Proyecto Gatekeeper H.323

Martín Morales, Marcos Perdomo, Carlos Quiñones

4 de diciembre de 2004

Índice general

Capítulo 1

Alcance y objetivos

1.1. Introducción

En la actualidad se está produciendo un cambio tecnológico desde las redes de circuitos conmutados hacia las redes de paquetes cuyo resultado será la convergencia de las redes y servicios existentes en una única red. Gran cantidad de investigadores trabajan en esta área desarrollando protocolos para las redes del futuro. H.323 constituye uno de los tantos protocolos surgidos para brindar servicios multimedias sobre redes de paquetes, siendo el más utilizado en el presente. Teniendo en cuenta estas premisas, este grupo de trabajo se puso como objetivo el estudio de la problemática en cuestión y la implementación de uno de los elementos claves en el protocolo H.323: el *gatekeeper*. El presente documento describe los aspectos más importantes de esta nueva tecnología y detalla el trabajo realizado durante el transcurso del proyecto. Planteados los objetivos, se comienza brindando cierta aproximación a la teoría de VoIP (Voice Over IP) y sus protocolos, haciendo hincapié en la norma H.323. Luego se detallan las herramientas utilizadas para la implementación del *gatekeeper* y se brinda una breve reseña de la evolución de la implementación. Para finalizar, se detallan las características tecnológicas de la implementación obtenida.

1.2. Objetivos

Teniendo en cuenta el trabajo realizado por un grupo anterior de proyecto, que desarrolló terminales de VoIP tomando como base la norma H.323, se planteó la necesidad de sumar a dicho trabajo una entidad centralizada en la red que proveyera a los terminales servicios tales como resolución de direcciones. En una primera instancia, se pensó en crear una entidad que actuaría como una especie de centralita a nivel de VoIP brindando algunos de los servicios habituales en las centralitas telefónicas tradicionales. Un primer estudio de la problemática de VoIP y de las posibles soluciones al problema sugirió la posibilidad de implementar un *gatekeeper* H.323 dotándolo de ciertas funciones auxiliares.

Por lo tanto, los objetivos del proyecto se especificaron de la siguiente manera: en primer lugar desarrollar vía software un *gatekeeper* H323 con

las funciones básicas (y obligatorias) que la norma H.323 le confiere; como segundo objetivo implementar algunas funciones adicionales, como pueden ser: registro de llamadas, desvío de llamadas, transferencia de llamadas, etc.

La realización de dichos objetivos implicaba primeramente un estudio detallado de los protocolos que se implementarían: RAS, Q.931/H.225, H245, H.450, etc, y las herramientas disponibles que se pudieran utilizar. Para el desarrollo del software se buscó la posibilidad de utilizar bloques constructivos, APIs, bibliotecas de código, etc., disponibles como "código libre", para la implementación del stack de protocolos.

A consecuencia de lo anterior surgió la necesidad del estudio del lenguaje de programación C++, así como de los ambientes de desarrollo a utilizar como por ejemplo, Visual C++ del paquete Visual Studio 6.0 para Microsoft Windows, así como las herramientas disponibles en Linux: Kdevelop, Make, GCC, etc.

Capítulo 2

Voz sobre IP y estandarizaciones

2.1. Servicios multimedia sobre redes de paquetes

Cada día más las redes de paquetes están siendo utilizadas como alternativa a las redes de circuitos de la telefonía tradicional para brindar servicios multimedia. Esta tendencia se basa en factores tecnológicos, comerciales, e históricos. A nivel tecnológico se destaca el mejor aprovechamiento de los recursos por parte de las redes de paquetes mediante el uso de la multiplexación estadística. Sin embargo, las redes de paquetes, en sus orígenes, no fueron pensadas para brindar servicios de tiempo real, y los protocolos que utilizan no están optimizados para ello. Las ventajas en cuanto al mejor uso de los recursos se contraponen con las limitaciones propias de estas redes y sus protocolos (retardo variable, congestión, etc). Durante muchos años se ha intentado obtener la funcionalidad, fiabilidad, y calidad de servicio propias de las redes de circuitos conmutados para las redes de paquetes, y ese objetivo está bastante cercano en la actualidad.

Otros factores que determinan esta tendencia es el crecimiento explosivo de la Internet y el uso creciente del protocolo IP en las distintas redes de paquetes. El desarrollo de la Internet determinó el surgimiento de una nueva red capaz de brindar servicios que antes sólo podían ser dados con las redes de circuitos de las compañías telefónicas. La desregulación del mercado de las telecomunicaciones a nivel mundial y la competencia surgida entre las compañías telefónicas y de datos, cada una tratando de hacer uso de sus redes para brindar los servicios que históricamente ofrecían sus competidoras, ayudó en este sentido. Durante años los analistas han pronosticado la convergencia de los distintos tipos de redes (redes de voz, de datos), y en años recientes, IP ha emergido como la plataforma unificadora.

Por todas estas razones la telefonía IP tiene una gran importancia en las comunicaciones actuales. En este documento se utilizarán los términos “telefonía IP” y “VoIP” indistintamente para definir las distintas formas de transmitir voz, video, datos y cualquier otro tipo de medios con

requerimientos de tiempo real a través de redes de paquetes conmutados basadas en IP. Sin embargo, cabe aclarar que existen otras concepciones respecto de esta terminología utilizándose el término "VoIP" para referirse a las tecnologías necesarias para transmitir los medios en tiempo real sobre IP, mientras que "Telefonía IP" se utiliza en un sentido más amplio, incluyendo funcionalidades de la telefonía tradicional como por ejemplo: desvíos, transferencias, etc.

2.2. Estándares

En los últimos años distintas organizaciones han emitido estándares para organizar el crecimiento de la telefonía IP. El primero y más ampliamente difundido fue H.323, auspiciado por la Unión Internacional de Telecomunicaciones (UIT) o su sigla en inglés ITU. Luego han surgido otros, con éxito variable, que han dispuesto una gran variedad de protocolos que se pueden usar con estos servicios: Session Initiation Protocol (SIP) patrocinado por el IETF; H.248 o MEGACO, por la IETF y la ITU, entre otros.

Las diferencias entre ellos, básicamente, se fundamentan en las distintas concepciones de las organizaciones donde surgieron. La ITU, con su pasado ligado a la telefonía tradicional basada en circuitos conmutados, resalta la necesidad de especificar lo más posible cualquier aspecto del problema, con normas bastante completas y detalladas. La comunidad Internet, nucleada en el Internet Engineering Task Force (IETF), promueve la simplicidad a cambio, quizás, de falta de completitud y universalidad, relativizando la compatibilidad hacia atrás con las soluciones telefónicas actuales.

De todas maneras, los dos protocolos más ampliamente difundidos comparten la misma clasificación en cuanto al modelo que plantean. Los dos especifican un modelo distribuido con la inteligencia repartida entre los distintos elementos de la red, más precisamente entre los terminales, pudiéndose agregar elementos opcionales que ofrecen otras funciones. Se podría decir que no existe una marcada jerarquía vertical entre las distintas entidades que conforman la red, pudiéndose establecer comunicaciones entre dos terminales con su sola presencia. Por el contrario, Megaco plantea un modelo cliente-servidor con una jerarquización de los elementos de la red. Megaco en sí, no surge como una solución distinta al mismo problema, sino como un complemento para H.323 y SIP.

Dada la variedad de protocolos es de destacar los avances que se están realizando en cuanto a la interconexión de éstos, ya que se asume que coexistirán durante los próximos años.

2.2.1. ITU-T H.323

La ITU comenzó a trabajar en definir los protocolos de señalización de VoIP en mayo de 1995. En diciembre de 1996 el Study Group 16 emitió H.323 v1 referido como un "estándar para videoconferencias en tiempo real sobre LAN sin calidad de servicio garantizada". Si bien surgió para redes LAN se puede considerar un estándar para comunicaciones multimedia basadas en redes de paquetes, incluyendo LAN, WAN y la Inter-

net. H.323 constituye un conjunto de normas (H.323, H.225, H.235, H.245, H.450, H.460) y especifica los componentes, protocolos y procedimientos que deben ser usados en el establecimiento de servicios multimedia (comunicaciones de tiempo real de audio, video y datos) sobre redes de paquetes. H.323 está basado en los protocolos multimedia de la ITU que lo precedieron (H221 para ISDN, H223 para B-ISDN). Algunos protocolos como H.225 y H.245 fueron derivados de H.320, más precisamente de H.221 y H.242; otros, ya existentes, fueron reusados, como RTP y RTCP. El soporte de la voz es obligatorio mientras que los datos y el video son opcionales. Desde el comienzo se propició un rápido proceso de estandarización teniendo en cuenta la interconexión con la PSTN. Su diseño está basado en forma primaria en IP, pero puede operar en cualquier tipo de redes de paquetes conmutados.

En cuanto a la arquitectura es un protocolo *peer-to-peer* que distribuye la inteligencia entre los distintos componentes de la red, siendo los terminales la parte fundamental, y la única necesaria, para el funcionamiento del sistema.

Las nuevas versiones que han surgido rescatan las nuevas tendencias en esta área. La evolución ha ido señalando una simplificación de ciertos aspectos del protocolo tratando de emular ciertas características en las que SIP se considera favorable. Se pueden destacar la adopción de mecanismos para hacer más rápida la conexión (*Fast Start*), y la incorporación de nuevas facilidades de abonado para añadir a la telefonía IP la mayoría de las características presentes en la telefonía tradicional. La versión actual (versión 5 en vigencia desde Junio del 2003) propone pocas modificaciones respecto a su predecesora, ámpliamente implementada en aplicaciones comerciales. Sus promotores dicen que ya es un protocolo maduro e intentan sólo agregar o mejorar aspectos accesorios a éste, sin grandes modificaciones en cuanto a la base del protocolo.

H.323 ha sido hasta ahora el líder en cuanto a utilización alrededor del mundo, debido principalmente a que ha sido el primero, y a que se considera maduro para desarrollar aplicaciones comerciales.

2.2.2. SIP - Session Initiation Protocol

SIP es un protocolo de señalización que sirve para crear, modificar y terminar sesiones multimedia con uno o más participantes. Es desarrollado en el Multiparty Multimedia Session Control Working Group (MMUSIC WG) de la IETF y fue propuesto como estándar en Marzo de 1999 en la RFC 2543. Actualmente se rige por la RFC 3261 de Junio del 2002 [SIP]. Entre sus funciones se destacan: localización de servicios y participantes, invitación a sesiones, y negociación de los parámetros de la sesión (con la ayuda de otros protocolos como *Session Description Protocol*, SDP). Al igual que H.323 es un protocolo *peer-to-peer* donde los terminales inician sesiones con la ayuda de otros dispositivos opcionales (*SIP Gateway, Registrar Server, Proxy Server, Redirect Server*). Las sesiones SIP van desde llamadas telefónicas, hasta distribución *broadcast* y conferencias multimedia. Los participantes pueden comunicarse vía *multicast*, *unicast,* o por una combinación de éstas, en ambientes IPV4 o IPV6.

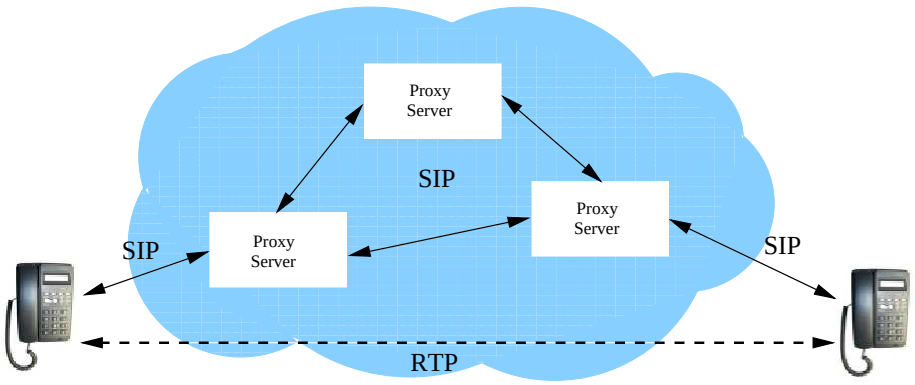

Figura 2.1: Red SIP

SIP se basa en el intercambio entre las entidades SIP de un reducido número de mensajes de texto mediante transacciones separadas . Utiliza muchos de los headers, códigos de error, técnicas de codificación y autenticación de HTTP, y aprovecha la experiencia lograda por la comunidad Internet con este protocolo. Tanto el *Registrar Server*, el *Proxy Server*, como el *Redirect Server* son entidades lógicas y pueden coexistir físicamente en la misma unidad. Los terminales se registran con el *Registrar Server* y para establecer una sesión, generalmente, utilizan un *Proxy Server* cuya principal función es la de ruteo de las llamadas. Para conocer la dirección de transporte del terminal llamado se utiliza un servicio de localización implementado en el *Proxy*, en el *Registrar,* u otra entidad separada, ver figura 2.2. Puede suceder que el *Proxy* no quiera participar en algunas sesiones, y en ese caso, se comporta como un *Redirect Server* contestando la invitación con una redirección hacia otro terminal o *Proxy Server*. Esto permite repartir la carga entre varios *Proxies,* o establecer dominios atendidos por diferentes servidores. Por ejemplo: un servidor sólo funciona como *Proxy* de las sesiones de su dominio, mientras que es un *Redirect Server* para las sesiones con otros dominios (figura 2.3).

En cuanto a la función que le compete a la entidades SIP durante una llamada, se pueden clasificar en UAC (*User Agent Client*) y UAS (*User Agent Server*). El primer tipo genera requerimientos SIP (inicio de sesión, etc), mientras que el segundo implementa una respuesta a los primeros (acepta, rechaza o redirecciona el requerimiento). Por ejemplo: un terminal se comporta como un UAC al iniciar una llamada y como un UAS al recibirla, de la misma manera un *Proxy Server* actúa como UAC y UAS al rutear las llamadas.

