando enumera las sesiones de acceso, `t_UserCtxtName` es devuelto (junto con t_AccessSessionId) pues el primero debe ser un nombre legible por seres humanos para el "terminal" al cual está conectada la sesión de acceso.

`accessIR` es una referencia a la interfaz `i_UserAccess` soportada por el dominio de consumidor para uso en esta sesión de acceso.

`terminalIR` es una referencia a la interfaz `i_UserTerminal` soportada por el dominio de consumidor para uso en esta sesión de acceso.

`inviteIR` es una referencia a la interfaz `i_UserInvite` soportada por el dominio de consumidor para uso en esta sesión de acceso.

Todas las referencias a las tres interfaces precedentes se deben fijar a interfaces válidas en el dominio de consumidor.

---

[40] El tipo de este parámetro escrito en IDL es el tipo de clase básico. El tipo real dependerá del punto de referencia utilizado. En este caso, el detallista puede esperar el tipo i_Consumer<>. Véase también la observación en requestNamedAccess(), output namedAccessIR.

sessionInfoIR es una referencia a la interfaz i_UserSessionInfo soportada por el dominio de consumidor para uso en la sesión de acceso. No es necesario suministrar una referencia para esta interfaz.

t_TerminalConfig es una estructura que contiene el id y el tipo de terminal, el id y el tipo de acceso de red y una lista de propiedades del terminal. Se han definido dos tipos de propiedades t_TerminalInfo, descritos a continuación y t_ApplicationInfoList, una lista de las aplicaciones de usuario en el terminal.

t_TerminalInfo da detalles del tipo de terminal, sistema operativo, 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;
};
```

Proyecto de definición: Esta estructura es sólo un proyecto y actualmente está en revisión (la nueva versión se proporcionará para la respuesta revisada).

t_TerminalType es un tipo enumerado, que indica el tipo de terminal.

operatingSystem proporciona el tipo y versión de sistema operativo como una cadena.

networkCards y devices son listas de propiedades de los dispositivos físicos del terminal. Actualmente no están definidos los nombres y valores de propiedad, por lo que su uso es específico del detallista.

maxConnections es el número máximo de conexiones de red que pueden ser soportadas por el terminal.

memorySize es la cantidad de RAM en megabytes.

diskCapacity es la cantidad de almacenamiento en disco en megabytes.

8.4.4 Información de servicio y de sesión

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

t_ServiceInfo es una estructura que describe un servicio al que el consumidor está abonado.

t_ServiceId es el identificador para el servicio. t_ServiceId es único entre todos los servicios abonados del consumidor. (Otro consumidor puede estar abonado al mismo servicio, pero tendrá un t_ServiceId diferente.) El valor t_ServiceId persiste mientras dura el abono.

t_UserServiceName es el nombre del servicio como una cadena. El nombre es elegido por el abonado cuando se abona al servicio. Es el nombre del servicio visualizado al usuario.

t_ServiceProperties es una lista de propiedades, que define las características de este servicio. Pueden ser utilizadas para buscar tipos de servicios con las mismas características, por ejemplo, empleando discoverServices() en i_RetailerNamedAccess.) Actualmente no se han definido propiedades para t_ServiceProperties, por lo que su uso es específico del detallista.

```
// 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;
};
```

`t_SessionInfo` es una estructura, que contiene información que permite al dominio de consumidor hacer referencia a una sesión particular cuando utiliza interfaces dentro de una sesión de acceso (por ejemplo `i_RetailerNamedAccess`). Contiene también información para la parte utilización de la sesión, incluidas las referencias de interfaz para interactuar con la sesión.

`id` es el identificador para esta sesión. Es único para esta sesión entre todas las sesiones a través de las cuales éste consumidor interactúa con el detallista (es decir, si el consumidor interactúa con múltiples detallistas simultáneamente, éstos pueden devolver `t_SessionId`'s que son idénticos.)

`purpose` es una cadena que contiene la finalidad de la sesión y que puede haber sido definida cuando se creó la sesión, o después por interacciones específicas de servicio.

`secretId` es un identificador que el consumidor debe utilizar cuando interactúa con las interfaces en la sesión que son definidas por el modelo de sesión SPFEE. (Para más detalles, véase la parte utilización del Ret-RP.)

`myPartyId` es el identificador de parte de este consumidor. Si la sesión está utilizando el modelo de sesión SPFEE, con el conjunto de características MultipartyFS, este identificador será utilizado para identificar esta parte. Los `t_PartyId` de otras partes en la sesión están disponibles también a través de las interfaces MultipartyFS.

`state` es el estado de la sesión percibido por este consumidor. Puede ser: `UserUnknownSessionState`, `UserActiveSession`, `UserSuspendedSession`, `UserSuspendedParticipation`, `UserInvited`, `UserNotParticipating`. Pero cuando la sesión acaba de comenzar, es probable que sea `UserActiveSession`.

`itfs` es una lista de tipos y referencias de interfaz soportados por la sesión. Puede incluir interfaces específicas de servicio para que el consumidor interactúe con la sesión.

`sessionModels` es una lista de los modelos de sesión y conjuntos de características soportados por la sesión. Puede incluir referencias a interfaces soportados para conjunto de características.

`properties` es una lista de propiedades de la sesión y su uso es específico del detallista.

8.5 Definiciones de interfaz de acceso: Interfaces de dominio de consumidor

Esta subcláusula describe detalladamente todas las interfaces de consumidor soportadas en Ret-RP. Se define cada interfaz y sus operaciones.

Muchas de las operaciones son heredadas de otras interfaces, aunque se describan aquí como si fuesen definidas en las interfaces específicas de Ret-RP. Sólo las interfaces definidas en este Suplemento deben ser soportadas para la Ret-RP. No es necesario soportar la misma jerarquía de herencia definida anteriormente en 8.2.

Las siguientes interfaces son soportadas por el dominio de consumidor y están disponibles a través del Ret-RP:

– i_ConsumerInitial
– i_ConsumerAccess

– i_ConsumerInvite
– i_ConsumerTerminal
– i_ConsumerSessionInfo
– i_ConsumerAccessSessionInfo

8.5.1 Interfaz i_ConsumerInitial

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

Esta interfaz se proporciona para que un detallista pueda iniciar una sesión de acceso con el consumidor. Permite también que el consumidor reciba invitaciones fuera de una sesión de acceso.

La finalidad de esta interfaz es proporcionar un punto de contacto inicial para el detallista que desea ponerse en contacto con el consumidor. (Por tanto su finalidad es similar a la de la interfaz `i_RetailerInitial`). Sin embargo, esta interfaz sólo está disponible para un detallista si el consumidor ha registrado previamente la interfaz para uso fuera de una sesión de acceso. Esto se logra con la operación `registerInterfaceOutsideAccessSession` en la interfaz `i_RetailerNamedAccess`.

Las operaciones descritas a continuación son heredadas en esta interfaz de la interfaz `i_UserInitial`, que soporta los cometidos genéricos de usuario-proveedor. Para esta interfaz no se definen especializaciones específicas de Ret-RP.

Las siguientes firmas de operaciones se toman del módulo `SPFEEUserInitial`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEUserInitial::` cuando son utilizados por clientes de la interfaz `i_ConsumerInitial`.

8.5.1.1 requestAccess()

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

Proyecto de definición: Esta operación es sólo un proyecto.

Esta operación permite al detallista pedir que se establezca una sesión de acceso entre el consumidor y el detallista.

Esta operación sólo permite al detallista pedir que se establezca una sesión de acceso con el consumidor. No permite el establecimiento real de la sesión de acceso. Para establecer una sesión de acceso, el consumidor debe ponerse en contacto con el detallista, utilizando la interfaz `i_RetailerInitial` y pedir que se establezca una sesión de acceso.

El detallista pasa su `t_ProviderId` al consumidor que lo utiliza para ponerse en contacto con el proveedor, y obtener una referencia a una interfaz `i_RetailerInitial`.

El parámetro `t_AccessReply` permite al consumidor informar al detallista la acción que ejecutará en respuesta a la petición. Se definen los siguientes códigos de respuesta:

– `SUCCESS` – el consumidor está de acuerdo en establecer una sesión de acceso. (El consumidor establecerá la sesión de acceso como se describe anteriormente.)
– `DECLINE` – el consumidor declina iniciar una sesión de acceso.
– `FAILED` – el consumidor no puede establecer una sesión de acceso.
– `FORBIDDEN` – el dominio de consumidor no está autorizado a aceptar la petición.

8.5.1.2 inviteUserAccessSession()

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

Proyecto de definición: Esta operación es sólo un proyecto.

Específicamente, los parámetros `t_SessionInvitation` y `t_InvitationReply` se definen de acuerdo con el proyecto de norma "Protocolo de iniciación de sesión" del Grupo de Trabajo de Ingeniería de Internet, Multimedia Multiparty Session Control (MMUSIC).

Esta operación permite que un detallista envíe una invitación de incorporación a una sesión de servicio a un consumidor que no participa en una función de acceso.

Esta operación se usa si el consumidor ha registrado previamente esta interfaz para uso fuera de una sesión de acceso y el consumidor no está en ese momento en una sesión de acceso. Si el consumidor está en una sesión de acceso con este detallista, las invitaciones no serán enviadas con esta operación, sino serán entregadas a la interfaz `i_ConsumerAccess` que interviene en la sesión de acceso.

Para incorporarse a la sesión de servicio descrita por la invitación, el consumidor debe establecer una sesión de acceso con el detallista, y utilizar la operación `joinSessionWithInvitation`() en la interfaz `i_RetailerNamedAccess`. No es posible incorporarse a la sesión de servicio sin una sesión de acceso con el detallista.

`t_ProviderId` identifica al detallista ante el consumidor.

`t_SessionInvitation` describe la sesión de servicio a la cual se ha invitado al consumidor y proporciona un `t_InvitationId` para identificar esta invitación al incorporarse. (No da referencias de interfaz a la sesión ni información alguna que permita al consumidor incorporarse a la sesión sin establecer primero una sesión de acceso con este detallista.) Proporciona también un `t_UserId` con el id del usuario invitado. El dominio de consumidor puede verificar que la invitación es para un "usuario de extremo" conocido de este dominio. (Para más detalles, véase la subcláusula sobre "invitaciones y anuncios".)

Se devuelve `t_InvitationReply`, lo que permite al consumidor informar al detallista la acción que ejecutará en relación con la invitación. (Para más detalles, véase "Invitaciones y anuncios".)

8.5.1.3 cancelInviteUserOutsideAccessSession()

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

Proyecto de definición: Esta operación es sólo un proyecto.

Específicamente, los parámetros `t_SessionInvitation` y `t_InvitationReply` se definen de acuerdo con el proyecto de norma "Protocolo de iniciación de sesión" del Grupo de Trabajo de Ingeniería de Internet, Multimedia Multiparty Session Control (MMUSIC).

Esta operación permite a un detallista cancelar un invitación de incorporación a una sesión de servicio que había sido enviada a un consumidor. La operación se puede utilizar para cancelar invitaciones que han sido enviadas dentro de una sesión de acceso (con `inviteUser`() en `i_ConsumerInvite`) y fuera de una sesión de acceso (con `inviteUserOutsideAccessSession` en `i_ConsumerInitial`).

t_ProviderId identifica al detallista ante el consumidor.

t_InvitationId se utiliza junto con t_ProviderId para determinar la invitación que se ha de cancelar. (t_InvitationId's es única para un detallista solamente. Si un consumidor ha recibido invitaciones de varios detallistas, las invitaciones de diferentes detallistas pueden tener el mismo id.)

Si la lista t_InvitationId es desconocida para el consumidor, la operación debe plantear una excepción e_InvitationError con el código de error InvalidInvitationId.

8.5.2 Interfaz i_ConsumerAccess

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

Esta interfaz permite el acceso del detallista al dominio de consumidor durante una sesión de acceso. Proporciona operaciones para que el detallista pida referencias a interfaces soportadas por el dominio de consumidor. Estas interfaces incluyen las definidas por Ret-RP y otras interfaces específicas del detallista.

Su finalidad es similar a i_RetailerAccess, en cuanto a que está disponible durante la sesión de acceso. Es transferida al dominio de detallista como parte de la interfaz setUserCtxt() en la interfaz i_RetailerNamedAccess.

Todas las operaciones descritas a continuación son heredadas en esta interfaz de la interfaz i_UserAccess, que soporta los cometidos genéricos de usuario-proveedor. Para esta interfaz no se definen especializaciones específicas de Ret-RP.

Las siguiente firmas de operaciones son tomadas del módulo SPFEEUserAccess. Todos los tipos no abarcados tienen que ser abarcados por SPFEEUserAccess:: cuando son utilizados por clientes de la interfaz i_ConsumerAccess.

8.5.2.1 cancelAccessSession()

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

Proyecto de definición: Esta operación es sólo un proyecto.

cancelAccessSession() permite que el detallista termine una sesión de acceso con el consumidor. El detallista puede utilizar esta operación para terminar una sesión de acceso sin el permiso del consumidor.

Cuando se invoca esta operación, ha terminado la relación segura y fiable entre el consumidor y el detallista. Las interfaces del lado detallista y las del lado consumidor disponibles durante la sesión de acceso pueden ser utilizadas para hacer peticiones. (Las interfaces que han sido registradas para uso fuera de una sesión de acceso pueden aún ser utilizadas.)

options es una lista de propiedades que describe opciones específicas del detallista o acciones ejecutadas por el detallista al cancelar la sesión de acceso (es decir, el detallista puede haber suspendido la participación del consumidor en sus sesiones de servicio activas). Actualmente no se han definido nombre ni valores de propiedad para t_CancelAccessSessionProperties, por lo que su uso es específico del detallista.

Esta operación no afecta ninguna relación contractual entre el consumidor y el detallista. El consumidor puede pedir el establecimiento de una sesión de acceso y otras sesiones de acceso no habrán sido terminadas.

8.5.2.2 getInterfaceTypes()

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

Esta operación devuelve una lista de los tipos de interfaces soportados por el dominio de consumidor.

`itfTypes` son todos los tipos de interfaces soportados por el dominio de consumidor. Es una secuencia de `t_InterfaceTypeName`'s, que son cadenas que representan los tipos de interfaz soportados por el consumidor. `itfTypes` debe incluir todos los tipos de interfaz que pueden ser soportados por el consumidor.

Si la lista `itfTypes` no está disponible porque los tipos de interfaz soportados por la sesión no son conocidos, la operación debe plantear una excepción `e_ListError` con el error de código `ListUnavailable`.

8.5.2.3 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
);
```

Esta operación devuelve una interfaz, del tipo solicitada, soportada por el dominio de consumidor.