La sesión se especifica en dos niveles: SIP contiene las direcciones de los participantes y los elementos de control, mientras que las descripciones de los trenes de medios son definidas por otro protocolo, por ejemplo: SDP. SDP, mas que un protocolo, es un formato de mensajes de texto para describir trenes de medios que puede ser incluido en el cuerpo de los mensajes SIP. Dado que el cuerpo del mensaje es transparente para SIP, cualquier tipo de descriptor de sesión puede ser utilizado, de modo que las sesiones SIP no se

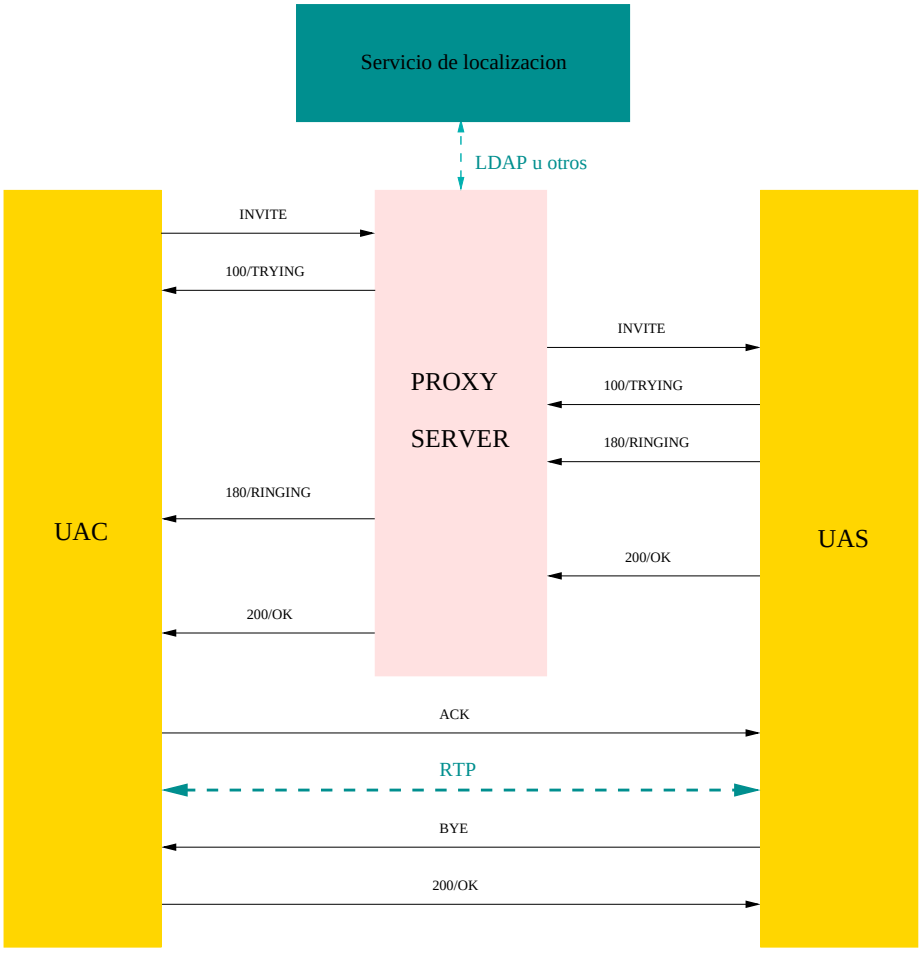

Figura 2.2: Sesión SIP entre dos terminales con participación de Proxy Server

restringen a llamadas telefónicas o conferencias, y pueden incluir difusión de información, sesiones broadcast, etc.

De forma similar a H.323, utiliza RTP y RCTP para la comunicación de medios entre los terminales.

Fue creado pensando en su utilización en toda la Internet debido a la organización en la que surgió (la comunidad Internet), y no tuvo como objetivo la compatibilidad con las soluciones telefónicas existentes. En los últimos tiempos ha tenido un crecimiento vertiginoso y ha sido elegido por *3GPP* para la telefonía móvil de tercera generación y por *Microsoft* en su programa *Messenger*.

2.2.3. MGCP y H.248 MEGACO

El *gateway*, con su función de interconexión con la PSTN, es uno de los elementos más importantes en la telefonía IP. Por razones de escalabilidad

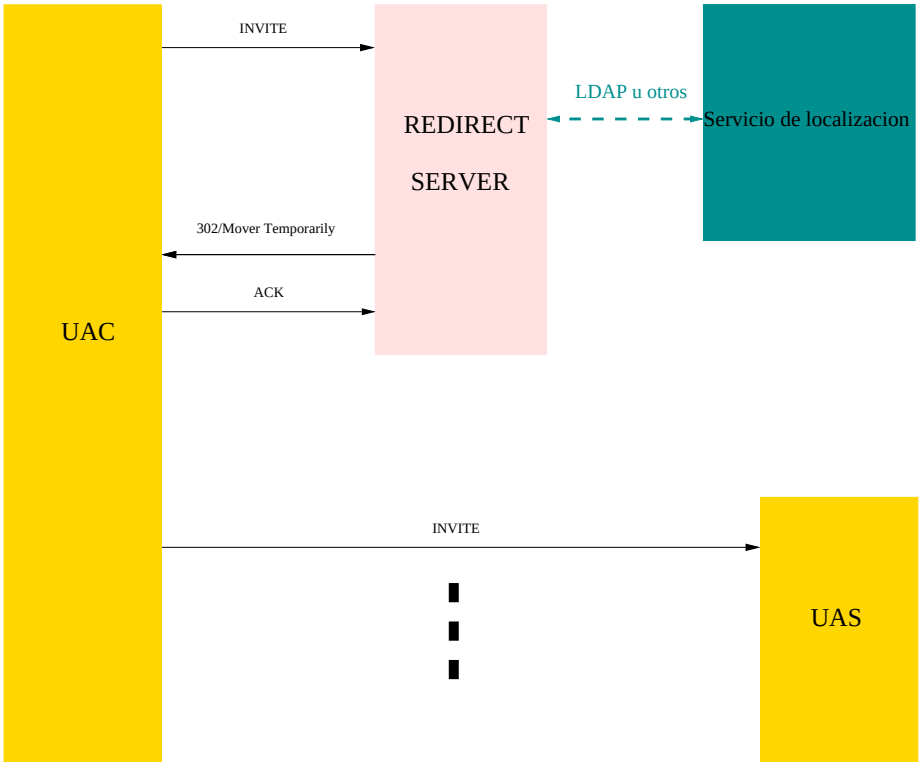

Figura 2.3: Intercambio de mensajes en una sesión SIP con Redirect Server

se ha hecho necesaria su descomposición en elementos e interfaces normalizadas. MGCP y MEGACO son un ejemplo en esta dirección.

MGCP 1.0 fue definido por la IETF en la RFC 2705 de Octubre de 1999. Está diseñado como un protocolo interno dentro de un sistema distribuido, que hacia el exterior es visto como un único *gateway VoIP*. H.248 o MEGACO es posterior a MGCP (aunque basado en éste) y es un estándar auspiciado tanto por la ITU como por la IETF. ITU SG16 (H.248) y IETF WG MEGACO acordaron emitir un único documento desde de Junio de 1999. Actualmente están vigentes las recomendaciones H.248v1 de la ITU y la RFC3015 (desde Noviembre del 2000) de la IETF, las cuales comparten el mismo texto.

MEGACO está basado en MGCP pero con más funcionalidades que su antecesor, y si bien MGCP tiene más implementaciones comerciales, esto se debe a su antigüedad. Además, dado que MGCP es sólo una RFC informativa y no será modificada ni corregida, se espera que la industria adopte a MEGACO como la opción más usada para la implementación de *gateways* distribuidos. Teniendo en cuenta la analogía, de acá en mas se hablará sólo de MEGACO, y cuando sea necesario, se hará explícita la diferencia.

Un *gateway* distribuido tiene los siguientes componentes: el *Media Gateway Controller* (MGC) que se encarga de la señalización; y los *Media Gateways* (MG) controlados por el MGC, que se ocupan de los trenes de medios

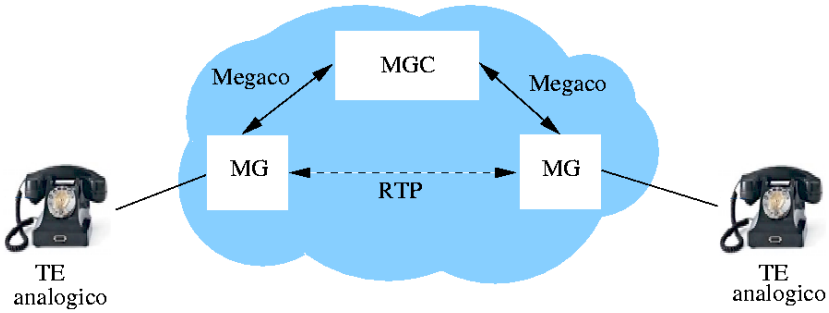

Figura 2.4: MEGACO con terminales analógicos

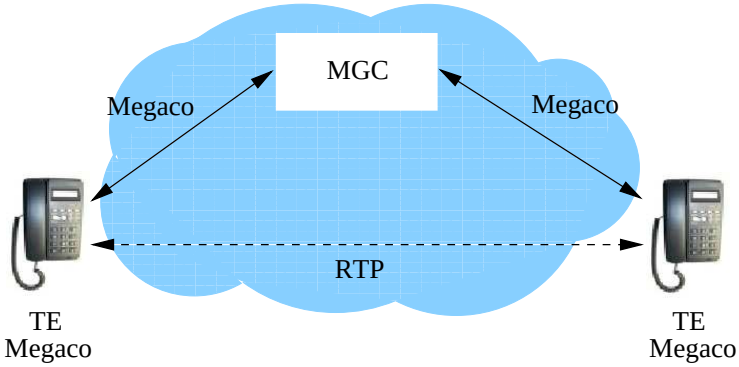

Figura 2.5: MEGACO con terminales digitales

(por ejemplo, algunos MG pueden ser: IP *phone*, VoIP *gateway*, DSLAM, *router*, etc). Ver figuras 2.4 y 2.5. Específicamente, MEGACO es el protocolo que se utiliza para comunicar estos dos tipos de entidades. Este modelo refleja un paralelismo con la arquitectura de la PSTN al crear una entidad de señalización que puede actuar como pareja de las entidades SS7. Se dice que MEGACO crea la "central telefónica IP", ver figura 2.6.

Esta división permite centralizar la inteligencia y desplazarla de los terminales volviendo al concepto de terminal tonto tan extendido en la telefonía tradicional, de allí la utilización del modelo cliente-servidor. Los terminales proveen la interacción con el usuario y cada acción del MG está supeditada al controlador. A modo de ejemplo, para cursar una llamada los pasos serían los siguientes. Un MG (un terminal) detecta que se ha levantado el tubo y se lo comunica al MGC. Éste ordena al MG que ponga tono en la línea y que espere los tonos DTMF para determinar a qué número se quiere llamar. En base a esto, el MGC determina cómo rutear la llamada usando un protocolo *inter-gateway* (SIP, H.323, Q.BICC). En caso de que se llame a un terminal bajo su control no es necesario el uso de estos protocolos. Luego, el MGC encargado instruye a su MG a dar el timbre de llamada y en caso de que se detecte que el usuario contesta (levanta el tubo) los MGC respectivos indican a los MG que establezcan los trenes de medios. Como se

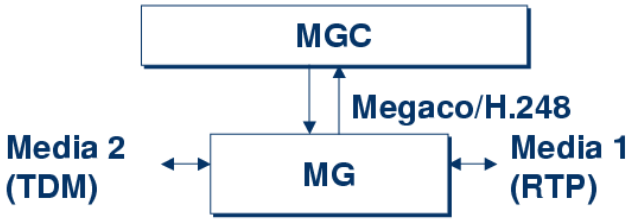

Figura 2.6: MEGACO

ve, este protocolo implica un control total sobre el terminal y concentra la inteligencia en el controlador, no el terminal.

Con MEGACO se tiene control de las conexiones, del terminal, pero no se tiene la posibilidad del ruteo de las llamadas lo cual hace indispensable su utilización con otros protocolos como H.323 o SIP, si esto es lo que se pretende. De todas maneras puede ser una solución completa en ciertas ámbitos limitados, a saber: centralitas privadas, sistemas de intercomunicación particulares, etc.

2.2.4. Comparación entre los distintos estándares

Dadas las características generales de los protocolos vistos hasta ahora se podría pensar en la complementariedad de muchos de ellos. Esta idea es cierta según el área de aplicación y la funcionalidad que se pretenda del protocolo a utilizar. Si bien no es lógico pensar en un sistema que utilice sólo MEGACO para dar servicio a una gran cantidad de terminales con un variado plan de numeración, se podría dar una solución parcial para un pequeño grupo de terminales utilizando la estructura de MEGACO, y algún otro protocolo, si se pretende la interconexión de éstos con otras redes. De la misma forma se puede utilizar H.323 y SIP sin tener que depender de MEGACO.

H.323 y SIP son protocolos completos que no necesitan de otros protocolos para brindar el servicio que pretenden. A lo largo de su evolución, se vio la necesidad de dividir el gateway y de allí la especificación de MEGACO, pero esto como una opción.

Teniendo lo anterior en consideración se procede a una comparación más detallada de H.323 y SIP, protocolos que pueden actuar separados y que, básicamente, pretenden dar solución al mismo problema: servicios multimedia en redes de paquetes.

2.2.4.1. Comparación entre H.323 y SIP

El cuadro 2.1 muestra algunas diferencias entre los dos protocolos.

Existen ciertas diferencias entre los dos protocolos que son defendidas arduamente por sus patrocinadores. Lo que unos ven como una ventaja, los otros lo consideran una desventaja, con argumentos válidos en los dos casos. Sin embargo, existen también muchas similitudes: a saber, el modelo *peer-to-peer*, la forma en que se reparten las funciones entre los elementos

H.323	SIP
Protocolo complejo	Protocolo simple
Arquitectura unitaria	Arquitectura modular
Pensado desde el comienzo para la interconexión con la PSTN	SIP no fue pensado para tener compatibilidad hacia atrás con la PSTN. Usa Megaco para la interconexión.
Representación binaria de los mensajes mediante ASN.1	Representación textual de los mensajes
Soporta conferencias para datos y video	Soporte limitado de video y no tiene soporte de datos
Servicios suplementarios claramente especificados y estandarizados	Se intenta estandarizar funciones para desarrollar servicios, pero no los servicios
Gran presencia en el mercado	Auspiciado por la IETF y adoptado por 3GPP para 3G de celular

Cuadro 2.1: Comparación H.323 y SIP

del sistema (se podría decir que existen elementos análogos: *gatekeeper*, SIP *Registrar*), etc.

Se sabe que H.323 es un protocolo más complejo que SIP, que utiliza más recursos de hardware que su competidor en las soluciones en que se ha implementado, pero sus defensores lo explican por el hecho de que H.323 es un protocolo con más funcionalidades que SIP y que por lo tanto cubre más detalles a los que SIP no hace caso, y que, por lo tanto, está mejor preparado para su uso actual y su interconexión con las redes existentes (PSTN). Los defensores de H.323 argumentan que los problemas no son simples, las soluciones tampoco.

H.323 contiene una visión más unitaria, determinando una cantidad de protocolos que funcionan coordinados intentando cubrir cada aspecto del problema (señalización básica, registración, intercambio de capacidades, etc). Desde sus críticos se dice que no existe una clara separación de los componentes de esos protocolos y que para brindar un servicio se requieren interacciones entre muchos de ellos (por ejemplo, en una transferencia de llamada participan H.225, H.245 y H.450). SIP, y en particular el IETF, tienen una visión más modular de la solución, especificando protocolos que deben ser independientes de la aplicación. SIP sirve para establecer sesiones, SDP se usa para intercambiar capacidades, pero se podría usar cualquier otro protocolo.