`type` identifica el tipo de la referencia de interfaz que se ha de devolver.

El parámetro `desiredProperties` se puede utilizar para identificar la interfaz que se ha de devolver. `t_MatchProperties` identifica las propiedades con las cuales las sesiones deben concordar. Define también si una sesión debe concordar con una, todas o ninguna de las propiedades. Actualmente no se han definido nombres ni valores de propiedad para Ret-RP, por lo que su uso es específico del detallista.

`itf` es devuelto por esta operación. Contiene el `t_InterfaceTypeName`, una referencia de interfaz (`t_IntRef`) y las propiedades de interfaz (`t_InterfaceProperties`) del tipo de interfaz solicitado.

Si el consumidor no soporta interfaces de este tipo, la operación debe plantear la excepción `e_InterfacesError` con el código de error `InvalidInterfaceType`.

Si se transfiere una propiedad no válida, la operación debe plantear `e_PropertyError`.

8.5.2.4 getInterfaces()

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

Esta operación devuelve una lista de todas las interfaces soportadas por el consumidor.

`itfs` es devuelto por esta operación. Es una `sequence` de estructuras `t_InterfaceStruct` que contienen `t_InterfaceTypeName`, una referencia de interfaz (`t_IntRef`) y las propiedades de interfaz (`t_InterfaceProperties`) de cada interfaz.

Si la operación no puede devolver las interfaces, o las rechaza, debe plantear la excepción `e_ListError`.

8.5.3 Interfaz i_ConsumerInvite

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

Esta interfaz permite al detallista enviar una invitación al consumidor pidiéndoles que se incorpore a una sesión de servicio. Sólo se puede utilizar durante una sesión de acceso para recibir invitaciones. Se pasa al dominio de detallista como parte de `setUserCtxt`() en la interfaz `i_RetailerNamedAccess`. Si el consumidor desea recibir invitaciones fuera de una sesión de acceso, debe registrar la interfaz i_ConsumerInitial.

Las operaciones descritas en las siguientes subcláusulas son heredadas en esta interfaz de la interfaz `i_UserInvite` que soporta los cometidos genéricos de usuario-proveedor. No se definen especializaciones específicas de Ret-RP para esta interfaz.

Las siguientes firmas de operaciones se toman del módulo `SPFEEUserAccess`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEUserAccess::` cuando son utilizados por clientes de la interfaz `i_ConsumerInvite`.

8.5.3.1 inviteUser()

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

Proyecto de definición: Esta operación es sólo un proyecto. Específicamente, los parámetros `t_SessionInvitation` y `t_InvitationReply` se definen de acuerdo con el proyecto de norma "Protocolo de iniciación de sesión" del Grupo de Trabajo de Ingeniería de Internet, Multimedia Multiparty Session Control (MMUSIC).

Esta operación permite a un detallista invitar al consumidor a incorporarse a una sesión de servicio. Sólo se puede utilizar durante una sesión de acceso.

`t_SessionInvitation` describe la sesión de servicio a la cual se ha invitado al consumidor, y proporciona un `t_InvitationId` para identificar esta invitación al incorporarse. (No da referencias de interfaz a la sesión, ni información que permitiría al consumidor incorporarse a la sesión fuera de una sesión de acceso con este detallista.)

Se devuelve `t_InvitationReply` que permite al consumidor informar al detallista de la acción que ejecutarán en relación con la invitación. (Para más detalles, véase "Invitaciones y anuncios".)

El consumidor se puede incorporar a la sesión de servicio descrita por la invitación, dentro de esta sesión de acceso, o puede establecer otra sesión de acceso con este detallista. El mismo `t_InvitationId` hará referencia a esta invitación en ambas sesiones de acceso. El consumidor debe utilizar `joinSessionWithInvitation`() en la interfaz `i_RetailerNamedAccess`. No es posible incorporarse a la sesión de servicio sin una sesión de acceso con el detallista.

8.5.3.2 cancelInviteUser()

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

Proyecto de definición: Esta operación es sólo un proyecto. Específicamente, los parámetros `t_SessionInvitation` y `t_InvitationReply` se definen de acuerdo con el proyecto de norma "Protocolo de iniciación de sesión" del Grupo de Trabajo de Ingeniería de Internet, Multimedia Multiparty Session Control (MMUSIC).

Esta operación permite a un detallista cancelar una invitación enviada a un consumidor para incorporarse a una sesión de servicio. La operación se puede utilizar para cancelar invitaciones que han sido enviadas dentro de una sesión de servicio (con `inviteUser`() en `i_ConsumerInvite`), y fuera de una sesión de servicio (con `inviteUserOutsideAccessSession` en `i_ConsumerInitial`).

`Se utiliza t_InvitationId` junto con el `t_ProviderId` para determinar la invitación que se ha de cancelar (los `t_InvitationId` son únicos en todas las sesiones de acceso con el mismo detallista).

Si el consumidor desconoce la lista de `t_InvitationId`, la operación debe plantear una excepción `e_InvitationError` con el código de error `InvalidInvitationId`. (Es posible recibir una `cancelInviteUser` antes de la correspondiente `inviteUser`, especialmente si la cancelación es enviada después de establecida la sesión de acceso. (De todos modos esta operación debe plantear la excepción.)

8.5.4 Interfaz i_ConsumerTerminal

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

Esta interfaz permite al detallista obtener información sobre la configuración del terminal y aplicaciones del dominio del consumidor. Se pasa al dominio de detallista como parte de `setUserCtxt`() en la interfaz `i_RetailerNamedAccess`. Si el consumidor desea que el detallista acceda a información de terminal fuera de una sesión de acceso, debe registrar esta interfaz, utilizando `registerInterfaceOutsideAccessSession`() en `i_RetailerNamedAccess`.

Proyecto de definición: Esta interfaz es sólo un proyecto. Concretamente, se puede mejorar esta interfaz para que el detallista haga preguntas más específicas sobre el dominio del consumidor.

Todas las operaciones descritas en las siguientes subcláusulas son heredadas en esta interfaz de la interfaz `i_UserTerminal`, que soporta los cometidos genéricos de usuario-proveedor. No se definen especializaciones específicas de Ret-RP para esta interfaz.

Las siguientes firmas de operaciones se toman del módulo `SPFEEUserAccess`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEUserAccess::` cuando es utilizado por clientes de la interfaz `i_ConsumerTerminal`.

8.5.4.1 getTerminalInfo()

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

Proyecto de definición: Esta operación es sólo un proyecto.

Esta operación permite al detallista recibir toda la información sobre la configuración del terminal del dominio de consumidor, a la cual el consumidor desea que el detallista acceda.

La operación devuelve la estructura `t_TerminalInfo`, con detalles del tipo de terminal, sistema operativo, etc. Véase 8.3.3, "Información de contexto de usuario".

8.5.5 Interfaz i_ConsumerAccessSessionInfo

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

Esta interfaz permite al detallista informar al consumidor los cambios de estados en otras sesiones de acceso con el consumidor (por ejemplo, sesiones de acceso con el mismo consumidor que son creadas o suprimidas). Sólo se informa al consumidor sobre las sesiones de acceso en las cuales participa.

Esta interfaz NO pasa automáticamente al detallista como parte de `setUserCtxt`() en la interfaz `i_RetailerNamedAccess`. Si el consumidor desea ser informado de los cambios en otra sesión de acceso, debe registrar esta interfaz con `registerInterface`() en `i_RetailerNamedAccess`. A continuación el detallista dirá al consumidor los cambios de sesión de acceso, hasta que esta interfaz sea desregistrada, o termine la sesión de acceso vigente.

Si el consumidor desea ser informado sobre los cambios de sesión de acceso fuera de una sesión de acceso, debe registrar esta interfaz con `registerInterfaceOutsideAccessSession`() en `i_RetailerNamedAccess`. Las operaciones no incluyen un `t_ProviderId`, por lo que si esta interfaz es registrada para uso fuera de una sesión de acceso, se debe registrar una interfaz distinta con cada detallista. Los detallistas no pueden compartir esta interfaz, porque `t_AccessSessionId` es única dentro de un detallista para este consumidor.

Todas las operaciones descritas en las siguientes subcláusulas son heredadas en esta interfaz de la interfaz `i_UserAccessSessionInfo`, que soporta los cometidos genéricos de usuario-proveedor. No se definen especializaciones específicas de Ret-RP para esta interfaz.

La siguientes firmas de operaciones se toman del módulo `SPFEEUserAccess`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEUserAccess::` cuando son utilizados por clientes de la interfaz `i_ConsumerAccessSessionInfo`.

8.5.5.1 newAccessSessionInfo()

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

Esta operación se usa para informar al consumidor que se ha establecido una nueva sesión de acceso.

`t_AccessSessionInfo` contiene el `t_AccessSessionId` de la nueva sesión, el `t_UserCtxtName`, de modo que el consumidor pueda decir el dominio/terminal de consumidor que ha establecido la sesión de acceso, y las `t_AccessSessionProperties` que son una lista de propiedades específicas del detallista que puede ser utilizada para proporcionar más información sobre la sesión de acceso.

8.5.5.2 endAccessSessionInfo()

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

Esta operación se usa para informar al consumidor que una sesión de acceso ha terminado.

`t_AccessSessionId` identifica la sesión de acceso que ha terminado.

8.5.5.3 cancelAccessSessionInfo()

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

Esta operación se utiliza para informar al consumidor que una sesión de acceso ha sido cancelada por el detallista. Para más detalles, véase 8.5.2.1, "cancelAccessSession()". `t_AccessSessionId` identifica la sesión de acceso que ha sido cancelada.

8.5.5.4 newSubscribedServicesInfo()

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

Esta operación se utiliza para informar al consumidor que ha sido abonado a algún nuevo servicio (El consumidor puede estar abonado a los servicios a través de un servicio en esta sesión de acceso u otra, o el consumidor puede tener sus usuarios abonados a un nuevo servicio.)

`t_ServiceList` es una lista de los servicios a los que está abonado el usuario. (Es una secuencia de estructuras `t_ServiceInfo`, véase "Información de servicio y de sesión".)

8.5.6 Interfaz i_ConsumerSessionInfo

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

Esta interfaz permite al detallista informar al consumidor los cambios de estado en las sesiones de servicio en las que participa el consumidor. Se invocan operaciones de información siempre que un cambio de la sesión de servicio afecta al consumidor (es decir, la sesión es suspendida), pero no cuando los cambios no lo afectan (es decir, otra parte en la sesión abandona). Esta interfaz es informada de los cambios en todas las sesiones de servicio en las que participa el consumidor, y no sólo de aquéllas asociadas con esta sesión de acceso.

Esta interfaz puede ser pasada al consumidor como parte de `setUserCtxt`() en la interfaz `i_RetailerNamedAccess`. Si el consumidor NO desea ser informado de los cambios en sus sesiones de servicio, esta interfaz NO tiene que ser pasada en `setUserCtxt`(). (Si no se pasa, el consumidor puede registrar esta interfaz utilizando `registerInterface`() en `i_RetailerNamedAccess`. A continuación el detallista informará al consumidor sobre los cambios de la sesión de servicio, hasta que esta interfaz sea desregistrada o termine la sesión de acceso vigente.)

Si el consumidor desea ser informado de los cambios de sesión de servicio fuera de una sesión de acceso, debe registrar esta interfaz, utilizando `registerInterfaceOutsideAccessSession`() en `i_RetailerNamedAccess`. Las operaciones no incluyen un `t_ProviderId`, por lo que si esta interfaz se registra para uso fuera de una sesión de servicio, se debe registrar una interfaz separada con cada detallista. Los detallistas no pueden compartir esta interfaz porque `t_SessionId` es único dentro de un detallista para este consumidor.

Todas las operaciones descritas en las subcláusulas siguientes son heredadas en esta interfaz de la interfaz `i_UserSessionInfo`, que soporta los cometidos genéricos de usuario-proveedor. No se definen especializaciones específicas de Ret-RP.

Se invocan las siguientes operaciones cuando una acción relacionada con este consumidor es ejecutada por la sesión de servicio. Estas operaciones ayudan a actualizar el conocimiento de la sesión de acceso sobre la participación del consumidor en sesiones de servicio. Se relacionan con

eventos que suelen ser específicos de utilización (específicos de servicio), pero que se consideran suficientemente genéricos para que sea útil informarlos a la sesión de acceso.

Sólo las operaciones asociadas con este consumidor producen operaciones de información, es decir, el consumidor A recibe una invocación `endMyParticipationInfo`() si termina su participación en una sesión, pero no recibe ninguna información si el consumidor B termina su propia participación. Si B terminase la participación de A, entonces A recibiría la información.

Todas las interfaces `i_ConsumerSessionInfo` reciben invocaciones de información cuando se ejecuta una acción en una sesión de servicio. Normalmente una de estas interfaces estará registrada en cada sesión de acceso. No importa en qué sesión de acceso se esté utilizando la sesión de servicio, todas las interfaces `i_ConsumerSessionInfo` recibirán una invocación de información.

Las siguientes firmas de operaciones se toman del módulo `SPFEEUserAccess`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEUserAccess::` cuando son utilizados por clientes de la interfaz `i_ConsumerSessionInfo`.

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

– El consumidor ha comenzado una nueva sesión de servicio. `session` contiene información sobre la nueva sesión que ha comenzado..

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

– Ha terminado una sesión de servicio. `sessionId` identifica la sesión.

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

– Ha terminado la participación del consumidor en una sesión de servicio. `sessionId` identifica la sesión.

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

– Una sesión de servicio ha sido suspendida. `sessionId` identifica la sesión.

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

– La participación del consumidor en una sesión de servicio ha sido suspendida. `sessionId` identifica la sesión.

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

– Una sesión de servicio suspendida ha sido reanudada. `sessionId` identifica la sesión. (El consumidor puede reincorporarse o no a la sesión de servicio, dependiendo de si él u otro consumidor reanudó la sesión.) `session` contiene información sobre la sesión reanudada.

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

– La participación del consumidor en la sesión de servicio ha sido reanudada. `session` contiene información sobre la sesión en la cual el consumidor ha reanudado su participación.

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

– El consumidor se ha incorporado a una sesión de servicio. `session` contiene información sobre la sesión a la que se ha incorporado el consumidor.

8.6 Definiciones de interfaz de acceso: Interfaces de dominio de detallista

Esta subcláusula describe todas las interfaces de detallista soportadas en Ret-RP. Se define cada interfaz y todas sus operaciones soportadas.

Muchas de las operaciones son heredadas de otras interfaces. Sin embargo, se describen como si fuesen definidas en las interfaces especificadas Ret-RP. Sólo las interfaces definidas en este Suplemento deben ser soportadas para el Ret-RP. No es necesario soportar la misma jerarquía de herencia definida anteriormente en 8.2.

Se han definido las siguientes interfaces para el dominio de detallista del Ret RP.

8.6.1 Interfaz i_RetailerInitial

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

La interfaz i_RetailerInitial es un punto de contacto inicial del consumidor con el detallista. Permite al consumidor pedir el establecimiento de una sesión de acceso entre él y el detallista.

Esta interfaz es devuelta cuando el consumidor se pone en contacto con el detallista. Ret-RP no especifica cómo el consumidor se pone en contacto con el detallista. Algunos ejemplos podrían ser: a través del servicio de denominación DPE, a través de otro tipo de servicio de directorio, a través del dominio comercial de corredor de SPFEE y a través del punto de referencia Bkr, o a través de un URL y una página de detallista. Se devuelve una interfaz de este tipo al consumidor como parte de la puesta en contacto con el detallista.

Esta interfaz hereda de la interfaz i_ProviderInitial. Define todas las operaciones que son genéricas para los cometidos usuario-proveedor y puede ser reutilizada en otros puntos de referencia entre dominios.

Esta interfaz desempeña un cometido en seguridad y puede utilizar la seguridad DPE para la criptación de mensajes y la autenticación de dominio. Es decir, el mensaje que pasa a través del DPE está protegido mediante criptación de los niveles acordados que varían y las credenciales de ambos dominios son intercambiadas para autenticación. Sin embargo, no impone que la autenticación y la adquisición de credenciales se produzca a través del DPE, por lo que proporciona la interfaz i_RetailerAuthenticate para poder autenticar al usuario, fuera de la seguridad DPE. Una referencia a la interfaz i_RetailerAuthenticate es pasada al dominio de consumidor por las operaciones requestNamedAccess() y requestAnonymousAccess() si el usuario no es autenticado por la seguridad DPE.

Las siguientes firmas de operaciones se toman del módulo `SPFEEProviderInitial`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEProvidedInitial::` cuando son utilizados por clientes de la interfaz `i_RetailerInitial`.

8.6.1.1 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
);
```

requestNamedAccess() permite al consumidor identificarse y pedir el establecimiento de una sesión de acceso con el detallista. Esta sesión proporciona acceso para utilizar sus servicios abonados, etc., a través de una interfaz i_RetailerNamedAccess.

Si los servicios de seguridad CORBA son utilizados por los DPE de los dominios de consumidor y detallista, las credenciales de ambos dominios y otra información de autenticación será intercambiada a través del DPE antes de invocar esta operación en el detallista. Esto significa que se puede haber establecido ya un contexto seguro para mensajes entre los dominios, y que la identidad del consumidor ha sido autenticada. En este caso, se establece una sesión de acceso, y se devolverá una referencia a la interfaz i_RetailerNamedAccess. Junto con esto se devuelve t_AccessSessionSecretId, que se ha de utilizar en todas las peticiones en la nueva interfaz.

Si no se están utilizando los servicios de seguridad CORBA, no hay un contexto seguro para los mensajes, y los mensajes DPE podrán ser interceptados y leídos por terceros.

Si el consumidor no ha sido autenticado aún, y el DPE no puede hacer la autenticación y establecer una sesión de acceso cuando se invoca esta operación, la misma fracasará. Se planteará una excepción e_AuthenticationError, que contiene una referencia a la interfaz i_RetailerAuthenticate. Esta interfaz se puede utilizar para autenticar y establecer el contexto seguro. Esta operación se invocará de nuevo para establecer la sesión de acceso.

`userId` identifica al consumidor ante el detallista. Para detalles sobre la estructura del `userId`, véase "Información de usuario".

`userProperties` es una secuencia de propiedades de usuario asociadas con este consumidor. En general, el consumidor no enviará información sensible al detallista hasta que se haya establecido una sesión de acceso. Sin embargo, se puede utilizar este parámetro para pasar la contraseña del consumidor al detallista, si ambos dominios utilizan la seguridad DPE para criptar los mensajes. Se puede enviar el contexto de seguridad y otra información que es comprendida por este detallista. Véase "Información de usuario".

S la petición tiene éxito, y el consumidor ha sido autenticado, se devuelven los siguientes parámetros de salida:

`namedAccessIR` es la referencia a la interfaz i_RetailerNamedAccess, que el dominio de consumidor utiliza durante la sesión de acceso.

NOTA – Aunque el IDL especifica el tipo (en texto) como `i_ProviderNamedAccess`, es sólo para indicar el tipo de referencia básico. Una (referencia de) interfaz abstracta nunca es exportada en un punto de referencia. La operación `requestNamedAccess()` se define dentro de la interfaz `i_ProviderNamedAccess` que es heredada ulteriormente en `i_RetailerNamedAccess` y utiliza así la (referencia de) interfaz de tipo `i_Retailer<...>`. El motivo para indicar el tipo de referencia básico en el IDL es permitir la reutilización en otras definiciones de punto de referencia. Para ello, `namedAccessIR` podrá ser también del tipo i_3ptyNamedAccess.

asSecretId es un t_AccessSessionSecretId siempre que el dominio del consumidor invoca una operación en namedAccessIR dentro de esta sesión de acceso. El asSecretId identifica el dominio del consumidor desde el cual se efectúan invocaciones en namedAccessIR. Este parámetro sólo se debe utilizar durante esta sesión de acceso, y sólo por el dominio de consumidor al cual se devuelve. Véase 8.4.

asId es un t_AccessSessionId utilizado para identificar esta sesión de acceso. Está disponible para todas las sesiones de acceso para este consumidor. Se puede utilizar para identificar esta sesión de acceso cuando se hacen peticiones en la interfaz i_RetailerNamedAccess entre este consumidor y el detallista, por ejemplo, utilizando listServiceSessions() se puede usar un t_AccessSessionId para abarcar la lista comenzada desde una sesión de acceso específica.

Si la petición fracasa, el consumidor no ha sido autenticado o la autenticación ha fracasado.

Se plantea una excepción e_AccessNotPossible si el detallista no puede permitir o rechaza que el dominio de consumidor establezca una sesión de acceso con él.

Se plantea una excepción e_AuthenticationError si el detallista no ha autenticado al consumidor. Contiene una lista de métodos de autenticación que pueden ser usados con la interfaz i_RetailerAuthenticate. La interfaz es devuelta como una referencia de interfaz, o como una referencia de objeto en forma de cadena, dependiendo del detallista. Esta referencia se utiliza para autenticar al consumidor ante el detallista. Una vez que el consumidor ha sido autenticado satisfactoriamente (utilizando uno de los métodos de autenticación indicados), el consumidor puede invocar esta operación de nuevo para pedir el establecimiento de una sesión de acceso, y obtener una referencia a la interfaz i_RetailerNamedAccess.

Si se plantea una excepción e_UserPropertiesError, hay un problema con userProperties. El errorCode proporciona el motivo del error.

8.6.1.2 requestAnonymousAccess()