Dada la organización desde donde surge, H.323 fue pensado desde el comienzo para interconectarse con la PSTN. Por esto se han cubierto todos los detalles para que esto fuera posible, mientras que con SIP se pensó en una solución totalmente nueva sin ataduras a las soluciones telefónicas existentes.

En cuanto a la representación de los mensajes, H.323 adopta ASN.1 que permite mensajes más concisos y un desarrollo más dinámico mediante el uso de “parsers” ya implementados. La gente de SIP argumenta que es más

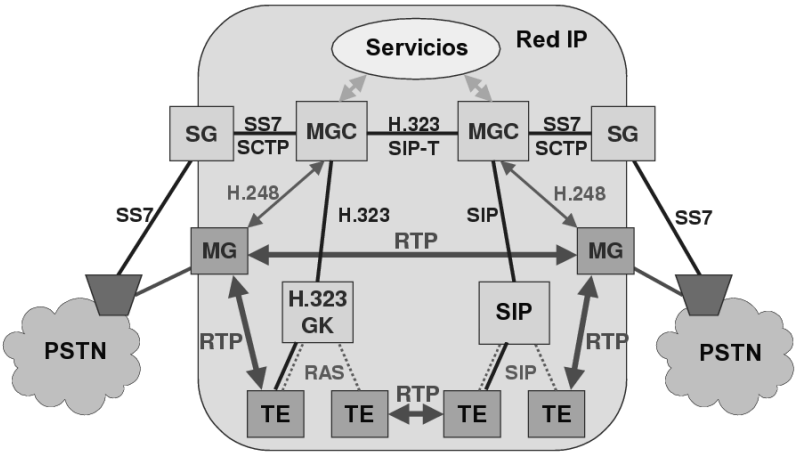

Figura 2.7: Interconexión de los distintos protocolos

simple usar mensajes de texto, que posibilitan un mejor "*debugeo*" y dicen que el ancho de banda no será un problema en el futuro; cosa que los patrocinadores de H.323 sospechan, principalmente en el área celular.

También existen diferencias en los servicios suplementarios. H.323 ha producido toda una serie de recomendaciones (la serie H.450) que especifican detalladamente los servicios suplementarios tal cual están implementados en la PSTN (*call forwarding*, *call hold*, etc). La visión de los promotores de SIP en cuanto al tema, promueve estandarizar un marco para el desarrollo de servicios suplementarios pero deja para los desarrolladores la especificación de éstos. No obstante, existen conceptos similares en las dos ópticas, tratando de mantener la inteligencia en los terminales y no usar elementos de la red para desarrollar servicios, si bien últimamente H.323 brinda la posibilidad de encarar otra opción mediante el uso del anexo L y el anexo K.

Evidentemente, la difusión en el mercado es mayor para H.323 debido a que fue más factible desarrollar aplicaciones comerciales que utilizaran dicho protocolo. Sin embargo, SIP surge cada día como un protocolo cada vez más usado. Es de esperar que en los próximos años las redes para telefonía IP evolucionen compartiendo los distintos protocolos existentes, figura 2.7.

2.3. H.323

El estándar H.323 es especificado por la ITU-T "Study Group 16" y su primer versión fue aceptada en Diciembre de 1996. La recomendación ITU-T H.323 es un compendio de otras recomendaciones (H.450, H.225, H.245), el conjunto de éstas especifica: el stack de protocolos, los elementos de la red y los procedimientos para establecer comunicaciones multimedia sobre redes de paquetes. El estándar originalmente fue pensado para dar servicio de conferencias multimedia sobre LAN's las cuales no proveen calidad de

servicio.

2.3.1. Arquitectura H.323

Una red H.323 está compuesta por cierto número de zonas interconectadas. Cada zona está compuesta por un único *gatekeeper* H.323, un cierto número de *terminales* H.323, uno o varios *gateways* H.323, y ciertos *Multipoint Control Unit*s o MCU H.323.

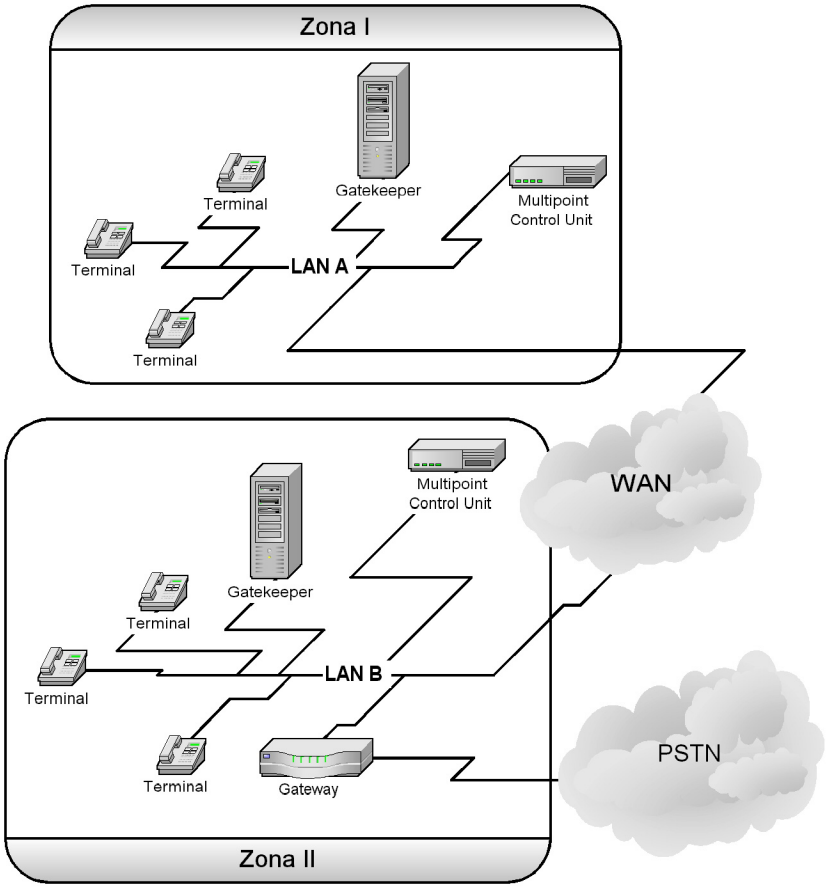

Figura 2.8: Ejemplo de red H.323

Los *terminales*, *gateways* y *multipoint control units* son denominados "puntos extremos" (*endpoints*). Cuando se habla de un *endpoint*, se refiere a cualquiera de estas tres entidades.

Una zona se define por todos los *endpoints* que pueden registrarse con el *gatekeeper*. El requisito fundamental es que cada zona contiene exactamente un *gatekeeper* cumpliendo las funciones de tal entidad en cada instante de tiempo. En la figura 2.8 se muestra un ejemplo donde dos zonas (I y II) son administradas por sus propios *gatekeepers* y están interconectadas por un WAN. Obsérvese que mientras cada zona posee su propio *MCU*, so-

lo la Zona II contiene un *gateway* conectado a la PSTN. Por lo que si es necesario que alguno de los terminales de la Zona I puedan comunicarse con este *gateway* de Zona II, deberá ser resuelto por el par de *gatekeepers* involucrados.

2.3.1.1. Terminal

Un terminal H.323 es el *endpoint* en la red que provee comunicaciones bidireccionales en tiempo real con algún otro *endpoint* (*terminal*, *multipoint control unit* o *gateway*).

Estas comunicaciones consisten en mensajes de control, indicaciones, paquetes de audio, video y/o datos, entre dos endpoints.

Una terminal puede establecer una llamada con otro *endpoint* directamente o con la ayuda de un *gatekeeper*. Puede ser una computadora personal corriendo una aplicación H.323 multimedia o un terminal H.323 nativo (dispositivo individual, cuya única función sea terminal H.323).

Deber ser capaz de soportar comunicaciones de audio y, opcionalmente, de video y/o de datos.

2.3.1.2. Gatekeeper

Esta entidad es la encargada de proveer resolución de direcciones y control de acceso en la red, a los *endpoints*.

Además puede proveer otros servicios como administración de ancho de banda, autenticación, facturación y otros.

La presencia de un *gatekeeper* es opcional en una red H.323, pero cuando este está presente en la red deberá ser utilizado por los *endpoints*.

Esta entidad es el núcleo del presente proyecto por lo que se la explicará con mayor detalle en la sección 2.3.4.

2.3.1.3. Gateway

Es la entidad *endpoint* encargada de proveer la posibilidad de establecer comunicaciones bidireccionales, en tiempo real, entre *terminales* H.323 en la red de paquetes y terminales en una red No-H.323.

Por ejemplo un *gateway* puede conectar y proveer comunicación entre un *terminal* H.323 y un terminal en la tradicional PSTN. Esta conectividad entre redes disímiles, debe ser implementada mediante la "traducción" de protocolos en ambos lados del *gateway*, para el establecimiento y terminación de llamadas, así como la conversión de formatos de audio/video entre las dos redes.

2.3.1.4. Multipoint Control Unit (MCU)

Es la entidad de la red H.323 encargada de proveer la capacidad de que tres o más terminales y/o *gateways* puedan participar en una conferencia multipunto.

Cada terminal y/o *gateway* participante de la conferencia debe establecer una conexión con esta unidad de control multipunto para el uso de este servicio. Entonces el MCU deberá, administrar recursos para conferencias,

negociar con los terminales y/o *gateways* para determinar el codec de audio o video a utilizar durante la conferencia y opcionalmente podrá manejar las tramas de audio y video que le llegan.

Específicamente esta entidad se encuentra dividida en dos unidades; el controlador multipunto (MC) que es obligatorio y procesadores multipunto (MP) que son opcionales. La función del primero es encargase de la señalización H.245 entre los participantes de la conferencia para el intercambio de capacidades con los *endpoints*. Mientras que la función del procesador multipunto es el procesamiento de las tramas de audio, video y/o datos.

Es factible que el MC esté presente en un *gateway*, en un *gatekeeper* o hasta en un *terminal*, además del propio MCU. Lo mismo puede ocurrir con el MP.

2.3.2. Protocolos en H.323

La recomendación H.323 directamente no define un nuevo protocolo, sino que especifica la manera en que sus componentes o entidades se comunican entre sí a través de una serie de otros protocolos (definidos en otras recomendaciones, ver figura 2.9), para propiciar el intercambio de información entre ellos, ya sea voz, video, o información de control. El objetivo final es propiciar el establecimiento de comunicaciones multimedia en redes de paquetes sin calidad de servicio garantizada.

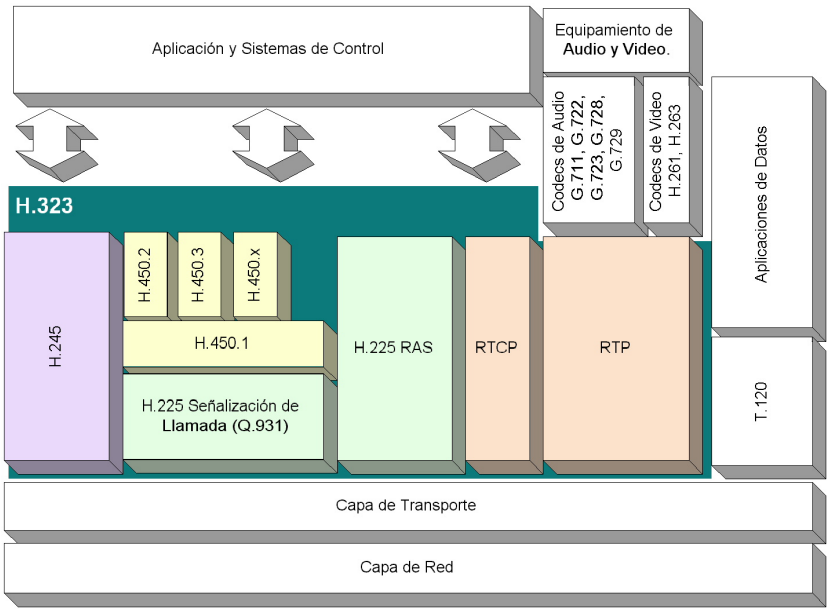

Figura 2.9: Stack de protocolos.

El stack es independiente de la red de paquetes y de los protocolos de transporte sobre los cuales es soportado, éstos pueden ser por ejemplo Ethernet, TCP-UDP/IP, ATM o Frame Relay.

O sea, la recomendación H.323 no especifica ningún tipo de interfaz de red como obligatoria o preferida. Sin embargo, sí se menciona que esta interfaz debe proporcionar un servicio fiable (ejemplos; TCP o SPX) para los canales de control H.245, canales de datos, y el canal de señalización de llamada H.225/Q.931, y un servicio de tipo no fiable (ejemplos; UDP o IPX) para los canales de audio, video y canal RAS. Tampoco se encuentra especificada la interfaz de usuario, ni dispositivos de audio o video, se ofrece total libertad en este sentido al fabricante.

A continuación se mencionarán aquellos protocolos y recomendaciones que son de interés. Se profundizará en aquéllos de mayor importancia para el presente proyecto y sobre el resto solo se mencionará su rol dentro de la recomendación.

2.3.2.1. Codecs de audio y video

Un codec de audio codifica una señal de audio proveniente de un micrófono para ser transmitida por el terminal emisor y decodifica las señales de audio recibidas que luego serán enviadas a un dispositivo de salida en el terminal receptor.

La comunicación de audio es el servicio mínimo que debe brindar un terminal H.323, por lo que todo terminal debe soportar por lo menos un codec de audio, este codec mínimo es el especificado en la recomendación ITU-T G.711, codificación de audio mediante Ley μ o Ley A a 64kbps.

Un terminal podría ser capaz de utilizar optativamente además otros codecs los cuales serían negociados entre los terminales involucrados a través de H.245. Algunos ejemplos de éstos son los especificados en las recomendaciones G.722 (64, 56 y 48kbps), G.723.1 (5,3 y 6,3kbps), G.728 (16kbps), G.729 (8kbps), etc.

El terminal H.323 debería tener la posibilidad de funcionamiento asimétrico para todas las capacidades de audio que haya declarado dentro del mismo conjunto de capacidades; por ejemplo, debería poder enviar G.711 y recibir G.728 si es capaz de ambas cosas.

Un codec de video se encarga de codificar señales de video provenientes de una cámara para ser transmitidas por el terminal emisor y decodificar señales de video recibidas por el terminal receptor para luego de decodificada ser enviada al dispositivo de salida de video del terminal.

Dado que el servicio de video es opcional en H.323 es opcional que los terminales soporten codecs de video, pero si lo hace éstos deben ser los especificados en las recomendaciones H.261 y H.263.