```
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
);
```

La requestAnonymousAccess() permite al consumidor pedir el establecimiento de una sesión de acceso con el detallista. Se utiliza cuando el consumidor no tiene una identidad de usuario con el detallista, lo que puede deberse a que no se ha puesto en contacto previamente con el detallista o a que desea permanecer anónimo para éste.

Esta operación devuelve una referencia a la interfaz i_RetailerAnonAccess, a través de la cual el consumidor puede acceder a servicios y registrarse como un usuario denominado con el detallista, si lo desea.

Si los DPE de los dominios de consumidor y detallista están utilizando los servicios de seguridad CORBA, ambos dominios pueden intercambiar credenciales a través del DPE antes de invocar esta operación en el detallista. Esto significa que ya se ha establecido un contexto seguro para mensajes entre los dominios, pero las credenciales no contendrán información sobre la identidad del consumidor en cuestión.

Si no se están utilizando los servicios de seguridad CORBA, no hay contexto seguro para los mensajes, y los mensajes DPE podrían ser interceptados y leídos por terceros.

`userProperties` es una secuencia de propiedades de usuarios asociados con este consumidor. Puede contender contexto de seguridad y otra información que es comprendida por el detallista específico. Para más detalles, véase "Información de usuario".

Si la petición tiene éxito, se ha establecido una sesión de acceso con el consumidor y se devuelven los siguientes parámetros de salida:

`anonAccessIR` es la referencia a la interfaz i_RetailerAnonAccess, que el dominio de consumidor utiliza durante la sesión de acceso.

`asSecretId` es un `t_AccessSessionSecretId` utilizado cuando el dominio de consumidor invoca una operación en `namedAccessIR` dentro de esta sesión de acceso. El `asSecretId` identifica el dominio de consumidor desde el cual se hacen las invocaciones en `namedAccessIR`. Este parámetro sólo se debe utilizar durante esta sesión de acceso y sólo por el dominio de consumidor al cual se devuelve. Véase 8.4.

`asId` es un `t_AccessSessionId` utilizado para identificar a esta sesión de acceso. Está disponible para todas las sesiones de acceso para este consumidor. Puede ser utilizado para identificar esta sesión de acceso cuando se hacen peticiones en cualquier interfaz i_RetailerNamedAccess en una sesión de acceso entre este consumidor y el detallista. (En general, los usuarios anónimos sólo pueden tener una sesión de acceso con el detallista, pues cada sesión de acceso con cada usuario anónimo debe ser tratada separadamente. Como los consumidores son anónimos para el detallista, cada consumidor aparece como un individuo distinto, incluso si, de hecho, son la misma persona.)

Si la petición fracasa, el consumidor no ha sido autenticado o la autenticación ha fracasado.

Se plantea una excepción `e_AccessNotPossible` si el detallista no puede o rechaza permitir que el dominio de consumidor establezca una sesión de acceso con él.

Se plantea una excepción `e_AuthenticationError` si el detallista requiere que el dominio de consumidor sea autenticado. Contiene una lista de métodos de autenticación que pueden ser utilizados con la interfaz i_RetailerAuthenticate. (Los métodos de autenticación sólo pueden autenticar los dominios, y no los consumidores.) La interfaz se devuelve como una referencia de interfaz, o una referencia de objeto en forma de cadena, dependiendo del detallista. Esta referencia se utiliza para autenticar al dominio del consumidor ante el detallista. Una vez que el dominio de consumidor ha sido autenticado satisfactoriamente (utilizando uno de los métodos de autenticación indicados), el consumidor puede invocar esta operación de nuevo para pedir el establecimiento de una sesión de acceso y obtener una referencia a la interfaz i_RetailerAnonAccess.

Si se plantea una excepción `e_UserPropertiesError`, hay un problema con `userProperties`. El `errorCode` proporciona el motivo del error.

8.6.2 Interfaz i_RetailerAuthenticate

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

La interfaz i_RetailerAuthenticate permite al consumidor y al detallista ser autenticados. Proporciona un mecanismo genérico de autenticación que se puede usar para soportar varios protocolos de autenticación diferentes.

La finalidad de esta interfaz es verificar al consumidor y al detallista que cada dominio está interactuando con el dominio indicado. Esto significa autenticación mutua de ambos dominios. Son posible también otros esquemas de autenticación que sólo autentican a uno de los dominios utilizando esta interfaz. La interfaz proporciona un conjunto de operaciones genéricas que se pueden usar en la autenticación. Sin embargo, las operaciones sólo proporcionan un mecanismo para

"transportar" información de autenticación. Ambos dominios deben conocer y utilizar un protocolo de autenticación común, y aplicar este protocolo utilizando estas operaciones para autenticar los dominios. Ret-RP no especifica un protocolo de autenticación determinado. La operación getAuthenticationMethods() en esta interfaz se puede usar para determinar los protocolos de autenticación soportados por el detallista, y un protocolo elegido para autenticación. El protocolo de autenticación puede identificar o no al consumidor. Sólo puede identificar y autenticar el dominio del consumidor.

Las siguientes firmas de operaciones se toman del módulo `SPFEEProviderInitial`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEProviderInitial::` cuando son utilizados por clientes de la interfaz `i_RetailerInitial`.

8.6.2.1 getAuthenticationMethods()

```
void getAuthenticationMethods {
       in  t_AuthMethodSearchProperties desiredProperties,
       out t_AuthMethodDescList authMethods
) raises (
       e_AuthMethodPropertiesError,
       SPFEECommonTypes::e_ListError
);
```

getAuthenticationMethods() permite al consumidor pedir al detallista una lista de los métodos de autenticación soportados. El consumidor puede elegir un determinado método de autenticación para uso en authenticate().

`desiredProperties` es una lista que contiene las propiedades que el consumidor desea que soporte el método de autenticación. (Véase `t_MatchProperties` en "Propiedades y listas de propiedades".) Por ejemplo, el consumidor puede pedir que el método de autenticación devuelva el soporte de autenticación mutua, o autenticación del detallista solamente. Actualmente no se han definido nombres ni valores de propiedades para `t_AuthMethodSearchProperties`, por lo que su uso es específico del detallista.

`authMethods` es una lista de métodos de autenticación que concuerdan con `desiredProperties`, y que el detallista soporta. La estructura `t_AuthMethodDesc` contiene el identificador de método de autenticación y una lista de propiedades del método. Se supone que el consumidor y el detallista conocen el protocolo que se ha de aplicar para utilizar el método de autenticación definido.

La lista `authMethods` puede estar vacía. Esto ocurre si el detallista no soporta métodos que concuerdan con las propiedades solicitadas, o si el detallista no desea permitir que el consumidor efectúe la autenticación utilizando un método con las propiedades deseadas, por ejemplo, si el consumidor solicita un método para autenticar solamente al detallista y éste desea que la autenticación sea mutua.

Si el formato del parámetro `desiredProperties` es erróneo, o proporciona un nombre o valor de propiedad no válido, se debe plantear la excepción `e_PropertyError`. (Los nombres de propiedades no reconocidos pueden ser pasados por alto, si `desiredProperties` requiere que sólo concuerden algunas o ninguna de las propiedades.)

Si no se dispone de la lista `authMethods`, se plantea una excepción `e_ListError` con el código de error `ListUnavailable`.

8.6.2.2 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
      );
```

`authenticate`() permite al consumidor seleccionar un método de autenticación y pasar los datos de autenticación al detallista.

Cuando el dominio del consumidor ha determinado un método de autenticación con el detallista, esta operación se utiliza para transportar datos de autenticación y otras credenciales al detallista. Estos datos se usan para efectuar el tipo de autenticación apropiada al método de autenticación (que puede ser autenticación mutua o autenticación solamente del dominio de consumidor/detallista, etc.).

El detallista devuelve sus datos de autenticación (si es necesario), pide datos que el consumidor debe dar utilizando continueAuthentication() (si es necesario) y las credenciales solicitadas (si es posible). Si se requiere otro protocolo de autenticación antes de que las credenciales sean devueltas, éstas pueden ser devueltas por continueAuthentication().

El consumidor envía los siguientes parámetros al detallista:

`authMethod` se utiliza para identificar el método de autenticación propuesto por el consumidor. Afecta el método de composición y generación de los otros parámetros de datos opacos. Actualmente no se han definido valores de métodos de autenticación específicos para `t_AuthMethod`, por lo que su uso es específico del detallista.

`securityName` es el nombre asumido por el consumidor para la autenticación. Puede ser una cadena vacía de acuerdo con el método de autenticación utilizado.

`authenData` son datos opacos que contienen atributos de consumidor que han de ser autenticados. Su formato depende del método de autenticación utilizado.

`privAttribReq` son datos opacos utilizados para especificar los derechos y privilegios que el dominio de consumidor pide al dominio de detallista. Estos datos pueden corresponder con niveles de seguridad para acceder a diferentes sectores del dominio de detallista. Su formato depende del método de autenticación aplicado.

El detallista devuelve los siguientes parámetros al consumidor:

`privAttrib` son datos opacos que definen los atributos de privilegio concedidos al consumidor, basados en `privAttribReq` y sus datos de autenticación. Su formato depende del método de autenticación aplicado.

`continuationData` son datos opacos utilizados para hacer preguntas al consumidor. El consumidor no ha sido aún autenticado y debe procesar estos datos y devolver el resultado al detallista utilizando la operación `continueAuthentication()`. Su formato depende del método de autenticación aplicado. Este parámetro puede ser pasado por alto si el valor de `authStatus` no es `SecAuthContinue`.

`authSpecificData` son datos opacos específicos del método de autenticación aplicado.

`authStatus` identifica el estado del proceso de autenticación. Es un tipo enumerado con los siguientes valores:

– SecAuthSuccess

  La autenticación se ha completado satisfactoriamente. No se necesitan más invocaciones a `continueAuthentication()`. El consumidor puede invocar requestNamedAccess() en la interfaz i_RetailerInitial para obtener una referencia a la interfaz i_RetailerNamedAccess. (O si desea ser un usuario anónimo, invocar requestAnonymousAccess() para i_RetailerAnonAccess.)

– SecAuthFailure

  La autenticación no se ha completado satisfactoriamente. El consumidor no ha sido autenticado, y no podrá establecer una sesión de acceso. Las invocaciones de requestNamedAccess() continuarán planteando `e_AccessNotPossible`, o `e_AuthenticationError`.

– SecAuthContinue

  La autenticación continúa, y el consumidor debe contestar a este resultado invocando `continueAuthentication()`.

– SecAuthExpired

  La autenticación ha expirado. El consumidor no invocó continueAuthentication() con suficiente rapidez después de la respuesta de authenticate(), o la invocación anterior a continueAuthentication(). Hay que repetir la autenticación desde el principio invocando authenticate(). Esta enumeración no debe ser devuelta por authenticate().

8.6.2.3 continueAuthentication()

```
void continueAuthentication{
       in t_opaque responseData,
       out t_opaque privAttrib,
       out t_opaque continuationData,
       out t_opaque authSpecificData,
       out t_AuthenticationStatus authStatus
);
```

`continueAuthentication`() permite al consumidor continuar un protocolo de autenticación, comenzado utilizando `authenticate`() y pasar los datos de autenticación al detallista.

Esta operación debe ser invocada por el consumidor si `authStatus` devuelto por `authenticate`(), o una invocación anterior a `continueAuthentication`() es `SecAuthContinue`. Ambas operaciones utilizan `authStatus` para indicar si el consumidor tiene que invocar de nuevo esta operación. Los parámetros devueltos por esta operación deben ser procesados por el consumidor de acuerdo con el método de autenticación, y los resultados se deben proporcionar en parámetros para la subsiguiente invocación de esta operación.

`responseData` son datos opacos del consumidor. Estos datos han sido generados por el consumidor de acuerdo con el método de autenticación, basados en los `continuationData` devueltos por la invocación anterior a `authenticate`() o `continueAuthentication`(). La forma precisa en que estos datos son generados y su formato es específica del método de autenticación aplicado.

`continuationData` son datos opacos utilizados para hacer preguntas al consumidor, que no ha sido aún autenticado y debe procesar estos datos y devolver el resultado al detallista utilizando la operación `continueAuthentication()`. Su formato depende del método de autenticación aplicado. Este parámetro puede ser pasado por alto si el valor de `authStatus` no es `SecAuthContinue`.

`authSpecificData` son datos opacos específicos del método de autenticación aplicado.

`authStatus` identifica el estado del proceso de autenticación. Tiene los mismos valores que para `authenticate`().

8.6.3 Interfaz i_RetailerAccess

```
// module SPFEEProviderAccess
interface i_RetailerAccess
{
};
```

i_RetailerAccess es una interfaz abstracta utilizada para heredar operaciones comunes en las interfaces i_RetailerNamedAccess, e i_RetailerAnonAccess.

La finalidad de esta interfaz es la herencia, como se describe anteriormente. No debe estar disponible por encima del Ret RP. No se debe crear casos de este tipo de interfaz.

Actualmente no se definen operaciones para esta interfaz. Contendrá operaciones que son compartidas entre las interfaces i_RetailerNamedAccess, e i_RetailerAnonAccess. Actualmente todas las operaciones se definen en la interfaz i_RetailerNamedAccess y no se han identificado operaciones para la interfaz i_RetailerAnonAccess.

8.6.4 Interfaz i_RetailerNamedAccess

```
// module SPFEEProviderAccess
interface i_RetailerNamedAccess
       : i_ProviderNamedAccess, i_RetailerAccess
{
            // Inherited operations shown in following subsections.
};
```

La interfaz i_RetailerNamedAccess permite que un consumidor conocido acceda a los servicios a los que está abonado. El consumidor la utiliza para todas las operaciones dentro de una sesión de acceso con el detallista.

Esta interfaz es devuelta cuando el consumidor ha sido autenticado por el detallista y se ha establecido una sesión de acceso. Es devuelta pidiendo requestNamedAccess() en la interfaz i_RetailerInitial.

Esta interfaz hereda de las interfaces i_ProviderNamedAccess e i_RetailerAccess. i_ProviderNamedAccess define todas las operaciones que son genéricas para acceder a cometidos de usuario-proveedor, y pueden ser reutilizadas en otros puntos de referencia entre dominios. Todas las operaciones en esta interfaz son heredadas. La interfaz i_RetailerAccess está actualmente en blanco. Contendrá operaciones que son compartidas entre la interfaz i_RetailerAnonAccess y esta interfaz, y que son específicas del Ret RP.

Las siguientes signaturas de operaciones son tomadas del módulo `SPFEEProviderAccess`. Todos los tipos no abarcados tienen que ser abarcados por `SPFEEProviderAccess::` cuando son utilizados por clientes de la interfaz `i_RetailerAccess`.

8.6.4.1 setUserCtxt()

```
void setUserCtxt (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in t_UserCtxt userCtxt
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_UserCtxtError
);
```

setUserCtxt() permite que el consumidor informe al detallista sobre las interfaces en el dominio del consumidor, y da otra información de dicho dominio (por ejemplo, aplicaciones de usuario disponibles en el dominio de consumidor, sistema operativo utilizado, etc.).

`userCtxt` es una estructura que contiene información de configuración e interfaces del dominio de consumidor.

Esta operación debe ser invocada inmediatamente después de recibir la referencia a esta interfaz. Si esta operación no ha sido invocada satisfactoriamente, las operaciones subsiguientes pueden originar una excepción `e_AccessError` con un código de error `UserCtxtNotSet.`

Si se plantena un problema con `userCtxt`, entonces `e_UserCtxtError` debe ser planteado con el código de error apropiado.

8.6.4.2 getUserCtxt()

```
void getUserCtxt (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_UserCtxtName ctxtName,
       out t_UserCtxt userCtxt
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_UserCtxtError
);
```

Esta operación permite que el consumidor extraiga información sobre contextos de usuario que han sido registrados en el detallista.

`ctxtName` es el nombre del contexto que el consumidor desea para extraer información de contexto de usuario. (`ctxtName` es fijado por el consumidor cuando registra un contexto de usuario, y es el término del consumidor para el contexto, por ejemplo,"Hogar", "Trabajo", "La casa de mamá", etc.)

`userCtxt` es una estructura que contiene información de configuración e interfaces del dominio de consumidor.

8.6.4.3 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 listAccessSessions()

```
void listAccessSessions (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       out SPFEEAccessCommonTypes::t_AccessSessionList asList
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       SPFEECommonTypes::e_ListError
);
```

listAccessSessions() devuelve una lista de sesiones de acceso, que contiene todas las sesiones de acceso que el consumidor ha establecido actualmente con este detallista. Es una `sequence` de estructuras `t_AccessSessionInfo`, que constan de `t_AccessSessionId`, `t_UserCtxtName` y `t_AccessSessionProperties`. La última es `t_PropertyList`. Actualmente no se han definido nombres ni valores de propiedad específicos para `t_AccessSessionProperties`, por lo que su uso es específico del detallista.

La información devuelta por esta operación puede ser usada por el consumidor para hallar las otras sesiones de acceso que están establecidas en ese momento, terminar algunas de estas sesiones de acceso (véase `endAccessSession ()`); enumerar las sesiones de servicio de esas sesiones de acceso (véase `listServiceSessions ()`); y ser informado de los cambios de estas sesiones de acceso y sesiones de servicio (véase `i_ConsumerAccessSessionInfo` e `i_ConsumerSessionInfo` interfaces).

Si la lista `asList` no está disponible, porque las sesiones de acceso del consumidor no están disponibles, la operación debe plantear una excepción `e_ListError` con el código de error `ListUnavailable`.

8.6.4.5 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
);
```

`endAccessSession ()` permite al consumidor terminar una sesión de acceso.

La operación puede terminar la sesión de acceso vigente, una sesión de acceso especificada, o todas las sesiones de acceso (incluida la vigente) mediante el uso del parámetro `t_SpecifiedAccessSession.`

`t_EndAccessSessionOptions` permite que el consumidor elija las acciones que el detallista debe ejecutar, si hay sesiones de servicio activas o suspendidas cuando la sesión de acceso termina. Las acciones son usadas solamente como parte de esta invocación. El detallista no recuerda la acción elegida. (Los detallistas pueden definir una política por defecto para sesiones de servicio cuando un consumidor termina la sesión de acceso en la cual fueron creadas, o permitir que el consumidor defina la política. Actualmente, Ret RP no admite la definición de esta política por el consumidor.)

Si el formato es erróneo, o proporciona un id de sesión de acceso no válido, se debe plantear la excepción `e_SpecifiedAccessSessionError.`

Se plantea `e_EndAccessSessionError` si la opción es no válida, o las sesiones de servicio permanecen activas, o son suspendidas, lo cual no está permitido por el detallista. (Un consumidor puede terminar una sesión de acceso, dejando la sesiones activas o suspendidas si esto se permite como una política del detallista para este consumidor.)

8.6.4.6 getUserInfo()

```
void getUserInfo(
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       out SPFEEAccessCommonTypes::t_UserInfo userInfo
) raises (
       SPFEEAccessCommonTypes::e_AccessError
);
```

`getUserInfo ()` permite al consumidor pedir información sobre sí mismo.

Esta operación devuelve una estructura `t_UserInfo` como un parámetro de salida, Contiene el `t_UserId` del consumidor, su nombre y una lista de propiedades del usuario. Actualmente no se han definido nombres ni valores de propiedades para `t_UserProperties`, por lo que su uso es específico del detallista.

8.6.4.7 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
);
```

`listSubscribedServices ()` devuelve una lista de servicios a los cuales el consumidor se ha abonado previamente.

Se puede utilizar el parámetro `desiredProperties` para indicar la lista de los servicios abonados. `t_SubscribedServiceProperties` identifica las propiedades con las que deben concordar los servicios abonados. Define también si un servicio abonado debe concondar con una, todas o ninguna de las propiedades (véase `t_MatchProperties` en "Propiedades y listas de propiedades"). actualmente no se han definido nombres ni valores de propiedades específicos para `t_SubscribedServiceProperties`, por lo que su uso es específico del detallista.

La lista de servicios a los que está abonado el consumidor y la concordancia con `desiredProperties`, es devuelta en `t_ServiceList`. Ésta es una secuencia de estructuras `t_ServiceInfo`, que contiene `t_ServiceId`, `t_UserServiceName` (nombre de consumidores del servicio) y una secuencia de propiedades del servicio, `t_ServiceProperties`. Actualmente no se han definido nombres ni valores de propiedad específicos para `t_ServiceProperties`, por lo que su uso es específico del detallista.

Si el formato del parámetro `desiredProperties` es erróneo, o se proporciona un nombre o valor de propiedad no válido, se debe plantear la excepción `e_PropertyError`. (Los nombres de propiedades no reconocidos pueden ser pasados por alto, si `desiredProperties` requiere que sólo concuerden algunas o ninguna de las propiedades.)

Si no se dispone de la lista de servicios, porque los servicios del detallista no están disponibles, la operación debe plantear una excepción `e_ListError` con el código de error `ListUnavailable`.

8.6.4.8 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
);
```

`discoverServices()` devuelve una lista de los servicios disponibles del detallista.

Esta operación se usa para descubrir los servicios proporcionados por el detallista para uso del consumidor. Se puede emplear para extraer información sobre todos los servicios prestados, o ser indicada por el parámetro `desiredProperties`. (Véase `t_MatchProperties` en "Propiedades y listas de propiedades".)

La lista de servicios de detallista que concuerdan con `desiredProperties` es devuelta en `services`. Ésta es una secuencia de estructuras `t_ServiceInfo`, que contienen `t_ServiceId`, `t_UserServiceName` (nombre del consumidor para el servicio), y una `sequence` de propiedades del servicio, `t_ServiceProperties`. Actualmente no se han definido nombres ni valores de propiedad específicos para `t_ServiceProperties`, por lo que su uso es específico del detallista.

El parámetro `howMany` define el número de estructuras `t_ServiceInfo` que se han de devolver en el parámetro `services`. La longitud de servicios no excederá de este número. Los servicios restantes que concuerden con `desiredProperties`, pero que no están incluidos en `services` son accesibles a través de `iteratorIR`, la interfaz `i_DiscoverServicesIterator`. Si no hay servicios restantes, `iteratorIR` debe ser nulo.

Si el formato del parámetro `desiredProperties` es erróneo, o proporciona un nombre o valor de propiedad no válido, se debe plantear la excepción `e_PropertyError`. (Los nombres de propiedades no reconocidos pueden ser pasados por alto, si `desiredProperties` requiere que sólo concuerden algunas o ninguna de las propiedades.)

Si no se dispone de la lista de `services`, porque los servicios del detallista no están disponibles, la operación debe plantear una excepción `e_ListError` con el código de error `ListUnavailable`.

8.6.4.9 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
);
```

getServiceInfo() devuelve información sobre un servicio específico, identificado por el serviceId. La lista desiredProperties puede contener la información que se ha de devolver.

8.6.4.10 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
);
```

Esta operación extrae información sobre cómo telecargar ssUAP en caso de Java applets. Se incluye terminalInfo como un parámetro de red inteligente para evitar una invocación explícita de la operación getTerminalInfo.

8.6.4.11 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
);
```

`listServiceSessions ()` devuelve una lista de las sesiones de servicio, en las que participa el consumidor, que incluye las sesiones activas y suspendidas.

El parámetro `as` contiene la lista de las sesiones por la sesión de acceso en la cual son utilizadas. Puede identificar la sesión de acceso vigente, una lista de sesiones de acceso, o todas las sesiones de acceso. [Una sesión está asociada con una sesión de acceso si está siendo utilizada dentro de esa sesión de acceso, o si ha sido suspendida (o suspendida la participación), y está siendo utilizada dentro de esa sesión de acceso cuando fue suspendida.]

El parámetro desiredProperties puede ser utilizado para indicar la lista de sesiones. t_SessionSearchProperties identifica las propiedades con las cuales deben concordar la sesiones. Define también si una sesión debe concordar con una, todas o ninguna de las propiedades. (Véase t_MatchProperties en "Propiedades y listas de propiedades".) Se han definido los siguientes nombres y valores de propiedad para t_SessionSearchProperties:

– nombre: "SessionState"

  valor: t_SessionState

  Si una propiedad en t_SessionSearchProperties tiene el nombre "SessionState", la sesión debe tener el mismo t_SessionState indicado en el valor de propiedad.

– nombre: "UserSessionState"

  valor: t_UserSessionState

  Si una propiedad en t_SessionSearchProperties tiene el nombre "UserSessionState", la sesión debe tener el mismo t_UserSessionState indicado en el valor de propiedad.

Es posible definir también otras propiedades específicas de detallista en desiredProperties.

La lista de sesiones que concuerdan con desiredProperties y accessSession son devueltos en sessions. Ésta es una sequence de estructuras t_SessionInfo, que definen t_SessionId, t_ParticipantSecretId, t_PartyId, t_UserSessionState, t_InterfaceList, t_SessionModelList y t_SessionProperties de la sesión.

Si el formato del parámetro as es erróneo, o proporciona un id de sesión de acceso no válido, se debe plantear la excepción e_SpecifiedAccessSessionError.

Si el formato del parámetro desiredProperties es erróneo o proporciona un nombre o valor de propiedad no válidos, se debe plantear la excepción e_PropertyError. (Los nombres de propiedades no reconocidos pueden ser pasados por alto, si desiredProperties requiere que sólo concuerden algunas o ninguna de las propiedades.)

Si no se dispone de la lista de sessions, porque las sesiones del detallista no están disponibles, la operación debe plantear una excepción e_ListError con el código de error ListUnavailable.

8.6.4.12 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
);
```

getSessionModels () devuelve un a lista de modelos de sesión soportados por una sesión de servicio. Se puede utilizar en sesiones activas y suspendidas.

sessionId identifica la sesión cuyos modelos de sesión son extraídos.

sessionModels son los modelos de sesión soportados por la sesión. Es una sequence de estructuras t_SessionModel, que contienen el nombre del modelo de sesión y una lista de propiedades para ese modelo de sesión. En el SPFEE se ha definido un modelo denominado el modelo de sesión SPFEE. En [6] figura más información sobre el modelo de sesión genérico orientado a objetos, pero está fuera del ámbito de este Suplemento.

Se plantea e_SessionError si el sessionId es no válido, o el estado de la sesión excluye el acceso a los modelos de sesión (por ejemplo, la sesión está suspendida), o la sesión rechaza devolver sessionModels.

Si no se dispone de la lista sessionModels, porque los modelos de sesión soportados por la sesión no son conocidos, la operación debe plantear una excepción e_ListError con el código de error ListUnavailable.

8.6.4.13 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
);
```

getSessionInterfaceTypes () devuelve una lista de tipos de interfaces soportados por una sesión de servicio. Se puede utilizar en sesiones activas o suspendidas.

sessionId identifica la sesión cuyos tipos de interfaz son extraídos.

itfTypes son todos los tipos de interfaz soportados por la sesión. Es una sequence de t_InterfaceTypeName, que son las cadenas que representan los tipos de interfaz soportados por la sesión. itfTypes debe incluir todos los tipos de interfaces que pueden ser soportados por la sesión.

Se plantea e_SessionError si el sessionId es no válido, o el estado de la sesión excluye el acceso a los modelos de sesión (por ejemplo, la sesión está suspendida); o la sesión rechaza devolver itfTypes.