2.3.2.2. Servicios de datos

La utilización de canales de datos por parte de los *endpoints* es opcional según la recomendación [H323]. Pero cuando esté presente se debe utilizar el protocolo T.120 como plataforma por defecto para el soporte de este servicio.

Los canales de datos podrán ser unidireccionales o bidireccionales, según lo requiera la aplicación, también se podrá abrir uno o varios de estos canales. Pero, el procedimiento para la apertura de estos canales deberá

seguir el mismo camino que los servicios de voz y video, que se detallarán más adelante.

2.3.2.3. H.225 RAS - Registration Admission and Status

Se refiere al conjunto de mensajes especificados en la recomendación H.225.0 y procedimientos en la H.323 que proveen tres funciones de señalización; registro, admisión y estado, de ahí las siglas utilizadas (RAS) para identificarlo. Forman un protocolo de comunicación entre los *endpoints* y el *gatekeeper*, usado por los *endpoints* para; descubrir el *gatekeeper*, registrase con él así como eliminar este registro, solicitar la admisión de llamadas, la reserva de ancho de banda H.323[1], y desconectar una llamada.

En la sección 2.3.5 se describen detalladamente los procedimientos de conexión y se pueden visualizar los mensajes intercambiados, por ejemplo particularmente ver los siguientes diagramas: para el establecimiento de una llamada ver figura 2.18 y para la liberacion de esta ver figura 2.21 .

Para el intercambio de mensajes de este protocolo, es abierto el canal de señalización RAS entre el *gatekeeper* y un *endpoint*, previo a cualquier otro tipo de canal, que además, es independiente del canal de señalización de llamada H.225/Q.931 y del canal de control H.245. Obviamente, si el *gatekeeper* no está presente este canal no es utilizado.

Los mensajes RAS son enviados sobre una canal no fiable, UDP/IP por ejemplo. Por lo tanto durante el intercambio de estos mensajes el *gatekeeper* debe iniciar *timers* y reintentar en caso que estos expiren, es decir, queda para el fabricante o implementador del *gatekeeper* asegurar la fiabilidad del canal RAS.

El *gatekeeper* puede usar este protocolo para consultar el estado de los endpoints. Además está previsto un mecanismo para que diferentes *gatekeeper* se puedan comunicar entre sí y resolver direcciones de *endpoints* en diferentes Zonas.

Mensajes RAS

En el cuadro 2.3 se muestran algunos de los mensajes RAS especificados por la recomendación, que se consideraron relevantes. A continuación se explicarán y detallarán éstos.

2.3.2.3.1. Mensajes de descubrimiento Este proceso de descubrimiento es empleado por los *endpoints* para encontrar el *gatekeeper* en el cual deben registrase.

El descubrimiento puede hacerse en forma estática o dinámica. En el primer caso el punto extremo conoce la dirección de transporte del *gatekeeper* a priori e inicia directamente una secuencia “registro de endpoint”.

En el método dinámico, el punto extremo debe enviar un mensaje GRQ en forma *multicast* a una dirección predeterminada para este fin. La recomendación H.225 indica, en caso de implementaciones sobre redes UDP/IP

[1]Se entiende por ancho de banda H.323 la velocidad binaria de audio o video más encabezados de los paquetes usados para el transporte de estos medios, en sentido bidireccional, entre entidades que cumplen con la referida recomendación.

Mensajes RAS		*Endpoint Tx*	*Endpoint Rx*	*Gatekeeper Tx*	*Gatekeeper Rx*
Mensajes de descubrimiento					
GRQ	Gatekeeper Request	Opcional			Obligatorio
GCF	Gatekeeper Confirm		Opcional	Obligatorio	
GRJ	Gatekeeper Reject		Opcional	Obligatorio	
Mensajes de registro					
RRQ	Registration Request	Obligatorio			Obligatorio
RCF	Registration Confirm		Obligatorio	Obligatorio	
RRJ	Registration Reject		Obligatorio	Obligatorio	
Mensajes de admisión					
ARQ	Admission Request	Obligatorio			Obligatorio
ACF	Admission Confirm		Obligatorio	Obligatorio	
ARJ	Admission Reject		Obligatorio	Obligatorio	
Mensajes de desregistro					
URQ	Unregistration Request	Opcional	Obligatorio	Opcional	Obligatorio
UCF	Unregistration Confirm	Obligatorio	Opcional	Obligatorio	Opcional
URJ	Unregistration Reject	Opcional	Opcional	Obligatorio	Opcional
Solicitud de cambio de ancho de banda					
BRQ	Bandwidth Request	Obligatorio	Obligatorio	Opcional	Obligatorio
BCF	Bandwidth Confirm	Obligatorio	Obligatorio	Obligatorio	Opcional
BRJ	Bandwidth Reject	Obligatorio	Obligatorio	Obligatorio	Opcional
Mensajes de Desligamiento					
DRQ	Disengage Request	Obligatorio	Obligatorio	Opcional	Obligatorio
DCF	Disengage Confirm	Obligatorio	Obligatorio	Obligatorio	Obligatorio
DRJ	Disengage Reject	Obligatorio	Obligatorio	Obligatorio	Obligatorio

Cuadro 2.3: Mensajes RAS

TCP/IP, puertos y direcciones IP multicast en los cuales el *gatekeeper* debe recibir mensajes, ver cuadro 2.4.

El resultado es el retorno de un mensaje GCF (*Gatekeeper Confirmation*) por parte de uno o varios *gatekeeper*. También existe el mensaje GRJ (*Gatekeeper Reject*) usado para indicar específicamente que el controlador de acceso no desea que el terminal se registre en él.

Si más de un *gatekeeper* responde queda al endpoint decidir en cual registrarse.

Puerto o dirección multicast	Finalidad
224.0.1.41	Dirección multicast para comunicaciones con el *gatekeeper*
1718	Puerto UDP para comunicaciones multicast con el *gatekeeper*
1719	Puerto UDP para comunicaciones RAS unicast

Cuadro 2.4: Puertos y direcciones del *gatekeeper*

Los tres mensajes de este grupo tienen un campo *protocolIdentifier* el cual identifica la versión de la Recomendación H.225.0 de quien lo envía. Existe un campo *requestSeqNum* presente en los tres mensajes el cual es enviado en el GRQ y debe ser devuelto en cualquiera de las dos posibles respuestas (GCF, GRJ) obtenidas.

En lo que sigue se detallaran sólamente los campos de los mensajes que se consideran importantes de destacar, existen otros como por ejemplo los referentes a seguridad según H.235 que no se detallarán.

***Gatekeeper Request* (GRQ, petición de *gatekeeper*)** A continuación se detallan algunos de los campos presentes en un mensaje *GRQ*:

1. *rasAddress:* el terminal utiliza este campo para indicarle al *gatekeeper* su dirección de transporte a utilizar para mensajes RAS.

2. *gatekeeperIdentifier:* es un string que identifica el *gatekeeper* al que le solicitará posteriormente autorización para registrarse. Si no está presente este campo, o es nulo, indica que el terminal está interesado en registrarse en cualquier *gatekeeper*.

3. *endpointAlias:* es utilizado por el terminal originante para enviar la lista de alias mediante los cuales se lo identificará.

***GatekeeperConfirm* (GCF, confirmación de *gatekeeper*)** El mensaje GCF incluye un conjunto de campos análogos a los explicados en el mensaje GRQ por lo que éstos no se detallaran aquí,

Los siguientes no están presentes o son diferentes a los explicados en el mensaje GRQ:

1. *gatekeeperIdentifier:* es un string que identifica al *gatekeeper*.

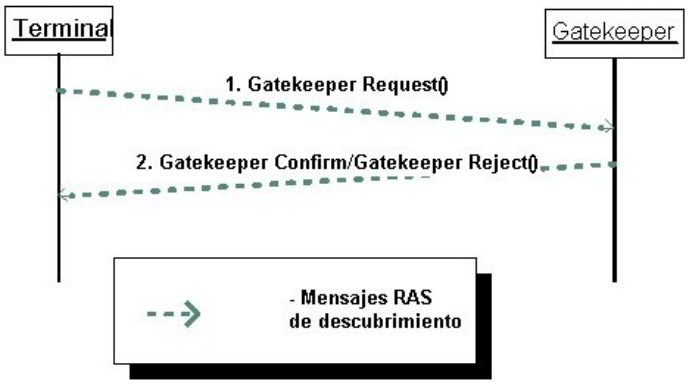

Figura 2.10: Mensajes de Descubrimiento

2. *rasAddress:* dirección de transporte que el *gatekeeper* utiliza para los mensajes RAS.

***GatekeeperReject* (GRJ, Rechazo desde el *gatekeeper*)** Al igual que lo dicho para el mensaje GCF, este mensaje incluye un conjunto de campos que no se detallan aquí, ya sea por no considerarse importantes o por ser análogos a los del mensaje GRQ.

Por lo que el único campo a destacar es el siguiente:

1. *rejectReason:* motivo de rechazo de la solicitud, es un enumerado que codifica el motivo por el que se rechazó la solicitud, como por ejemplo que no se tengan suficientes recursos disponibles, u otras causas relacionadas con aspectos de seguridad.

En la figura 2.10 se puede ver un ejemplo de los mensajes intercambiados.

2.3.2.3.2. Mensajes de registro Este grupo de mensajes es empleado por los terminales para "ingresar" en la zona administrada por el *gatekeeper* e informar a éste de su dirección de transporte, así como de su alias. Todos los puntos extremos deben ser configurados para registrarse con un *gatekeeper* en caso de que éste se encuentre disponible en la red.

Se indica que los *gateways* y MCUs podrían registrar una única dirección de transporte para el canal RAS y una única para el canal de señalización de llamada.

El mensaje usado para iniciar el registro es el RRQ (*Registration Request*), mientras que el *gatekeeper* responderá con el mensaje RCF (*Registration Confirmation*) para aceptar o RRJ (*Registration Reject*) para rechazar.

Es posible recibir periódicamente mensajes RRQ desde el mismo terminal, para lo cual la recomendación sugiere los procedimientos enumerados en el cuadro 2.5.

Mensaje RRQ recibido	Acción recomendada, obligatoria o sugerida
Contiene mismo alias y misma dirección de transporte que un registro activo (registro previo aceptado por el *gatekeeper*).	Es obligatorio responder con un RCF.
Contiene mismo alias y diferente dirección de transporte que un registro activo.	Se puede confirmar el registro (RCF) si se cumple con las políticas de registración, en caso contrario debería rechazar (RRJ).
Contiene diferente alias y la misma dirección de transporte que un registro activo (y no es un RRQ aditivo)	Se debería remplazar el registro en la tabla de traducción de direcciones.

Cuadro 2.5: Acciones al recibir diferentes mensajes RRQ.

Lo dicho para los mensajes de descubrimiento referente a los campos *protocolIdentifier, requestSeqNum* es igualmente aplicable a este grupo de mensajes.

***RegistrationRequest* (RRQ, solicitud de registro)** Éste es el mensaje que inicia la registración.

A continuación se detallan algunos de los campos que se incluyen en el RRQ:

1. *callSignalAddress:* dirección de señalización de llamada, es la dirección de transporte para la señalización de llamada que utilizará el endpoint.
2. *rasAddress:* dirección RAS del terminal originante.
3. *terminalAlias:* alias del terminal, este campo es opcional, es utilizado por el terminal originante para enviar la lista de alias mediante los cuales se lo identificará.
4. *timeToLive:* tiempo de vida o duración del registro, se expresa en segundos.
5. *keepAlive:* mantener vivo, es un booleano que indica que es una reregistración donde se omiten algunos campos para facilitar la tarea. Esto se explicará luego donde se describen los mensajes RRQ ligeros.

***RegistrationConfirm* (RCF, confirmación de registro)** El mensaje RCF incluye un conjunto de campos análogos a los explicados en el mensaje RRQ por lo que éstos no se detallaran aquí.

Los siguientes no están presentes o son diferentes a los explicados en el mensaje RRQ:

1. *endpointIdentifier:*identificador del terminal, es el string que identifica el terminal el cual es asignado por el *gatekeeper*.

2. *preGrantedARQ:* este campo es utilizado por el *gatekeeper* para indicarle al endpoint que en futuras llamadas no necesita enviar un ARQ para solicitar la autorización. Ver sección 2.3.6.1.

***RegistrationReject* (RRJ, rechazo de registro)** De forma análoga a lo anterior aquí se detallaran solo los campos que no están presentes o son distintos de los presentes en los mensajes anteriores.

1. *rejectReason:* motivo por el cual se rechazó el registro. Algunas de las posibles razones por las que se pudo haber rechazado el registro son por ejemplo: que el RRQ contenga un alias de un *endpoint* ya registrado, que la dirección RAS sea inválida, que la dirección de transporte de señalización sea inválida, etc.

Otros tipos de registración Existen variantes a la forma de registración antes explicada, esto se detalla a continuación.

- Mensajes RRQ ligeros

Es posible que el registro de un terminal con el *gatekeeper* exista durante un tiempo finito, es decir, después de que este tiempo ha expirado la información de dicho terminal es eliminada de la tabla de traducción de direcciones del *gatekeeper*. Este tiempo es solicitado por el endpoint mediante el campo *timeToLive* del mensaje RRQ, el *gatekeeper* podría responder con un RCF incluyendo el campo *timeToLive* con un valor menor o igual al solicitado.

Antes de que el tiempo expire el terminal debería enviar un nuevo RRQ el cual contiene el campo *keepAlive*, solicitando sea extendido por un período más el registro del mismo. Estos RRQ podrían contener un mínimo volumen de información, o sea, menos campos que un RRQ normal. Es por esto último que se los conoce como RRQ ligeros, o también *keepalive* RRQ.

- Registración aditiva

Un terminal registrado podría enviar un RRQ al *gatekeeper* incluyendo el campo *additiveRegistration*, este último deberá agregar la información recibida en este RRQ al registro del terminal. Si el mensaje contiene un lista de alias para el terminal, esta lista debe ser sumada a la que ya posee el controlador de acceso, sin que sea borrada la información que ya se poseía. En caso de las direcciones de señalización de llamada o dirección RAS deben ser suplantadas en el registro del terminal.

Este mecanismo de registro aditivo no es obligatorio para los terminales, ni para el *gatekeeper*.

2.3.2.3.3. Mensajes de admisión Este grupo de mensajes es utilizado para solicitar autorización por parte del terminal para realizar una llamada y para dar la respuesta por parte del *gatekeeper*. El terminal envía un

ARQ (*Admission Request*) solicitando autorización, el *gatekeeper* puede permitírselo contestando con un ACF (*Admission Confirm*) o negárselo mediante el envío de un ARJ (*Admission Reject*). Esto permite la implementación de políticas de admisión de llamada por parte del *gatekeeper*.

Los siguientes campos están presentes en los tres mensajes de este grupo por lo que se los explica en forma global.