Si la lista itfTypes no está disponible, porque los tipos de interfaz soportados por la sesión no son conocidos, la operación debe plantear una excepción e_ListError con el código de error ListUnavailable.

8.6.4.14 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
);
```

getSessionInterface () devuelve una interfaz, del tipo solicitado, soportado por una sesión de servicio. Se puede utilizar en sesiones activas.

sessionID identifica la sesión cuya interfaz es extraída.

itfType identifica el tipo de interfaz de la referencia de interfaz que se ha de devolver.

itf es devuelto por esta operación. Contiene el t_InterfaceTypeName, una referencia de interfaz (t_IntRef) y las propiedades de interfaz (t_InterfaceProperties) del tipo de interfaz solicitado.

Se plantea e_SessionError si el sessionId es no válido, o el estado de la sesión excluye el acceso a los modelos de sesión (por ejemplo, la sesión está suspendida) o la sesión rechaza devolver itfTypes.

Si la sesión no soporta interfaces de itfType, la operación debe plantear ed_SessionInterfacesError, con el código de error InvalidSessionInterfaceType.

8.6.4.15 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
);
```

`getSessionInterfaces ()` devuelve una lista de todas las interfaces soportadas por una sesión de servicio. Sólo se puede utilizar en sesiones activas.

`sessionId` identifica la sesión cuyos tipos de interfaces son extraídos.

`itfs` es devuelto por esta operación. Es una `sequence` de estructuras `t_InterfaceStruct` que contienen el `t_InterfaceTypeName`, una referencia de interfaz (`t_IntRef`) y las propiedades (`t_InterfaceProperties`) de cada interfaz.

Se plantea `e_SessionError` si `sessionId` es no válido, o el estado de la sesión excluye el acceso a los modelos de sesión (por ejemplo, la sesión está suspendida) o la sesión rechaza devolver `itfTypes.`

Si la lista `itfs` no está disponible, porque las interfaces soportadas por la sesión no son conocidas, la operación debe plantear una excepción `e_ListError` con el código de error `ListUnavailable`.

8.6.4.16 listSessionInvitations()

```
void listSessionInvitations (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       out SPFEEAccessCommonTypes::t_InvitationList invitations
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       SPFEECommonTypes::e_ListError
);
```

`listSessionInvitations ()` devuelve una lista de las invitaciones de incorporación a una sesión de servicio que han sido enviadas al consumidor a través del detallista.

`invitations` es devuelta por esta operación. Es una secuencia de estructuras de `t_SessionInvitation`:

```
struct t_SessionInvitation {
       t_InvitationId id;
       t_UserId inviteeId;
       t_SessionPurpose purpose;
       t_InvitationReason reason;
       t_InvitationOrigin origin;
};
```

`id` identifica la invitación particular. Identifica de manera única esta invitación con respecto a otras para este consumidor en este detallista. (Otros consumidores con este detallista pueden tener invitaciones con el mismo id). Este id se usa en `joinSessionWithInvitation`() para incorporación a la sesión designada por esta invitación.

`inviteeId` es el id de usuario de este consumidor. (No es necesario aquí, porque el id de usuario es conocido a través de la sesión de acceso. Se incluye en esta estructura para poder entregar invitaciones fuera de una sesión de acceso, y que el recibiente pueda verificar que la invitación es para él.)

`purpose` es una cadena que contiene la finalidad de la sesión.

reason es una cadena que contiene el motivo por el cual este consumidor ha sido invitado a incorporarse a la sesión.

origin es una estructura que contiene el userId del consumidor que ha solicitado el envío de la invitación a este consumidor, y su sessionId para la sesión a la que ha sido invitado este consumidor. (El sessionId se proporciona para que si el consumidor invitado se pone en contacto con el consumidor invitante, pueda indicar a que sesión se refiere.)

Si no se dispone de la lista de invitaciones, la operación debe plantear e_ListError, con el código de error ListUnavailable.

8.6.4.17 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
);
```

listSessionAnnouncements() devuelve una lista de los anuncios de sesiones que han sido emitidos a través de este detallista.

Las sesiones pueden ser anunciadas debido a peticiones de participantes en sesiones (véase el conjunto de características multipartitas), o debido a propiedades de inicialización de la sesión, fábrica de servicios o políticas del usuario que comienza el servicio. El proceso por el cual se anuncian las sesiones no es definido por el Ret-RP. Sin embargo, esta operación se proporciona para que un consumidor pueda pedir listas de sesiones que han sido anunciadas. (Los anuncios pueden ser emitidos para restringir la distribución del anuncio a grupos particulares.) Esta operación devuelve una lista de anuncios que concuerdan con las desiredProperties, especificadas por el consumidor.

El parámetro desiredProperties se puede utilizar para indicar la lista de anuncios. t_AnnouncementSearchProperties identifica las propiedades con las cuales el anuncio debe concordar (véase t_MatchProperties en "Propiedades y listas de propiedades"). Actualmente no se han definidos nombres ni valores de propiedad para t_AnnouncementSearchProperties, por lo que su uso es específico del detallista.

announcements es una lista de anuncios disponibles para el consumidor y que concuerda con desiredProperties. Es una secuencia de estructuras de t_SessionAnnouncement que contienen las propiedades del anuncio, t_AnnouncementProperties. Actualmente no se han definido nombres ni valores de propiedad para t_AnnouncementProperties, por lo que su uso es específico del detallista.

Si el formato del parámetro desiredProperties es erróneo, o proporciona un nombre o valor de propiedad no válido, se debe plantear la excepción e_PropertyError. (Los nombres de propiedades que no son reconocidos pueden ser pasados por alto, si desiredProperties requiere que sólo concuerden algunas o ninguna de las propiedades.)

Si no se dispone de la lista de anuncios, la operación debe plantear e_ListError, con el código de error ListUnavailable.

8.6.4.18 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
);
```

`startService()` comienza una sesión de servicio para el consumidor.

`serviceId` es el identificador de tipo de servicio, que indica el tipo de servicio de la sesión que el consumidor desea comenzar.

`app` es una estructura que contiene información sobre la aplicación, que será utilizada para interactuar con la sesión de servicio. Comprende: nombre de aplicación, versión, número de serie, lista de propiedades, etc. Incluye también una lista de interfaces soportadas por la aplicación, que pueden contener facultativamente referencias a algunas de esas interfaces si están disponibles, una lista de modelos de sesión y conjuntos de características, que incluye también referencias de interfaces, si procede, y una lista de descripciones de interfaces de trenes.

`sessionModelReq` define los modelos de sesión y conjuntos de características que el dominio de consumidor desea tener. Permite que el consumidor pida que algunos, todos o ninguno de los modelos de sesión sean soportados por la sesión.

`uaProperties` es una lista de propiedades que será interpretada por el dominio de detallista antes de que comience la sesión de servicio. No se definen nombres ni valores de propiedad, por lo que su uso es específico del detallista. Su finalidad es también permitir que el consumidor defina algunas preferencias u otras constricciones que desee aplicar a esta sesión de servicio solamente, y que el detallista necesita conocer antes de que la sesión comience. (Estas propiedades pueden afectar la elección de fábrica de servicios para la sesión.)

`ssProperties` es una lista de propiedades que será interpretada por la sesión de servicio, tan pronto como ésta comience (es decir, antes que las referencias a la sesión sean devueltas al dominio de consumidor). No se definen nombres ni valores. Su uso es enteramente específico del servicio, y solo la sesión de servicio está destinada a interpretar las propiedades indicadas. (Este parámetro permite que el dominio/aplicación de consumidor pase información específica de servicio a la sesión de servicio, que no está destinada a ser interpretada por el dominio de detallista.)

`sessionInfo` es una estructura que contiene información que permite al dominio de consumidor hacer referencia a esta sesión utilizando operaciones en estas interfaz. Contiene también información para la parte utilización de la sesión, que incluye las referencias de interfaz para interactuar con la sesión (véase "Información de servicio y de sesión").

Esta operación plantea las siguientes excepciones:

`e_ServiceError`, si el `serviceId` es no válido/desconocido para el detallista, o si una sesión de servicio no puede ser creada.

`e_ApplicationInfoError`, si hay valores desconocidos o no válidos para `t_ApplicationInfo`, o si la aplicación es incompatible con el tipo de servicio comenzado.

e_SessionModelError, si se requieren modelos de sesión y/o conjuntos de características no válidos para la sesión de servicio.

e_StartServiceUAPropertyError, si hay un error en las propiedades para uaProperties. Tiene los mismos códigos de error de propiedades que e_PropertyError. (Para más detalles, véase "Propiedades y listas de propiedades".)

e_StartServiceSSPropertyError, si hay un error en las propiedades de ssProperties. Tiene los mismos códigos de error de propiedades que e_PropertyError.

8.6.4.19 endSession()

```
void endSession (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError
);
```

endSession() termina una sesión de servicio para el consumidor. Se puede utilizar para terminar sesiones en las que consumidor está activo, y sesiones que han sido suspendidas, o el consumidor ha suspendido su participación.

sessionId es el identificador de la sesión que ha de terminar.

Se plantea la excepción e_SessionError si sessionId es no válido, o la sesión rechaza terminar debido al estado de la sesión, o el usuario no tiene permiso.

8.6.4.20 endMyParticipation()

```
void suspendMyParticipation (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError
);
```

endMyParticipation() termina la participación del consumidor en una sesión de servicio. Se puede usar en una sesión en la que el consumidor está activo, o que ha sido suspendida, o el consumidor ha suspendido su participación.

sessionId es el identificador de la sesión para terminar esta participación del usuario.

Se plantea la excepción e_SessionError si el sessionId es no válido, o si la sesión rechaza terminar la participación de este usuario debido al estado de la sesión, o el usuario no tiene permiso.

8.6.4.21 suspendSession()

```
void suspendSession (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError
);
```

suspendSession() suspende una sesión de servicio para el consumidor. Se puede usar para suspender sesiones en las que el consumidor está activo, y sesiones en las que el consumidor ha suspendido ya su participación.

sessionId es el identificador de la sesión que se ha de suspender.

Se plantea la excepción e_SessionError si el sessionId es no válido, o la sesión rechaza suspender debido al estado de esta sesión de usuario, o el usuario no tiene permiso.

8.6.4.22 suspendMyParticipation()

```
void suspendMyParticipation (
       in SPFEEAccessCommonTypes::t_AccessSessionSecretId asSecretId,
       in SPFEECommonTypes::t_SessionId sessionId
) raises (
       SPFEEAccessCommonTypes::e_AccessError,
       e_SessionError
);
```

suspendMyParticipation() suspende la participación del consumidor en una sesión de servicio. Se puede usar en una sesión en la que el consumidor está activo.

sessionId es el identificador de la sesión para suspender la participación de este usuario.

Se plantea la excepción e_SessionError si sessionId es no válido o la sesión rechaza suspender la particular de este usuario debido a su estado de sesión, o el usuario no tiene permiso.

8.6.4.23 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
);
```

resumeSession() reanuda una sesión de servicio. Se puede utilizar en una sesión en la cual el consumidor ha suspendido su participación previamente.

sessionId es el identificador de la sesión que se ha de reanudar

app es una estructura que contiene información sobre la aplicación, que se utilizará para interactuar con la sesión de servicio. Esta aplicación puede ser diferente de la aplicación original del usuario que que estaba siendo utilizada cuando se suspendió la sesión.

sessionInfo es una estructura que contiene información que permite al dominio de consumidor hacer referencia a esta sesión utilizando otras operaciones en esta interfaz. Contiene también información para la parte utilización de la sesión, que incluye referencias de interfaces para interactuar con la sesión (véase "Información de servicio y de sesión").

Se plantea la excepción e_SessionError si el sessionId es no válido, o la sesión rechaza la reanudación debido al estado de la sesión de usuario, o el usuario no tiene permiso.

Se plantea la excepción e_ApplicationInfoError cuando hay valores desconocidos o no válidos para t_ApplicationInfo, o si la aplicación es incompatible con el tipo de servicio que se reanuda.

8.6.4.24 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
        );
```

`resumeMyParticipation()` reanuda la participación del consumidor en una sesión de servicio. Se puede utilizar en una sesión en la cual el consumidor ha suspendido previamente su participación.

`sessionId` es el identificador de la sesión para reanudar la participación del usuario.

`app` es una estructura que contiene información sobre la aplicación, que será utilizada para interactuar con la sesión de servicio. Esta aplicación puede ser diferente de la aplicación original del usuario que se estaba utilizando cuando suspendieron su participación.

`sessionInfo` es una estructura que contiene información que permite al dominio de consumidor hacer referencia a esta sesión utilizando otras operaciones en esta interfaz. Contiene también información para la parte utilización de la sesión, que incluye referencias de interfaces para interactuar con la sesión (véase "Información de servicio y de sesión").

Se plantea la excepción `e_SessionError` si el `sessionId` es no válido, o la sesión rechaza reanudar la participación del usuario debido a su estado de sesión, o no tiene permiso.

Se plantea la excepción `e_ApplicationInfoError` si hay valores desconocidos o no válidos para `t_ApplicationInfo`, o si la aplicación es incompatible con el tipo de servicio que se reanuda.

8.6.4.25 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
        );
```

`joinSessionWithInvitation()` permite al consumidor incorporarse a una sesión de servicio existente para la cual ha recibido una invitación.

`invitationId` es el identificador de la invitación que, mantenido por el detallista, contiene información suficiente que permite al detallista ponerse en contacto con la sesión de servicio y pedir que se permita al consumidor incorporarse a la misma.

`app` es una estructura que contiene información sobre la aplicación, que se utilizará para interactuar con la sesión de servicio.

`sessionInfo` es una estructura que contiene información que permite al dominio de consumidor hacer referencia a esta sesión utilizando otras operaciones en esta interfaz. Contiene también información para la parte utilización de la sesión, que incluye referencias de interfaces para interactuar con la sesión (véase "Información de servicio y de sesión").

Se plantea la excepción `e_SessionError` si la sesión rechaza la incorporación del consumidor.

Se plantea la excepción `e_InvitationError` si el `invitationId` es no válido.

Se plantea la excepción `e_ApplicationInfoError` si hay valores desconocidos o no válidos para `t_ApplicationInfo`, o si la aplicación es incompatible con el tipo de servicio al que se incorpora.

8.6.4.26 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
);
```

`joinSessionWithAnnouncement()` permite al consumidor incorporarse a una sesión de servicio existente que ha descubierto por un anuncio. Es posible obtener anuncios de sesión de varias maneras (que no se describen en Ret-RP), incluso a través de una sesión de servicio especializada.

`announcementId` es el identificador del anuncio que, mantenido por el detallista, contiene información suficiente que permite al detallista ponerse en contacto con la sesión de servicio y pedir que se permita al consumidor incorporarse a la misma.

`app` es una estructura que contiene información sobre la aplicación, que se utilizará para interactuar con la sesión de servicio.

`sessionInfo` es una estructura que contiene información que permite al dominio de consumidor hacer referencia a esta sesión utilizando otras operaciones en esta interfaz. Contiene también información para la parte utilización de la sesión, que incluye referencias de interfaces para interactuar con la sesión (véase "Información de servicio y de sesión").

Se plantea la excepción `e_SessionError` si la sesión rechaza la incorporación del consumidor.

Se plantea la excepción `e_AnnouncementError` si el `announcementId` es no válido.

Se plantea la excepción `e_ApplicationInfoError` si hay valores desconocidos o no válidos para `t_ApplicationInfo`, o si la aplicación es incompatible con el tipo de servicio al que se incorpora.

8.6.4.27 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
);
```

`replyToInvitation()` permite al consumidor responder a una invitación recibida, e informar al detallista su intención de incorporarse o no a la sesión, o indicar una ubicación diferente para buscar al consumidor. (No es posible incorporarse a la sesión mediante esta operación.)

`invitationId` es el identificador de la invitación.

`reply` es una estructura que contiene información sobre la respuesta del consumidor (para más detalles, véase "Invitaciones y anuncios").

Se plantea la excepción `e_InvitationError` si el `invitationId` es no válido.

Se plantea la excepción `e_InvitationReplyError` si hay un error en `reply`.

8.6.5 Interfaz i_RetailerAnonAccess

```
interface i_RetailerAnonAccess
      : i_RetailerAccess
{
      // No operations defined at present
};
```

La interfaz i_RetailerAnonAccess permite a un consumidor anónimo acceder a servicios. El consumidor la utiliza para todas las operaciones dentro de una sesión de acceso con el detallista.

Esta interfaz es devuelta cuando el consumidor ha establecido una sesión de acceso anónima con el detallista, invocando requestAnonAccess() en la interfaz i_RetailerInitial.

Esta interfaz hereda de la interfaz i_RetailerAccess, que actualmente está en blanco. Contendrá operaciones que son compartidas entre la interfaz i_RetailerNamedAccess y esta interfaz. Esto significa que las operaciones ofrecidas por esta interfaz cambiarán en el futuro.

8.6.6 Interfaz i_DiscoverServicesIterator

```
interface i_DiscoverServicesIterator
{
      // Operations defined in the following subsections
};
```

Esta interfaz es devuelta por la operación discoverServices() en la interfaz i_RetailerNamedAccess y se usa para acceder a servicios restantes, que no fueron devueltos por la operación discoverServices().

La operación discoverServices() devuelve una lista de servicios que concordaron con algunas propiedades definidas por el consumidor. Esta interfaz permite al consumidor acceder a los servicios restantes que no fueron devueltos por la invocación a discoverServices(). Esto es necesario porque la lista de servicios que concuerdan con las propiedades puede ser muy larga, e incluye una gran cantidad de información, posiblemente demasiada, para la aplicación que ha de manejar el consumidor.

La operación discoverServices() seguida de posibles múltiples invocaciones de la operación nextN() en esta interfaz, permite al consumidor acceder a todos los servicios que concuerdan con las propiedades, sin tener que recibirlos todos a la vez.

8.6.6.1 maxLeft()

```
void maxLeft (
     out unsigned long n
) raises (
     e_UnknownDiscoverServicesMaxLeft
);
```

maxLeft() indica el número máximo de servicios que serán devueltos a través de esta interfaz. Se puede acceder a estos servicios a través de múltiples invocaciones de la operación nextN().

Se plantea maxLeft() si no es posible que el detallista determine el número máximo de servicios que podrán ser devueltos.

8.6.6.2 nextN()

```
void nextN (
      in unsigned long n,
      out SPFEEAccessCommonTypes::t_ServiceList services,
      out boolean moreLeft
) raises (
      SPFEECommonTypes::e_ListError
);
```

`nextN()` permite al consumidor acceder a los servicios restantes que no fueron devueltos por la operación `discoverServices()` o por invocaciones previas de esta operación. Se puede acceder a estos servicios mediantes múltiples invocaciones de la operación `nextN()`.

El parámetro `n` determina el número máximo de servicios devueltos. La longitud de la lista de servicios no excederá de `n`.

Los servicios restantes serán devueltos como `t_ServiceList services`. Ésta es una `sequence` de estructuras de `t_ServiceInfo`, que contiene `t_ServiceId`, `t_UserServiceName` (nombre del consumidor para el servicio) y una `sequence` de propiedades de servicio, `t_ServiceProperties`.

El parámetro `moreLeft` es un `booleano` para informar al consumidor si hay servicios restantes, después de esta invocación de `nextN()`.

8.6.6.3 destroy()

```
void destroy ();
```

La operación `destroy()` se usa para informar al detallista que el consumidor ha terminado con la interfaz `i_DiscoverServicesIterator`. El consumidor puede invocarla en cualquier momento (es decir, el consumidor no tiene que haber extraído los servicios antes de destruir la interfaz). Una vez devuelta, el consumidor no podrá utilizar de nuevo su referencia a la interfaz `i_DiscoverServicesIterator`.

8.7 Gestión de abono

Esta subcláusula esta dedicada a la descripción de las interfaces ofrecidas por el detallista al consumidor, para soportar la funcionalidad relacionada con abono. Existen dos tipos de interfaces, correspondientes a dos tipos de consumidores: una accedida por el usuario anónimo, que le permite convertirse en abonado, y otra accedida por el abonado y que le permite gestionar toda la información relacionada con su abono. Las interacciones relacionadas con abono se efectúan de manera genérica (independientes del servicio). Obsérvese que los servicios auxiliares puede implementar una facilidad de abono en línea. Como esto es específico del servicio, se considera fuera del alcance de este Suplemento. Se accede a las interfaces de abono en el contexto de la sesión de acceso, para consultar la lista de servicios con los que está asociado el consumidor/abonado/usuario y los correspondientes perfiles de servicios, y para modificar los datos de abono.

La principal funcionalidad ofrecida a un usuario anónimo es:

– consulta de la lista de servicios (es decir, los que están disponibles a través de este detallista),
– creación de un nuevo abonado.

La principal funcionalidad ofrecida a un abonado es:

– consulta de la lista de servicios, a saber, los que están disponibles a través de este detallista, o los abonados,
– creación, modificación, supresión e indagación de información relacionada con el abonado (usuarios de extremo asociados, grupos de usuarios de extremo, etc.),
– creación, modificación, supresión e indagación de contratos de servicio (definición de perfiles de servicios abonados),
– consulta del perfil de servicio (SAGServiceProfile) de un usuario específico (o terminal o NAP).

8.7.1 Definiciones de tipos de gestión de abono

En esta subcláusula se incluye la definición IDL de la información requerida para tratar abonos, abonados y usuarios de extremo en un dominio de proveedor. Esto permitirá comprender con más claridad las descripciones de las interfaces.

La figura 8-4 representa principalmente la relación entre un servicio y un abonado, descrita de acuerdo con el número de perfiles de servicio (plantillas de servicio, perfil de abono y perfil de servicio SAG).

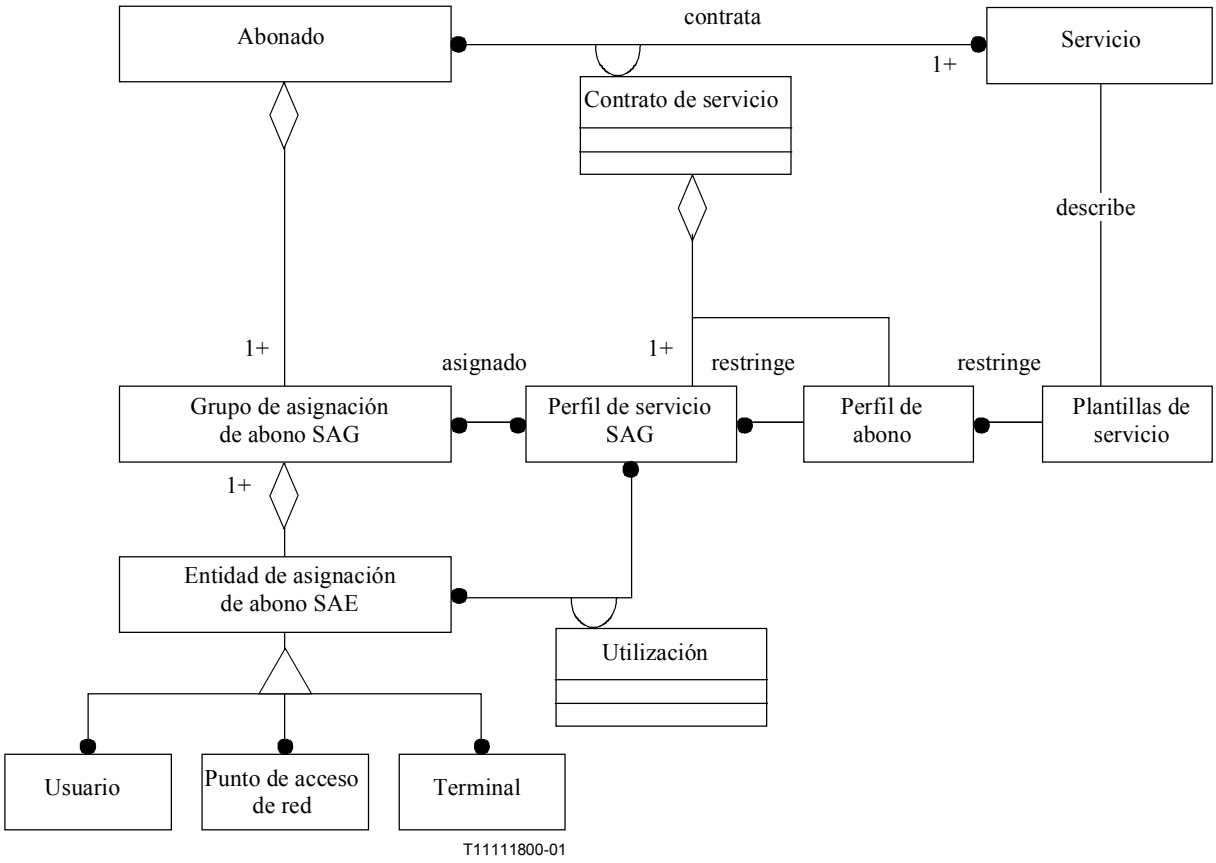


Figura 8-4 – Modelo de información de gestión de abono

Un abonado contrata varios servicios (por lo menos uno, para ser considerado como tal). La información asociada con un abonado es:

```
struct t_Subscriber {
   t_AccountNumber                    accountNumber;
   SPFEECommonTypes::t_UserId         subscriberName;
   t_Person                           identificationInfo;
   t_Person                           billingContactPoint;
   string                             RatePlan;
   any                                paymentRecord;
   any                                credit;
};
```

El `accountNumber` es generado por el detallista y es único en su dominio. Se utiliza dentro del dominio de detallista para identificar al abonado y probablemente también las facturas. El

subscriberName es el nombre por el que el abonado desea ser denominado[41], y que puede ser más familiar para el usuario que el accountNumber. Habrá una correspondencia de uno a uno entre estos dos identificadores. El campo identificationInfo almacena información de nombre de abonado, dirección, etc. El billingContactPoint mantiene la información requerida para enviar las facturas. El paymentRecord mantiene información sobre las últimas facturas pagadas, para verificar el estado contable. El campo credit almacena información sobre depósitos, créditos autorizados al abonado, etc.

El contrato de servicio acordado define las condiciones de la prestación de servicio para cada uno de los abonos de servicio. Se define como:

```
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;
};
```

El serviceId y el accountNumber, identifican juntos, de manera única un contrato de servicio. Los perfiles son la parte principal del contrato de servicio. Otros campos proporcionan información adicional sobre el contrato (fecha de comienzo, fecha de comienzo solicitada, solicitante, punto de contacto técnico, etc.).

Una plantilla de servicio describe las características del servicio accesibles a través del detallista:

```
struct t_ServiceTemplate {
   SPFEEAccessCommonTypes::t_ServiceId           serviceInstanceId;
   SPFEEAccessCommonTypes::t_UserServiceName     serviceInstanceName;
   t_ServiceIdList                               requiredServices;
   t_ServiceDescription                          serviceDescription;
};
```

La descripción de servicio contiene las características de un tipo de servicio genérico. Se reutiliza en la plantilla de servicio para describir las características de un servicio específico (prestación particular de un tipo de servicio) y en los perfiles de servicio para representar las características del servicio contratado por el abonado (para todo el conjunto de usuarios asociados o para un grupo de ellos).