El campo *destinationInfo* contiene la lista de identificadores del terminal destino, mientras que el *srcInfo* contiene la lista de identificadores del terminal origen. Estos identificadores pueden ser de tres tipos: dígitos, cadenas de caracteres o *PartyNumbers* de acuerdo con la recomendación E.164.

El campo *destCallSignalAddress* contiene la dirección de transporte de señalización de llamada a ser usado por el destino. Análogamente *srcCallSignalAddress* ser refiere a la dirección del origen.

El campo *requestSeqNum* es un número de secuencia para relacionar las solicitudes y sus respuestas, debiendo contestarse en el ACF o ARJ con el mismo *requestSeqNum* que el presente en el mensaje ARQ.

***AdmissionRequest* (ARQ, solicitud de admisión)** Para este mensaje no es necesario que estén presentes los dos elementos *destinationInfo* y *destCallSignalAddress*, pero al menos uno debe estar presente, excepto en el caso de que el ARQ provenga del terminal destino de la llamada.

A continuación se detallan algunos de los campos incluidos en el mensaje ARQ.

1. *callModel*: modelo de llamada, éste determina el modo de llamada solicitado por el terminal, modo de llamada directa o modo de llamada ruteada por el *gatekeeper*, ver sección 2.3.2.7.

2. *bandwidth:* ancho de banda, representa el ancho de banda bidireccional solicitado por el originante, en unidades de 100 bit/s. Donde ésta representa la velocidad total de audio o video, más encabezados.

3. *callReferenceValue:* valor de referencia de la llamada, utilizado por el terminal y *gatekeeper* para identificar la llamada a nivel de señalización Q.931, y asociar ésta con la ARQ en cuestión. Este valor es generado por el terminal originante y será repetido en los mensajes de señalización de llamada Q.931. En caso de llamada ruteada por *gatekeeper* este valor será diferente para los mensajes del terminal origen al *gatekeeper* y desde éste al terminal destino.

4. *answerCall:* booleano que indica el sentido de la llamada, el valor TRUE indica que este ARQ proviene del terminal destino, en este caso está solicitando autorización para contestar la llamada. Si es FALSE muestra que el ARQ proviene del terminal origen de la llamada.

5. *callIdentifier:* identificador de llamada, es el identificador único de la llamada, será el mismo para todos los mensajes, RAS o de señalización Q.931, relacionados con esta llamada. En caso de llamada ruteada por *gatekeeper* este valor deberá ser el mismo para los mensajes del terminal origen al *gatekeeper* y desde éste al terminal destino.

6. *willSupplyUUIEs:* valor booleano utilizado por el terminal para notificar al *gatekeeper* que posee la capacidad de enviar copia de los mensajes de señalización de llamada Q.931 enviados o recibidos por él. Un valor TRUE indica que posee esta capacidad mientras que el valor FALSE lo contrario.

***AdmissionConfirm* (ACF, confirmación de admisión)** A continuación se detallan algunos de los campos incluidos en este mensaje.

1. *bandwidth:* ancho de banda máximo permitido para la llamada por el *gatekeeper* pudiendo ser éste menor que el solicitado por el terminal.
2. *destinationType:* tipo de *endpoint* que es el destino.
3. *uuiesRequested:* mediante este campo el *gatekeeper* le solicita al *endpoint* que mensajes de señalización Q.931 le debe enviar como copia, en el caso en que el *endpoint* halla indicado esta posibilidad mediante el campo *willSupplyUUIEs* en el mensaje ARQ.

***AdmissionReject* (ARJ, rechazo de admisión)** A continuación se detallan algunos de los campos incluidos en este mensaje.

1. *RejectReason:* causa del rechazo, indica el motivo por el que se rechazó la solicitud. Algunas de las posibles causas por las que se puede rechazar la solicitud pueden ser, porque no se pudo traducir la dirección, esto es indicado por un valor *calledPartyNotRegistered,* otra causa podría ser que no se dispone de ancho de banda suficiente, esto se indicaría con un valor *requestDenied,* también podría ser porque se está en el modo de funcionamiento llamada ruteada por *gatekeeper* indicado por el valor *routeCallToGatekeeper.*
2. *callSignalAddress:* es la dirección de señalización de llamada del *gatekeeper* devuelta por éste en caso de que el motivo del rechazo sea modo de funcionamiento llamada ruteada por *gatekeeper*.

2.3.2.3.4. Mensajes de desligamiento Este grupo de mensajes es usado por *gatekeeper* y *endpoints* durante la finalización de una comunicación, en la cual dicho punto extremo se encontraba formando parte. Se utilizan los mensajes detallados a continuación.

***Disengage Request* (DRQ, solicitud de desligamiento)** Este mensaje puede ser enviado desde un terminal hacia el *gatekeeper* o a la inversa. En el primer caso el terminal está informando al *gatekeeper* que la comunicación está finalizando, esto tiene utilidad en el caso de que el *gatekeeper* no esté encaminando la señalización de llamada Q.931. Mientras que en el segundo caso, es el *gatekeeper* el que está obligando al terminal a terminar la comunicación.

***Disengage Confirm* (DCF, confirmación de desligamiento)** Este mensaje es utilizado para confirmar la finalización de una comunicación, pudiendo ser enviado desde el *gatekeeper* hacia un terminal o a la inversa. En el primer caso es enviado directamente al recibir éste el DRQ, mientras que en el caso de que el DRQ haya sido enviado por el *gatekeeper* los terminales terminarán las secciones RTP, cerraran los canales lógicos, terminarán la llamada mediante el intercambio de señalización Q.931 y por último envía el mensaje DCF al *gatekeeper* para informarle que la llamada ha finalizado.

***Disengage Reject* (DRJ, rechazo de desligamiento)** Este mensaje solo es enviado por el *gatekeeper* hacia un terminal en respuesta a un DRQ, el envío de este mensaje puede tener tres posibles causas, en primer lugar que el DRQ provenga de un terminal que no está registrado con el *gatekeeper*, que el DRQ provenga de un terminal que está intentando finalizar una llamada de otro terminal y por último puede deberse a causas especificadas mediante un valor *securityDenial*.

2.3.2.3.5. Mensajes de desregistro Este grupo es empleado para terminar o finalizar la relación entre un terminal y el *gatekeeper,* por ejemplo porque el usuario desea utilizar su alias desde otra dirección de transporte.

Para solicitar la desregistración por parte del terminal se utiliza el mensaje URQ (*UnregistrationRequest*), respondiendo el *gatekeeper* con un mensaje UCF (*UnregistrationConfirmation*). El *gatekeeper* termina el registro del terminal informándoselo con el mensaje URQ, respondiendo el terminal con el UCF.

Existe también el mensaje URJ (*UnregistrationReject*) empleado únicamente por el *gatekeeper* para rechazar la solicitud en el caso de que el terminal no se encuentre registrado, que exista un llamada en proceso u otras razones relacionadas con aspectos de seguridad.

2.3.2.3.6. Mensajes para cambio de ancho de banda Grupo de mensajes empleado para cambiar el ancho de banda utilizado por el terminal. El terminal pude solicitar un cambio en el ancho de banda utilizado empleando el mensaje BRQ (*BandwidthRequest*) y el *gatekeeper* le otorgará el nuevo ancho de banda confirmándoselo mediante el mensaje BCF (*BandwidthConfirm*), o podrá rechazarlo mediante el mensaje BRJ (*BandwidthReject*).

También pude ser usado por el *gatekeeper* para solicitarle al terminal que cambie su ancho de banda utilizado (elevar, o reducir este), el terminal responderá con BCF o BRJ. En caso de rechazo algunas posibles causa son por ejemplo: que no se cuente con recursos suficiente, que el terminal no posea los permisos necesarios, u otras razones de seguridad.

2.3.2.3.7. Otros grupos de mensajes Además existen otros grupos de mensajes los cuales no fueron incluidos en el cuadro 2.3, dichos grupos se detallan a continuación.

***Mensajes de localización* (LRQ, LCF, LRJ)** El grupo de mensajes LRQ (*Location Request*) y LCF (*Location Confirmation*) pueden ser usados por un terminal o un *gatekeeper* para solicitar la dirección de señalización de llamada y/o la dirección de canal RAS para un *endpoint* del cual sólo conoce su alias. El primero - LRQ - es usado para solicitar la información, mientras que LCF será empleado por el *gatekeeper* donde reside el registro del endpoint buscado para responder al LRQ y contendrá la información solicitada.

Estos mensajes pueden ser enviados directamente a la dirección de transporte del canal RAS o la dirección *multicast*, como los mensajes GRQ. En el primer caso podrían emplearse los mensajes LRJ (*Location Reject*) por parte de los *gatekeepers* donde no está registrado el terminal buscado, justamente para notificar que no poseen datos.

***Mensajes de Información* (IRQ, IRR, IACK, INAK)** El *gatekeeper* puede solicitarle a un terminal que le envíe información de estado, esto lo puede hacer enviándole un mensaje IRQ (*InfoRequest*) o mediante el campo *irrFrequency* en el mensaje ACF, con lo que además le está indicando la frecuencia en segundos con la que espera recibir los IRR. El terminal le enviará la información solicitada mediante un mensaje IRR (*InfoRequestResponse*). También es posible que un terminal envíe mensajes de información IRR que no le fueron solicitados, pudiendo él solicitar que éstos sean reconocidos mediante el acuse de recibo, para lo cual el *gatekeeper* utilizará los mensajes de acuse de recibo positivo IACK (*InfoRequestAck*) o negativo INAK (*InfoRequestNak*).

Mensajes no estandarizados La recomendación abre la posibilidad a los fabricantes de construir sus propios mensajes, para lo que define una estructura denominada *NonStandardMessage*, que contiene un campo (*nonStandardData*) para los datos propietarios.

***Mensajes de disponibilidad de recursos del* gateway (RAI, RAC)** Un gateway tiene la posibilidad de informar al *gatekeeper* de los recursos que tiene disponibles en ese momento mediante el RAI (*ResourcesAvailableIndicate*), y éste le responderá con un RAC (*ResourcesAvailableConfirm*) como confirmación. Se entiende por recursos disponibles un detalle de cada protocolo y la velocidad soportada para cada uno de ellos por el *gateway*.

2.3.2.4. H.225 - Protocolo de señalización de llamada H.225/Q.931

En una red H.323 los terminales deben utilizar un protocolo para establecer y terminar conexiones (llamadas). El empleado por H.323 para este fin es una variante del protocolo Q.931 definido para la PSTN. Se adoptó esta modalidad para simplificar la interconexión con redes PSTN/ISDN y estándares de conferencias multimedia en redes de conmutación de circuitos, como H.320 y H.324.

Las distintas entidades dentro de una red H.323 se comportan como entidades de señalización Q.931, pudiendo intercambiar un subconjunto de

mensajes Q.931 con elementos propios de H.323. No todos los tipos de mensajes, ni todos sus campos son utilizados por H.225, ya que muchos de ellos resultan inservibles en este ámbito.

Para el intercambio de mensajes de señalización de llamada H.225.0 se abre un canal de señalización de llamada *(call-signalling channel)* entre dos *endpoints* o entre un *endpoint* y el *gatekeeper*. Es abierto antes que el establecimiento del canal H.245 (canal de negociación de capacidades) y que cualquier otro canal lógico entre terminales H.323.

Según la recomendación H.225, cada *endpoint* que pretenda cumplir con esta, debe ser capaz de: recibir e identificar todos los mensajes Q.931 o H.450 entrantes, procesar los mensajes Q.931 que le son obligatorios, eventualmente procesar los mensajes Q.931 opcionales, y en caso de mensajes desconocidos poder ignorarlos sin que se altere su funcionamiento.

Las entidades intermedias como lo son el *gatekeeper* y los *gateway* deben funcionar como se explica a continuación;

1. Para el *gatekeeper* es obligatorio reenviar todos los campos de los mensajes obligatorios de Q.931 recibidos, aunque está habilitado a realizar modificaciones en estos. Para el *gateway* esto es preferible aunque no obligatorio. Esto incluye los elementos de información User-User que como se verá más adelante son de especial importancia para la señalización de llamada H.225/Q.931.

2. Un *gateway* debría remitir todos los mensajes opcionales Q.931 o H.450 y elementos de información en ambos sentidos.

3. Si el canal de señalización H.225/Q.931 está activo, el *gatekeeper* debe remitir todos los mensajes Q.931 o H.450 opcionales y los elementos de información luego de modificarlos si fuera necesario. Si el *gatekeeper* no mantiene el canal de señalización activo, esto no es posible. El *gatekeeper* puede actuar como una entidad de señalización modificando, originando y terminando mensajes Q.931 para proveer determinadas características, como por ejemplo servicios suplementarios del tipo transferencia etc.

En el cuadro 2.6 se muestran los tipos de mensajes Q.931 utilizados por la recomendación H.225. En la tabla se deja constancia de la obligatoriedad, o no, de las entidades H.323 de soportar dicho mensaje. Los restantes mensajes Q.931 están expresamente prohibidos para su utilización en redes H.323 debido a las características de este tipo de redes, que hacen inútil su utilización.

	Transmisión	Recepción y Acción
Mensajes de establecimiento de llamada		
Aviso (alerting)	Obligatorio	Obligatorio
Llamada en curso (call proceeding)	Opcional	Condicionalmente Obligatorio
Conexión (connect)	Obligatorio	Obligatorio
Progreso (progress)	Opcional	Opcional
Establecimiento (setup)	Obligatorio	Obligatorio
Acuse de Establecimiento (setup Ack)	Opcional	Opcional
Mensajes de liberación de llamada		
Liberación Completa (release complete)	Obligatorio	Obligatorio
Mensajes de información de llamada		
Información de Usuario (user information)	Opcional	Opcional
Mensajes misceláneos		
Información (information)	Opcional	Condicionalmente Obligatorio
Notificación (notification)	Opcional	Opcional
Situación (status)	Obligatorio	Obligatorio
Consulta de situación (status inquiry)	Opcional	Obligatorio
Mensajes Q.932/H.450		
Facilidad (facility)	Obligatorio	Obligatorio

Cuadro 2.6: Mensajes Q.931 usados en H.323

A continuación se explican brevemente los mensajes más importantes.

Mensajes Q.931 de establecimiento de llamada

Los mensajes Q.931 utilizados en el establecimiento de una llamada son los que se detallan a continuación. Para visualizar mejor la forma en que se da este intercambio se puede ver en la sección 2.3.5 diferentes diagramas; por ejemplo, para el caso de una llamada directa ver figura 2.18.

1. *Establecimiento (Setup)* Este mensaje es enviado por el terminal originante de la llamada para indicar que desea establecer una llamada con el terminal destino.

2. *Acuse de establecimiento (SetupAck)* Puede ser enviado por una entidad H.323 para indicar el acuse de recibo del mensaje *setup*. Su procesamiento es opcional por parte del que lo recibe, mediante el campo

canOverlapSend en el mensaje *setup* que puede indicar que soportará el mensaje *setupAcknowledge*.

3. *Llamada en curso (Call Proceeding)* Este mensaje puede ser enviado por el terminal destino para indicar que el establecimiento de la llamada solicitada ha sido iniciado y no aceptara más información de establecimiento de llamada.

4. *Aviso (Alerting)* Es enviado por el usuario destino al origen para indicar que se ha iniciado el aviso de usuario llamado, es decir el teléfono llamado está "timbrando", y el originante de la llamada escucharía el tono de "ring back".

5. *Conexión (Connect)* Es enviado por el terminal destino al originante del mensaje *setup* para indicar que se ha aceptado la llamada. El originante de este mensaje pude ser un *endpoint* o el controlador de acceso en caso de llamada encaminada por *gatekeeper*.

6. *Progreso (Progress)* Este mensaje es enviado por un *gateway* H.323 para indicar el progreso de una llamada en caso de interconexión con la PSTN.

Mensajes Q.931 de liberación de llamada

En H.323 no se usa la secuencia desconexión/liberación/liberación completa como en Q.931, ya que lo que aporta es la posibilidad de agregar un elemento de información red a usuario al mensaje liberación. Esto no es aplicable al entorno de la red de paquetes, por lo cual se utiliza el método de liberación de un solo paso enviando sólo el mensaje liberación completa. A modo de ejemplo se puede ver el diagrama de la figura 2.21.

1. *Liberación Completa (Release Complete)* Este mensaje es enviado por el terminal para indicar la liberación de la llamada si el canal de señalización esta abierto. Luego de esto el CRV (*Call Reference Value*) queda disponible para ser reutilizado.

Mensajes Q.932/H.450

1. *Facilidad (Facility)* Este mensaje deriva de la recomendación Q.932 y de H.450. El mensaje *Facility* puede cumplir varias funciones, por ejemplo para que un terminal indique que la llamada entrante debe pasar por un *gatekeeper* (*FacilityReason* = *routeCallToGatekeeper*). También puede usarse para dar acuse de recibo o solicitar un servicio suplementario según H.450.

Los únicos campos del encabezado de un mensaje Q.931 que deben ser utilizados por las entidades H.323 son: discriminador de protocolo, CRV (*Call Reference Value*) y *Message Type*. El CRV lo elige el terminal que origina la llamada y se conserva dicho valor para los restantes mensajes de control de llamada. La norma aclara expresamente que el valor del CRV debe ser único para cada tramo de la llamada. O sea, si la llamada es ruteada a través del *gatekeeper*, existen dos valores de CRV para la misma llamada, uno para

los mensajes que intercambia el terminal llamante con el *gatekeeper*, y otro para los mensajes que intercambian el *gatekeeper* y el terminal llamado.

En cuanto a los *Information-Elements* definidos en la norma Q.931 , H.225 especifica la forma en que se utilizan, determinando para cada tipo de mensaje la obligatoriedad, o no, de su utilización particular.

El *Information-Element User-User* de Q.931 contiene una estructura ASN.1 [ASN1] llamada *H323-UserInformation* la cual tiene la información relevante de H.323 (*h323-uu-pdu*), así como la información de usuario (*user-data*).

La estructura *h323-uu-pdu* contiene campos específicos con información particular para un mensaje de Q.931 (*h323-message-body*), paquetes de H.450 (*h4501SupplementaryService*), campos para transportar paquetes de H.245 en el caso de que se utilice tunelización de H.245 (*H245Control*), paquetes de H.248 para el protocolo basado en estímulos, etc.

En la estructura *h323-message-body* se transporta información específica de H.323 para cada tipo de mensaje Q.931. Para ellos se definen estructuras en ASN.1 (*Setup-UUIE*, *Facility-UUIE*, *Connect-UUIE*) con campos obligatorios y opcionales. A través de estos campos los terminales se intercambian: identificación de la versión de protocolo que soportan, información para abrir los canales lógicos (direcciones de transporte, etc), el método de negociación de los canales lógicos (*fastStart*, tunelización de H.245, etc), elementos de autenticación y autorización (H.325), etc., ver cuadro 2.7.

Nombre de campo	Función
h323-message-body	Contiene información específica de cada mensaje Q.931 particular
nonStandarData	Para transportar información no definida en H.225. Información propietaria
h4501Supplementary-Service	Secuencia de APDU de H.450.1
h245Tunneling	Booleano que indica la tunelización de H.245
h245Control	PDU de H.245 cuando existe tunelización de H.245
nonStandarControl	Control de información propietaria
callLinkage	Sirve para agrupar llamadas correspondientes a un mismo servicio o aplicación
tunnelledSignalling-Message	Mensaje tunelizado de señalización externa para el control de llamada por entidades externas
provisionalRespTo-H245Tunneling	Bandera que indicar a la entidad llamada que el llamado aún no ha decidido si tunelizará H.245
stimulusControl	Para uso de control por estímulo H.248
genericData	Datos genéricos para servicios definidos fuera de H.225

Cuadro 2.7: Campos de h323–uu-pdu

2.3.2.5. H.245 Protocolo de control

Este protocolo es utilizado por los *endpoints* como punto de partida para el establecimiento de los canales de medios.

La Recomendación H.245 especifica los procedimientos de intercambio de mensajes así como su sintaxis y semántica.

Los mensajes pueden ser referentes al intercambio de capacidades de recepción y transmisión de medios (audio y/o video) y datos, para manifestar la preferencia de un modo de transmisión determinado, para la gestión de los canales lógicos, para la determinación de relaciones del tipo maestro/esclavo en caso de canales bidireccionales, también pueden ser mensajes de indicación o control, o para la medida de retardos entre terminales, etc.

La recomendación define los mensajes y protocolos de forma independiente del mecanismo de transporte usado, se supone una capa de transporte fiable. En la figura 2.19 se puede ver un ejemplo de los mensajes de control intecambiados.

A continuación se describirán cada una de las principales funciones cumplidas a través de H.245.

Intercambio de capacidades El intercambio de capacidades tiene por objeto que los datos y medios transmitidos sean entendidos por el terminal receptor. Por lo cual es necesario que cada terminal conozca las capacidades de recepción y decodificación del otro terminal.

Para lograr lo anterior es que se utilizan procedimientos de intercambio y negociación de las capacidades entre los terminales.

Estas negociaciones también se utilizan para determinar modos de funcionamiento en los que se estén transmitiendo y recibiendo simultáneamente flujos de trenes de medios independientes con diferente codificación, por ejemplo audio con G.722 y video con H.262, o audio en diferentes formatos por ejemplo G.711 y G.729.

Estas negociaciones no son estáticas, es decir que se puede renegociar con la llamada en curso.

La negociación se hace en base a estructuras de capacidades según se explica a continuación.

Estructuras intercambiadas: Cada codec y/o modo de funcionamiento que es capaz de utilizar un *endpoint*, recibe un número único que lo identificará. Estos codecs se agrupan, en una primera instancia, en una tabla denominada justamente *capabilityTable*. Esta contiene entonces todos los codecs (y sus identificadores) soportados por el punto extremo.

Luego, los identificadores son agrupadas en estructuras llamadas *AlternativeCapabilitySet,* con lo que el terminal indica que puede funcionar en exactamente uno y solo uno de los modos especificados en ella, es decir, con uno solo de los codecs listados a la vez, por ejemplo la lista {G.711 Ley μ, G.711 Ley A, G.723.1}.

A su vez estas *AlternativeCapabilitySet* son agrupadas en estructuras denominadas *simultaneousCapabilities,* con las que el terminal da a conocer en que modos puede funcionar simultáneamente. Esta estructura indica que el terminal es capaz de funcionar simultáneamente con un codec

de cada *AlternativeCapabilitySet* dada. Un ejemplo puede ser el siguiente *simultaneousCapabilities* {{H.261}, {H.261, H.263}, {G.711, G.723.1, G.728}} compuesto de tres *AlternativeCapabilitySet* está indicando que el terminal es capaz de operar con un canal de video H.261, otro canal de video H.261 o H.263 y un canal de audio G.711, G.723.1 o G.728.

Por último, se crean las *CapabilityDescriptor,* estas constan de una *simultaneousCapabilities* y un número que la identifica, denominado *capabilityDescriptorNumber.* Las capacidades totales de un terminal están descriptas por un conjunto de *CapabilityDescriptor.* Todos los terminales deben transmitir por lo menos una *CapabilityDescriptor.* Se pueden renegociar las capacidades con la llamada en curso transmitiendo nuevas *CapabilityDescriptor*.

Determinación de la relación maestro/esclavo En comunicaciones bidireccionales se pueden dar conflictos si ambos terminales participantes desean disparar determinados eventos simultáneamente y sólo puede ocurrir uno a la vez, por ejemplo porque los recursos que usan están disponibles para uno solo de ellos por vez.

Para resolver esto es que se usa una modalidad de funcionamiento en la que uno de los terminales oficia de maestro y el otro de esclavo en cuanto a la toma de decisiones.

Existe un procedimiento para la determinación de cual terminal oficia de maestro y cual de esclavo, éste puede ser disparado nuevamente para cambiar la situación aun con la llamada en curso bajo ciertas condiciones.

Apertura y cierre de canales lógicos La recomendación define un procedimiento para la apertura y cierre de canales lógicos, para lo cual se intercambian mensajes de solicitud y de acuse de recibo. Esto tiene por objeto que cuando se abran los canales lógicos los terminales ya estén en condiciones de recibir y procesar los datos que se transmitirán por él.

Para este procedimiento se usa el esquema maestro/esclavo descripto antes. En caso de canales bidireccionales, sólo el terminal maestro puede solicitar su apertura.

En el mensaje de apertura del canal se envía una descripción del tipo de información que se transmitirá por él.

Los canales lógicos son abiertos y cerrados desde lado del terminal transmisor. Se da un mecanismo por el cual el terminal receptor puede solicitar al transmisor el cierre de un canal lógico entrante, por ejemplo porque no puede seguir procesando los datos por algún motivo, el terminal transmisor puede aceptar o no esta solicitud.

Control de modo de funcionamiento Una vez intercambiadas las capacidades el terminal receptor utiliza un conjunto de mensajes para solicitar al terminal transmisor determinado modo de transmisión.

Se utiliza un mensaje de *petición de modo* para solicitar un modo determinado de transmisión.

Este mensaje es una lista, en orden de preferencia decreciente de los modos en que el terminal desea trabajar. Cada modo es descripto a través de

un *ModeDescription*. Se utilizan números de secuencia *(sequenceNumber)* para identificarlos.

Un ModeDescription es un conjunto de uno o más *ModeElements*. Estos últimos *(ModeElement)* indican el tipo de tren elemental solicitado.

Pudiendo ser éste uno de los siguientes: *VideoMode*, *AudioMode*, *DataMode*, *EncryptionMode* y *H235Mode*.

Se utiliza un mensaje de *Acuse de recibo de petición de modo* el cual es enviado para confirmarle al receptor que el transmisor está intentando transmitir en uno de los modos solicitados por él. En el acuse de recibo se manda el número de secuencia del modo reconocido.

2.3.2.6. RTP - Real Time Protocol

Es el protocolo usado para el transporte de las tramas de audio y video en H.323. Se lo adoptó directamente de las RFC de la IETF (Internet Engineering Task Force), además de su protocolo de control asociado; RTCP (Real-Time Control Protocol).

En la sección 2.3.5 se describen los procedimientos de conexión, donde se puede ver en la figura 2.20 el intercambio de paquetes RTP.

2.3.2.7. Modos de funcionamiento

En una arquitectura H.323 se pueden tener dos modelos de control de llamada: llamada directa (*direct call*) y llamada encaminada por *gatekeeper* (*gatekeeper routed call*).

En cualquiera de ellos los canales RTP son establecidos directamente entre los terminales involucrados, es decir el tráfico de los trenes de medios no pasa por el *gatekeeper*.

Los trenes de audio y video son transportados en sesiones RTP separadas. Si se tienen múltiples sesiones RTP se las distingue por su dirección de transporte, esta es la dirección de red más los dos TSAP, uno para el RTP y otro para el RTCP.

2.3.2.7.1. Modo llamada directa (direct call) En esta modalidad de funcionamiento la señalización de llamada y el canal de control H.245 se establece directamente entre los terminales.Todos los mensajes de señalización Q.931 y H.245, son intercambiados directamente entre los terminales, además del intercambio de tramas RTP habitual. Si el terminal originante conoce la dirección de transporte del extremo llamado, éste puede establecer una llamada directamente con el otro participante de la comunicación. El *gatekeeper* y los canales RAS son opcionales. Cuando el *gatekeeper* está presente, el terminal llamante debe solicitar el servicio de resolución de dirección a este. Además el terminal llamado debe solicitar autorización al *gatekeeper* para aceptar la llamada. Este modelo no permite llevar un correcto registro de las llamadas para su facturación y para la correcta administración de los recursos. Por más detalles ver sección 2.3.5, figura 2.18.

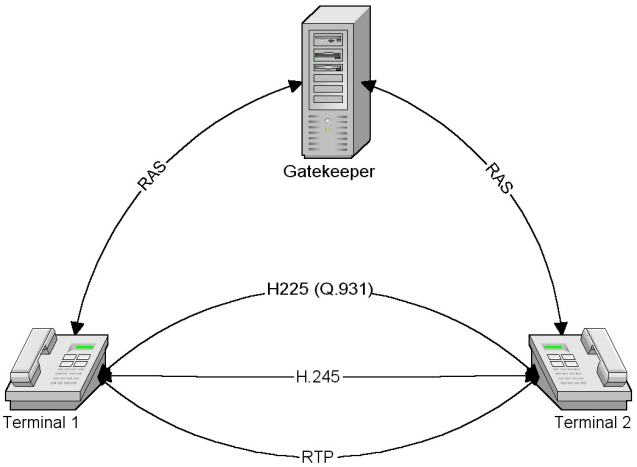

Figura 2.11: Llamada directa.

2.3.2.7.2. Modo llamada encaminada por gatekeeper (gatekeeper routed call) En este modo la señalización de llamada H.225/Q.931 es enrutada hacia los terminales a través del *gatekeeper* y el canal de control H.245 podría serlo. Todos los mensajes de señalización pasan a través del *gatekeeper*, el uso del protocolo RAS es necesario. Este modelo permite a los terminales acceder a servicios brindados por el *gatekeeper*, como resolución de direcciones y encaminamiento de llamadas. También permite al *gatekeeper* forzar el uso del control de admisión para los terminales, así como reparto y administración de ancho de banda sobre su zona de control.

Por las características mencionadas este modelo si es aceptable por los ISP y *carriers* de gran escala, permite una mejor administración de recursos, así como la fácil implementación de funciones de *accounting* y *billing*. Permite controlar los recursos usados por cada usuario y por lo tanto poder facturar correctamente los mismos.