```
struct t_ServiceDescription {
   SPFEEAccessCommonTypes::t_ServiceId           serviceTypeId;
   SPFEEAccessCommonTypes::t_UserServiceName     serviceTypeName;
   t_ParameterList                               serviceCommonParams;
   t_ParameterList                               serviceSpecificParams;
};
```

El parámetro List consiste en una secuencia de triples compuesta de nombre de parámetro, configurabilidad de parámetro y valor de parámetro:

```
typedef string  t_ParameterName;
enum t_ParameterConfigurability { FIXED_BY_PROVIDER,
   CONFIGURABLE_BY_SUBSCRIBER, CUSTOMIZABLE_BY_USER };
```

[41] Se pudiera utilizar para generar identificadores de usuario. Por ejemplo, al usuario X abonado con el detallista A se pudiera dar un identificador semejante a X@A.

```
typedef any     t_ParameterValue;

struct t_Parameter {
   t_ParameterName                    name;
   t_ParameterConfigurability         configurability;
   t_ParameterValue                   value;
};

typedef sequence<t_Parameter> t_ParameterList;
```

El detallista puede dar al abonado la opción de seleccionar parámetros de servicio específicos para aplicarlos a todos sus entidades asociadas[42], perfil de abonado, o a un grupo de ellos, perfil de servicio SAG, lo que reduce las alternativas (restringe la asociación indicada en la figura 8-4) de la plantilla de servicio. Estos perfiles son la parte principal del contrato de servicio.

```
typedef string t_ServiceProfileId;
struct t_ServiceProfile {
   t_ServiceProfileId             spId;
   t_ServiceDescription           serviceDescription;
};
typedef t_ServiceProfile t_SagServiceProfile;
typedef t_ServiceProfile t_SubscriptionProfile;
```

Un conjunto de entidades, usuarios, terminales o puntos de acceso de red (NAP), pueden estar asociados a un abonado, y cabe denominarlos entidades de asignación de abono (SAE, *subscription assignment entities*).

```
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;
};
```

Es posible que el abonado no desee dar a todas estas entidades las mismas características de servicio (o privilegios), por lo que puede agruparlas en un conjunto de grupos de asignación de abono (SAG, *subscription assignment groups*):

```
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;
};
```

El abonado puede asignar perfiles de servicio particulares (perfiles de servicio SAG) a cada grupo. El principal motivo para usar esta agrupación es facilitar el proceso de abono (asignación de perfiles

[42] Usuarios, terminales o puntos de acceso de red.

a usuarios) en los dominios de abonado donde los usuarios de extremo están clasificados naturalmente en categorías, zonas de organización o geográficas, etc., que requieren los mismos privilegios de utilización de los servicios. La única restricción es que a cada SAE se debe asignar solamente un perfil de servicio SAG para cada servicio.

Para cada abonado se crea un SAG por defecto con SAGId (0). Cada SAE se asigna siempre a este SAG, incluso si tiene asignado otro SAG particular. Si se suprime un usuario de un SAG, seguirá asociado a este SAG por defecto. El SAG por defecto no puede estar asociado a perfiles de servicio, por lo que no se puede asignar usuarios explícitamente a ste SAG (son asignados implícitamente a su creación).

Es posible también asignar y suprimir las SAEs a/de perfiles de servicio. Esto es especialmente interesante en organizaciones de abonados pequeñas (como cliente residenciales), cuando la definición de grupos de usuarios no es estrictamente necesaria y no facilita al abonado la gestión del abono. Además, proporciona una gran flexibilidad en la asignación de perfil servicio, dado que se puede discriminar a algunos usuarios en un grupo en cuanto al acceso a un servicio específico, sin tener que suprimirlos del grupo o definir un nuevo grupo.

Una estructura como t_UsagePermit puede facilitar la definición de estas retricciones:

```
enum t_UsagePermitFlag {USAGE_ALLOWED, USAGE_DISALLOWED};
struct t_UsagePermit {
   t_entityId                  entityId;
   t_ServiceProfileId          serviceProfileId;
   t_UsagePermitFlag           flag;
};
```

A continuación se describen las interfaces ofrecidas por el detallista al consumidor para soportar la funcionalidad de abono. Hay un conjunto de interfaces específicas de servicio ofrecidas a través del punto de referencia Ret para la gestión en línea de los abonos de servicio. Hay interfaces diferentes para cada tipo de usuario (abonados u operadores detallistas).

8.7.2 i_SubscriberSubscriptionMgmt

```
// module SPFEERetSubscriberSubscriptionMgmt
interface i_SubscriberSubscriptionMgmt
{
};
```

Esta interfaz está dedicada a abonados. Proporciona operaciones para abonarse con el detallista, contratar servicios y definir información de contratos de abonado y de servicio. Permite la cancelación y modificación de contratos de abono y de servicio y la indagación de toda información relacionada con abonados.

Las operaciones para solicitar contratos de servicio, abonos y cancelaciones son:

– **`listServices`**() – Devuelve la lista de servicios proporcionados por el detallista.

– **`subscribe`**() – Permite crear un contrato de abono con el detallista. Tiene como parámetros de entrada la información de abonado y una lista de servicios que desea el abonado. Devuelve un identificador de abonado y una lista de identificadores de contrato de servicios, que se utilizarán a continuación para hacer referencia a contratos de servicio específicos.

– **`unsubscribe`**() – Permite suprimir un contrato de servicio (una lista de contratos de servicio) o toda la relación con el detallista.

– **`contractService`**() – Efectúa el abono a un servicio y devuelve una referencia de interfaz donde el abonado puede definir el contrato de servicio.

– **`listSubscribedServices`**() – Devuelve la lista de los servicios contratados (sólo los identificadores) e identificadores de contratos relacionados con servicios. El identificador de abonado (número de cuenta) se indica como un parámetro de entrada.

Las operaciones que tratan información de abonado son:

– **`listSAEs`**() – Devuelve la lista de entidades asociadas con el abonado. Si se especifica un identificador (o lista de identificadores) de SAG, sólo devuelve los usuarios asignados a ese (o esos) SAG.

– **`listSAGs`**() – Devuelve la lista de (id) de SAG para ese abonado.

– **`getSubscriberInfo`**() – Devuelve la información sobre el abonado.

– **`createSAEs`**() – Crea las entidades especificadas como un parámetro que devuelve un identificador para cada uno de ellas.

– **`deleteSAEs`**() – Suprime las entidades especificadas como un parámetro. Suprime cualquiera asignación existente a los SAG que estas entidades puedan tener.

– **`createSAGs`**() – Crea uno o varios SAG. Se puede especificar una lista de entidades para cada SAG. Se devuelve un identificador de SAG para facilitar la gestión.

– **`assignSAEs`**() – Asigna una lista de entidades a un SAG.

– **`removeSAEs`**() – Suprime una lista de entidades de un SAG.

– **`setSubscriberInfo`**() – Modifica la información sobre el abonado.

Las operaciones para definir y modificar contratos de servicio e indagar sobre ellos son:

– **`getServiceTemplate`**() – Devuelve la plantilla para la definición del perfil del servicio especificado.

– **`defineServiceContract`**() – Permite definir el contrato para un servicio especificado. Este contrato incluye, además de otra información contractual, el conjunto de perfiles de de servicio que componen el contrato de servicio, a saber, el perfil de abono (aplicable a todos los usuarios) y el conjunto de perfiles de servicio de SAG (cada uno aplicable a un SAG y coherente con el perfil de abono). Devuelve a lista de identificadores de perfil de SAG para facilitar su referencia futura. Se utiliza para definir y redefinir (modificar) contratos de servicio.

– **`defineServiceProfiles`**() – Permite definir un conjunto de perfiles de servicio para un contrato de servicio, a saber, el perfil de abono y el conjunto de perfiles de servicio de SAG. Devuelve a lista de identificadores de perfil de SAG para facilitar su referencia futura. Se utiliza para definir y redefinir (modificar) contratos de servicio.

– **`deleteServiceProfiles`**() – Suprime un perfil de servicio.

– **`getServiceContractInfo`**() – Devuelve la información relacionada con el contrato de servicio cuyo identificador es pasado como un parámetro. Si se especifica una lista de identificadores de perfiles de SAG, devuelve el conjunto de perfiles de servicio SAG asociados.

Las operaciones para autorización y activación de perfiles de servicio son:

– **`assignServiceProfile`**() – Asocia una lista de las SAE y SAG con un SAGServiceProfile. Si el perfil de servicio está activo, las SAE (las indicadas explícitamente y las incluidas en los SAG) podrán utilizar el servicio. Estos componentes de acceso de las SAE serán notificados y el perfil de servicio de SAG se pondrá a disposición de los mismos. A partir de este perfil, la SAE podrá personalizar su propio perfil de servicio de usuario utilizando el servicio de gestión de perfil de usuario.

– **`removeServiceProfile`**() – Disocia una lista de SAE y SAG de un SAGServiceProfile. Las SAE especificadas (individualmente o dentro de un SAG) ya no podrán utilizar el servicio, a menos que sean asociadas con otro con otro perfil de usuario activo.

– **activateServiceProfiles**() – Activa una lista de perfiles de servicio de SAG poniéndolos a disposición para ser usados. Sólo las SAE y los SAGs asignados a un perfil de servicio activo pueden utilizar el servicio.

– **deactivateServiceProfiles**() – Desactiva una lista de perfiles de servicio de SAG. Los usuarios (o SAE) asignados a estos perfiles de servicio no podrán utilizar el servicio.

8.7.3 i_RetailerSubscriptionMgmt

```
// module SPFEERetRetailerSubscriptionMgmt
interface i_RetailerSubscriptionMgmt
{
};
```

Esta interfaz proporciona las capacidades para abonar clientes, añadir, modificar, cancelar contratos de servicios e indagar sobre ellos, así como añadir, suprimir, modificar información de abonado e indagar sobre ella. En general, proporciona acceso a toda la base de datos de abono. Es un superconjunto de la interfaz anterior, dedicado a un consumidor del tipo operador detallista

Las operaciones adicionales son:

– **listSubscribers**() – Devuelve la lista de los abonados (identificadores). Si se especifica un Id de servicio, devuelve la lista de abonados para ese servicio.

– **listServiceContracts**() – Devuelve la lista de los contratos de servicio (identificadores). Si se especifica un Id de servicio, devuelve la lista de los contratos de servicio para ese servicio. Si se especifica un abonado, actúa como la `listSubscribedServices` en la interfaz `i_SubscriberInfoMgmt` para ese abonado particular.

– **listUsers**() – Devuelve la lista de usuarios (identificadores) para un servicio especificado.

9 Especificaciones IDL completas

9.1 Definiciones IDL comunes

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 Definiciones IDL generales para el usuario y el proveedor

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 Definiciones IDL para el punto 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 Especificaciones IDL para la subscripción del punto 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
```

SERIES DE RECOMENDACIONES DEL UIT-T

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

Ginebra, 2001