Cuando se utiliza la señalización de llamada encaminada por *gatekeeper* se dispone de dos métodos para encaminar el canal de control H.245. En el primero de ellos el canal de control H.245, se establece directamente entre los terminales.

A modo de ejemplo se puede ver en la figura 2.22 el establecimiento de una llamada en este caso, y en las siguientes figuras en la misma sección se puede ver el intercambio de mensajes de control, de paquetes RTP y la liberación de la llamada.

En el segundo método, el canal de control H.245 es encaminado entre los terminales a través del *gatekeeper*.

Este método permite al *gatekeeper* reencaminar el canal de control H.245 a un MCU cuando una conferencia pasa de conferencia punto a punto a conferencia multipunto. Cuando se utiliza la señalización de llamada directa, el canal de control H.245 sólo puede ser conectado directamente entre los terminales.

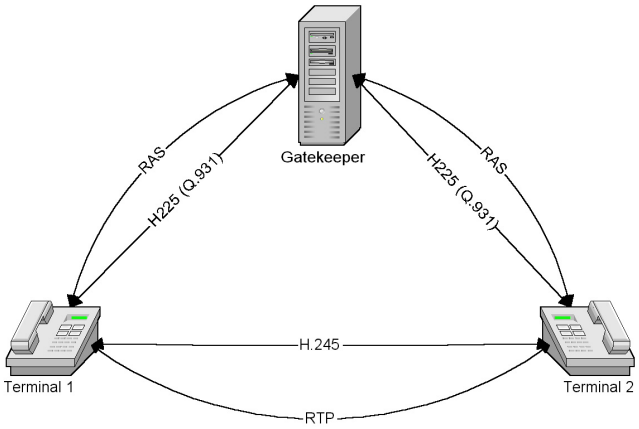

Figura 2.12: Llamada encaminada por *gatekeeper*, sólo H.225/Q.931

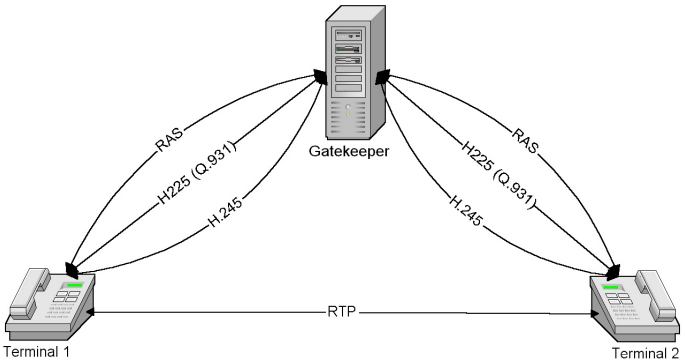

Figura 2.13: Llamada encaminada por *gatekeeper*, H.225/Q.931 y H.245

2.3.3. Servicios suplementarios en H.323

Los servicios suplementarios constituyen un aspecto fundamental en el desarrollo de la telefonía IP debido a las posibilidades de negocios que conllevan. Según la definición que adopta la ITU en la sección 2.4 de su recomendación I.210[I210], un servicio suplementario modifica o complementa a un servicio de telecomunicación básico. En un principio, la ITU concibió los servicios suplementarios en telefonía IP de manera análoga a los de la PSTN. Así surgieron la serie de recomendaciones H.450 que implementan muchos de los servicios suplementarios disponibles en la telefonía tradicional en un ambiente VoIP (transferencia de llamada, llamada en espera, etc). La búsqueda de la interconexión con las redes existentes y la visión del negocio de las compañías telefónicas explicaban dicha elección.

Con las nuevas versiones de la norma se han incorporado nuevos modelos para brindar servicios suplementarios dando la posibilidad a los im-

plementadores de utilizar las distintas soluciones de acuerdo a sus requerimientos.

A seguir se da una breve reseña de la forma de implementar servicios suplementarios en H.323[S323].

2.3.3.1. Control distribuido - H.450

Cada recomendación H.450 define la semántica y la sintaxis de un servicio suplementario. Semánticamente los servicios son idénticos a los definidos en la PSTN y se implementan mediante una serie de funciones genéricas definidas en H.450.1 para el intercambio de mensajes definidos mediante ASN.1. La tabla 2.9 muestra los distintos tipos de servicios y las recomendaciones donde se definen.

Recomendación	Servicio suplementario	Subfunciones
H.450.1	Funciones genéricas para servicios suplementarios en H.323	
H.450.2	Transferencia de llamada	
H.450.3	Desviación de llamada	Reenvío de llamada incondicional Reenvío de llamada en caso de ocupado Reenvío de llamada en caso de ausencia de respuesta Reflexión de llamada
H.450.4	Retención de llamada	Retención de llamada en el extremo cercano Retención de llamada en el extremo distante
H.450.5	Depósito y extracción de llamada	Depósito dirigido de llamadas Depósito de llamadas de un grupo Extracción de llamada de aviso
H.450.6	Llamada en espera	
H.450.7	Indicación de mensaje en espera	
H.450.8	Identificación de nombres	
H.450.9	Completitud de llamadas	
H.450.10	Ofrecimiento de llamadas	
H.450.11	Intrusión de llamadas	
H.450.12	Información común	

Cuadro 2.9: Serie de Recomendaciones H.450

La arquitectura de servicios suplementarios H.450 se basa en el concepto de descentralización de funciones. Las entidades se comunican en-

tre sí utilizando señalización H.450 sin la necesidad de un mecanismo de control centralizado en la red. Los servicios se implementan mediante la interacción de los distintos terminales y la inteligencia se distribuye entre los distintos componentes. Esto implica que en todo servicio suplementario brindado con H.450 todos los terminales participantes deben disponer e implementar el recurso. La incorporación o modificación de un servicio determina el cambio en cada uno de los terminales participantes. De todas maneras, se prevé alguna centralización del servicio mediante el uso de servidores de facilidades (tipo especial de terminal H.323/H.450), o de *proxies* de H.450 que actúan en representación de los terminales para brindar el servicio a terminales tontos. Estos *proxies* de H.450 pueden ser colocados, por ejemplo, en el *gatekeeper*.

Aparte de los servicios definidos, se prevén distintos mecanismos para una fácil extensión de ese conjunto. A modo de ejemplo, a las APDU de H.450 se les puede agregar información específica de cada fabricante (NonStandarData) permitiendo definir servicios propietarios, o nuevas variaciones de los ya estandarizados. H.450.1 define en forma genérica como proceder ante H.450 APDU no conocidas o soportadas. De esta manera se permite interoperabilidad entre los terminales que poseen un conjunto distinto de facilidades y a su vez permite un desarrollo de servicios no uniforme entre los terminales sin la necesidad de que todos tengan que soportar todas las facilidades al mismo tiempo.

En el diseño de H.450 uno de los objetivos básicos era simplificar la interconexión de facilidades con redes privadas conmutadas (QSIG) y públicas (ISDN).

H.450 determina una arquitectura modular basada en una visión orientada a objetos, es decir, cada servicio suplementario es una entidad independiente.

Como se ve en la figura 2.14, H.323/H.450 permite la separación entre el control de la llamada básica y los servicios suplementarios. Los mensajes de H.225 son encaminados hacia la entidad que atiende la llamada básica, donde desencadenan las acciones y cambios de estado necesarios. La información de H.450, que se encapsula dentro de los mensajes H.225, es pasada a la entidad que implementa H.450.1. Se cumplen los servicios genéricos por parte de H.450.1 y las operaciones ROS (*Remote Operation Service*) son pasadas a la entidad de servicio suplementario respectiva. La entidad que implementa H.450.1 es donde se coordinan las facilidades simultáneamente activadas o se bloquean en caso de interacción entre ellas. Cada entidad de S.S. (servicio suplementario) posee su propia máquina de estados que define la semántica del servicio. Esta es invocada cuando se pide algo relacionado con el S.S. en cuestión. En las recomendaciones H.450.x las maquinas de estado se especifican a través de diagramas SDL.

2.3.3.2. Control por estímulo - H.323 Anexo L

H.323 Anexo L utiliza el modelo de control de terminal de H.248, para el control de los servicios suplementarios. Describe procedimientos de señalización (*Stimulus Signalling*) entre los terminales H.323 Anexo L y una entidad que actúa de servidor de facilidades (feature server) estableciendo una relación maestro-esclavo entre ellos y estructura centralizada de con-

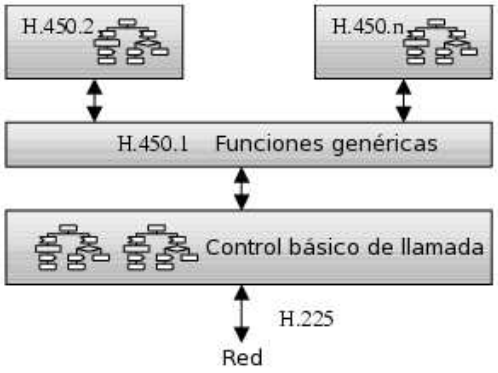

Figura 2.14: Arquitectura H.323/H.450 en los terminales

trol. Para esto se encapsulan mensajes H.248 en los mensajes H.225. Este método permite al proveedor de servicios implementar nuevos servicios suplementarios para los terminales sin tener que realizar cambios en el software del terminal. El *Feature Server* puede estar colocado con el *gatekeeper*, ver figura 2.15.

Los servicios pueden ser provistos por uno o más *Feature Servers*. Para asegurar interoperabilidad, señalización H.225 es usada para el control de la llamada básica, y todas las manipulaciones de las tramas de datos (*media streams*) son hechas utilizando H.245 o procedimientos *fast-connect* (ver sección 2.3.6.2, en "procedimientos de conexión rápida"). Métodos basados en la recomendación H.248 se usan para manipular las entidades físicas de los terminales, como son el parlante, la horquilla, etc.

Este protocolo puede soportar el modelo de señalización directa como el de reencaminamiento por el *gatekeeper*.

El *Feature Server* tiene acceso a la señalización H.323, lo cual le brinda información sobre el estado de la llamada que le puede ser útil para servicios particulares, permitiéndole modificar los trenes de medios usando H.245 o señalización de *fast-connect* (por fast connect ver sección 2.3.6.2).

Anexo L explícitamente excluye todas las partes de H.248 relacionadas al control de los trenes de medios que se hacen con H.245 o *fast-connect*.

El *Feature Server* o el *gatekeeper* son responsables de brindar una función de *proxy* para el manejo de los procedimientos H.450 en la red en representación del terminal. En este caso, el *Feature Server* se convierte en el terminal para las operaciones H.450 e implementa todos los servicios y las máquinas de estado necesarias. La interacción con el usuario se produce a través de la interfase de usuario, la cual el *gatekeeper* es capaz de controlar por medio de la stimulus signalling.

El *Feature Server* se parece mucho a los componentes del MGC no relacionados con con el control de trenes de medios y es capaz de controlar distintos elementos de la interfaz de usuario. Por ejemplo: escribir en un display de texto, recibir entrada de usuario: dígitos, texto,teclas especiales (horquilla, etc), solicitar que se emitan distintos tonos, etc.

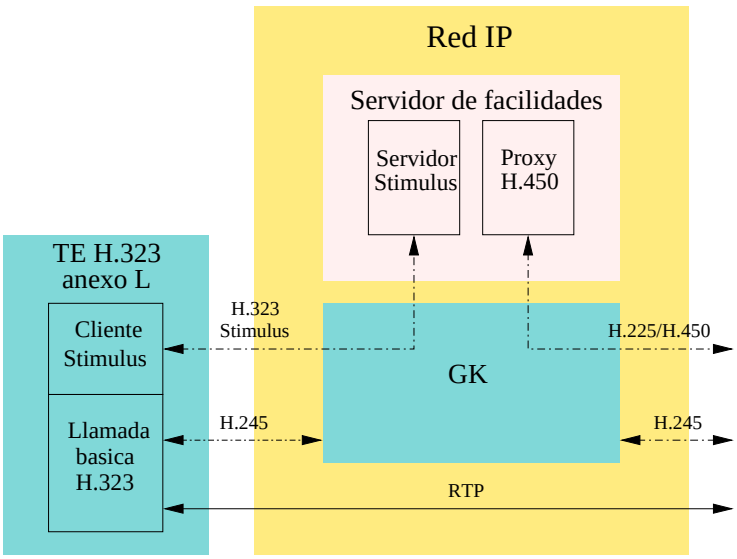

Figura 2.15: H.323 Anexo L

Con este método se tiene poca inteligencia en los terminales con la lógica de los servicios y los procedimientos semánticos definidos en el servidor central. Este modelo no define ni estandariza la semántica de los servicios dando libertad a los implementadores para desarrollar los servicios. Lo que brinda son las herramientas para desarrollar los servicios sin ocuparse de cuáles deben ser estos. Esta visión concuerda con la postura de SIP frente a los servicios suplementarios. Se le critican dificultades de escalabilidad debido a la capacidad limitada de procesamiento determinada por la centralidad.

2.3.3.3. Control de capa de aplicación - H.323 Anexo K

Este anexo agrega un plano de servicios sobre el plano de control de llamadas. Se establece una sesión de control de servicios después de intercambiar la información necesaria (*sessionID*, URL para el control de servicios, etc) en RAS u otros mensajes de H.225. El canal de control de servicios puede ser utilizado tanto para servicios relacionados a una llamada como los restantes y puede ser abierto entre dos terminales, así como entre un terminal y la red (por ejemplo el *gatekeeper*).

Al ser este canal independiente de los servicios, ningún servicio específico ha sido definido. Los datos intercambiados a través del canal tienen un sentido informativo (interfaz de usuario) y serán seguidos de las acciones apropiadas (por ejemplo, invocación de operaciones H.450 a través de señalización *proxy*) en el plano de señalización de llamada cuando sea apropiado.

Pudiendo utilizar muchos protocolos en dicho canal este anexo describe el uso de HTTP para ofrecer, activar y seleccionar los servicios. HTTP es abierto, flexible, se puede usar a través de firewalls y además es un protocolo muy difundido y probado. La lógica de los servicios se describe en páginas

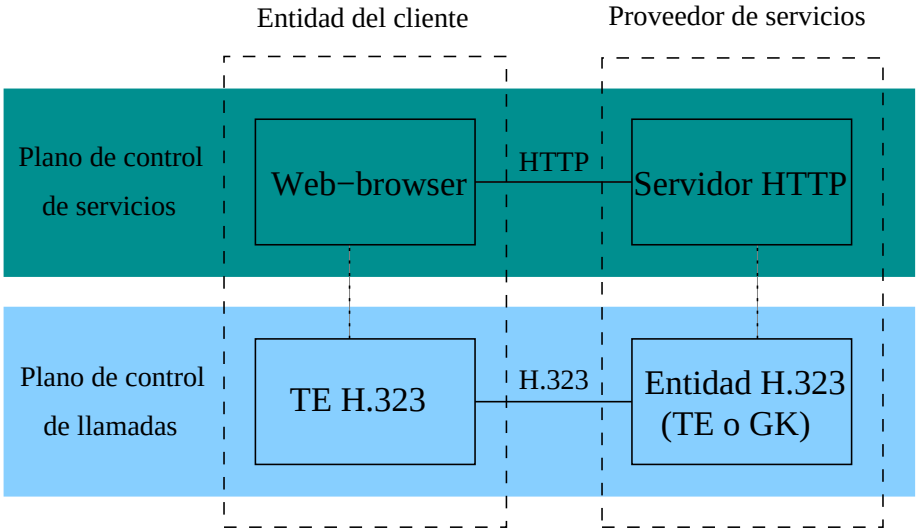

Figura 2.16: Secuencia de intercambio de mensajes en H.323 Anexo K

HTML, scripts, etc, que se transfieren mediante HTTP. Entre los ejemplos de aplicaciones que se pueden desarrollar se encuentran: páginas XML que incluyan Java y *scripts*, bajada de tonos y anuncios para los clientes, subida de CPL (*Call Processing Language*) desde el cliente al *gatekeeper*, etc.

A través de la señalización RAS el *gatekeeper* informa a los terminales la presencia del servidor WEB a los que se deben conectar.

La idea central es aligerar los terminales de forma de poder brindar servicios para terminales más tontos que los típicos de H.323, sin necesidad de actualizar el protocolo o los terminales. Por ejemplo, se podría hacer un terminal anexo K que implementara solamente RAS e incluyera un cliente de HTTP (*browser*) y a través del cual los usuarios se conectaran a una página y pudieran, por ejemplo, enviar un fax.

Al igual que con el anexo L no se define la semántica de ningún servicio, dando libertad a los desarrolladores para idear e implementar los servicios que precisen.

2.3.4. Gatekeeper

El *gatekeeper* o controlador de acceso - como se lo denomina en castellano - es la entidad o componente de la arquitectura H.323 encargada de cumplir con diferentes tareas de control y administración de la zona. Estas tareas son realizadas brindando una serie de servicios y funciones a los *endpoints* que componen dicha zona, o sea, aquellos que están registrados con él.

Los servicios brindados incluyen; resolución de direcciones, administración de ancho de banda y control de admisión, a su vez puede llegar a brindar algunas funciones opcionales como; autenticación y autorización de usuarios, enrutamiento de señalización de llamadas, y otros que serán descritos a continuación.

A pesar de la importancia de estos servicios para la administración de un sistema de voz sobre IP donde intervienen varias decenas a centenas de terminales, la entidad *gatekeeper* es opcional según la recomendación.

Aunque, es obligatorio para un *endpoint* H.323[2], poseer la capacidad de comunicación con esta entidad. O sea, soportar los mensajes RAS H.225 y procedimientos establecidos en H.323, que le permitirán la comunicación con el *gatekeeper*.

El *gatekeeper* puede coexistir físicamente con otras entidades de la arquitectura H.323 o de la red, por ejemplo con un *gateway*, sin embargo lógicamente debe estar separado de estas últimas.

Por otro lado, las funciones y servicios de *gatekeeper* deben ser brindadas por una única entidad en un momento dado dentro de una zona. Aunque es posible, según la recomendación, que estas sean ofrecidas por varios dispositivos distintos dentro de la zona. Por ejemplo; un dispositivo A se encargaría de la mensajería RAS con los *endpoints*, el registro de los mismos y la resolución de direcciones, mientras que otro B, diferente, se encargaría de la mensajería y enrutamiento H.225/Q.931 únicamente, debido a la complejidad y recursos de procesamiento que esta última necesita. Juntos A y B brindan la funcionalidades de un *gatekeeper*.

Las comunicaciones entre *endpoints* y *gatekeeper* solicitando sus servicios están definidas por el protocolo RAS en la recomendación H.225.

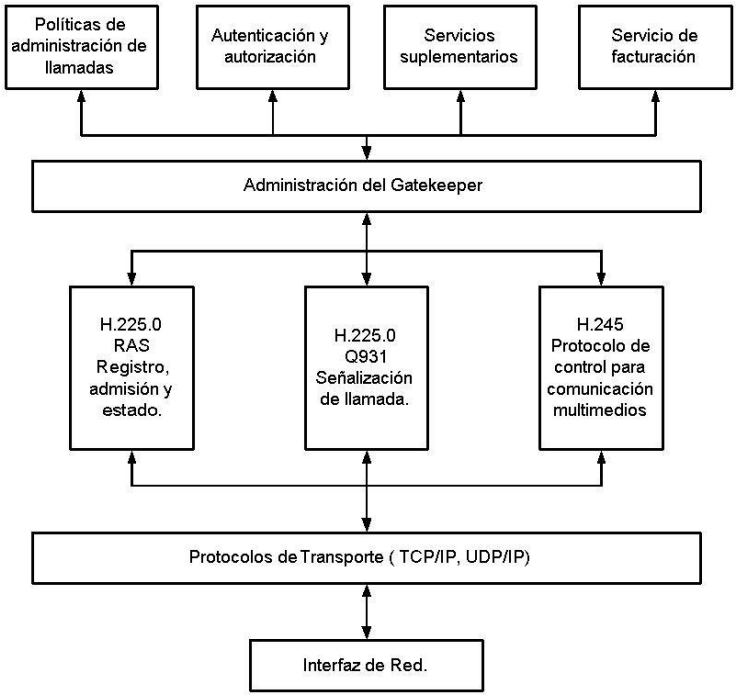

Figura 2.17: Componentes del *gatekeeper*

En la figura 2.17 también se ven otros protocolos involucrados en el fun-

[2]Terminal de audio y/o video, Gateway o MCU que cumple con la recomendación H.323.

cionamiento del *gatekeeper* y algunos servicios.

En lo que sigue se pasa a detallar los mencionados servicios obligatorios del *gatekeeper* y la funciones opcionales.

2.3.4.1. Funciones obligatorias del gatekeeper

- Resolución de direcciones

Implica la capacidad del *gatekeeper* para realizar la traducción de un identificador de *endpoint* -denominado alias- en una dirección de red y de transporte. Esta función es necesaria debido a que normalmente no conocemos direcciones de transporte para localizar recursos en una red de paquetes.

Las comunicaciones originadas dentro de una red H.323 usan una dirección alias para ser encaminadas al *endpoint* destino, mientras que llamadas originadas fuera de la red H.323 deben usar un número telefónico de acuerdo con E.164, número que llega al *gateway* o es generado desde este. Entonces, el *gatekeeper* debe traducir estos número y alias en la dirección de transporte correspondiente al *endpoint* destino, por ejemplo una dirección IP:puerto en una red de tipo TCP/IP. Así, el *endpoint* destino puede ser accedido usando esta dirección de red.

Esta previsto el uso de direcciones alias como; URLs, e-mails, tramas de caracteres (hasta 256), cadenas de dígitos genéricos o también cadenas de dígitos cumpliendo con la E.164.

Se especifica que la obtención de una dirección de transporte debe realizarse a partir de una tabla modificable mediante el registro de los terminales, aunque se deja la posibilidad de utilizar otros métodos.

Durante el registro del *endpoint*, mediante un mensaje RRQ (*Registration Request*), el *gatekeeper* recibe la dirección de transporte del *endpoint* en el campo *callSignalAddress* y el alias a utilizar en *terminalAlias*, siendo sólamente el primero de estos obligatorio. Si el segundo campo no está presente en el RRQ el *gatekeeper* debe asignar uno en su respuesta RCF (*Registration Confirm*). De esta forma el controlador de acceso puede ir construyendo su tabla de conversión de direcciones.

- Control de admisión

Los puntos extremos en una red H.323 pueden pasar por un proceso de control de admisión efectuado por el *gatekeeper*. Se usa en este caso los mensajes H.225 RAS; solicitud de admisión ARQ (*Admission Request*), confirmación ACF (*Admission Confirm*) o rechazo ARJ (*Admission Reject*). Es obligatorio para el *gatekeeper* soportar estos mensajes.

El fabricante está en libertad de elegir el criterio que sea necesario para admitir o rechazar estas solicitudes. Es posible entonces restringir el número de conexiones H.323 en la red de forma de acotar el ancho de banda que éstas utilizan. Podría rechazarse una solicitud simplemente porque el terminal no se encuentra registrado, o también podrían implementarse funciones más complejas que rechazarían llamadas de acuerdo con una lista de restricciones configurable que posee el usuario o terminal que origina la llamada. Pero también se admite una función de control de admisión nula, que siempre admite nuevas solicitudes.

- Control de ancho de banda

El *gatekeeper* debe soportar los mensajes H.225 RAS; solicitud de cambio de ancho de banda BRQ (*Bandwidth Change Request*), confirmación BCF (*Bandwidth Change Confirmation*), o rechazo BRJ (*Bandwidth Change Reject*). O sea, debe poder interpretarlos y responder.

Mediante estos mensaje es posible implementar una función de administración de ancho de banda. Pero es preciso aclarar lo siguiente; este control será efectuado luego que un conexión está establecida. Este grupo de mensajes BRQ/BCF/BRJ puede ser utilizados por un *endpoint* o *gatekeeper* para modificar el ancho de banda durante el trascurso de una conexión. Para acotar o solicitar ancho de banda durante el inicio de una conexión se utiliza el campo *bandwidth* en los mensajes ARQ/ACF.

La mencionada administración en realidad es opcional (ver 2.3.4.2) y perfectamente pude implicar la utilización de una función nula, que siempre dé como respuesta BCF a las solicitudes del cambio de ancho de banda.

- Administración de zona

Es obligatorio para el *gatekeeper* proveer las funciones arriba mencionadas (conversión de direcciones, soporte de mensajes ARQ/ACF/ARJ, y soporte para mensajes BRQ/BCF/BRJ) a todos los terminales que se encuentren registrados en este, o sea, los cuales componen la zona.

La forma en que deben ser soportados estos mensajes está especificado en la recomendación H.323, así como la estructura de los mensajes se encuentra en la recomendación H.225.0. Más adelante (2.3.5) se describen algunos casos de establecimiento y desconexión de llamadas como forma de ejemplo.

2.3.4.2. Funciones opcionales del gatekeeper

La siguiente lista de funciones son habilitadas en forma opcional por la recomendación H.323;

1. Señalización de control de llamada - El *gatekeeper* puede encaminar la señalización de llamada (mensajes H.225/Q.931) entre puntos extremos H.323. O sea, cada *endpoint* envía sus mensajes de señalización de llamada H.225.0 al *gatekeeper*, quien luego los encamina al *endpoint* destino. Estamos en el modo "*gatekeeper-routed call*", que ya habíamos mencionado. Esto permite al controlador de acceso una mejor administración de las llamadas en su zona. Pero también puede permitirse que los *endpoints* envíen la señalización de llamada H.225 directamente entre sí. Estamos en lo que se denomina modo "*direct call*".

2. Autorización de llamadas - Si el *gatekeeper* se encuentra trabajando en modo *gatekeeper-routed*, éste puede aceptar o rechazar solicitudes de inicio de llamada (mensaje H.225/Q.931 *setup*), de acuerdo con algún criterio preestablecido. Pueden tomarse como razones de rechazo la existencia de restricciones al *endpoint* que origina la llamada, en el

acceso a comunicaciones vía un determinado gateway o hacia un *endpoint* dado. Aunque la recomendación deja claro que también pueden utilizarse otras razones de rechazo.

3. Administración de llamadas - Otra posibilidad que ofrece el hecho de estar utilizando el modo de señalización encaminada por *gatekeeper* es implementar esta función de administración de llamadas. La cual implica mantener información, en una lista por ejemplo, de todas las comunicaciones H.323 que están en curso en cada momento en la zona. Esta información es necesaria para la función de administración de ancho de banda, por ejemplo, o para saber si un terminal está ocupado y a partir de esto lanzar un función de desvío de llamada.

4. Facturación de llamadas - Una vez que el *gatekeeper* detectó la finalización de una llamada este puede notificar a la entidad encargada de administrar las cuentas de usuarios sobre los detalles de esta última llamada; duración, número destino y origen, recursos de ancho de banda usados, etc. A partir de esto incrementar contadores de usuario, imprimir un ticket, generar un registro en un archivo de tipo log, etc.

5. Administración del ancho de banda - Como ya se mencionó, mediante el protocolo H.225/Q.931 de señalización, el *gatekeeper* puede controlar el número de comunicaciones simultaneas y el uso de ancho del banda de estas. Tiene la posibilidad de limitar el requerimiento de ancho de banda de una llamada nueva, incluso rechazarla si determina que no hay más ancho de banda disponible para comunicaciones H.323. Además esta función puede realizarse sobre una comunicación en curso, cuando el terminal solicita mayor ancho de banda.

6. Servicios suplementarios - Como ya se mencionó, están previstos en la serie de recomendaciones H.450.x y se efectúan en conjunto con los *endpoints*.

7. Servicio de directorio - El *gatekeeper* pude llevar una base de datos con información de los usuarios de *endpoints* en su zona. Esto le permite mantener un servicio de directorio telefónico o guía telefónica, para ayudar en la búsqueda de usuarios.

2.3.4.3. Protocolo RAS en el gatekeeper

Este protocolo, cuyos mensajes se encuentran especificados en la recomendación H.225, es usado para comunicaciones entre *gatekeeper* y *endpoints* con la finalidad de cumplir con las funciones mencionadas en 2.3.4.1 y opcionalmente también es usado por *gatekeepers* de diferentes zonas para resolver una comunicación entre terminales registrados con estos.

Los mensajes RAS son enviados sobre una canal no fiable, UDP/IP por ejemplo. Por lo tanto durante el intercambio de estos mensajes el *gatekeeper* debe iniciar *timers* y reintentar en caso que estos expiren, es decir, queda para el fabricante o implementador del *gatekeeper* asegurar la fiabilidad del canal RAS.

2.3.4.4. Protocolo H.225 en el gatekeeper

Este es usado para establecer una conexión entre dos puntos extremos, sobre la cual serán transportados en tiempo real, audio, video o datos. La señalización de llamada involucra el intercambio de mensajes del protocolo H.225, sobre un canal de señalización de llamada confiable. En el caso de una red basada en IP, el canal será soportado sobre TCP.

Como ya se mencionó existen dos métodos de trabajo, en caso de la presencia de un *gatekeeper* en la red, para el intercambio de mensajes de señalización de llamada H.225.0. Un método directo y otro encaminado por *gatekeeper*. En el primero caso los mensajes H.225.0 son intercambiados directamente entre los puntos extremos, mientras que en el segundo método, los mensajes son enrutados a través del *gatekeeper*. Para ello, el terminal que origina envía los mensajes a un canal de señalización de llamada del *gatekeeper*, mientras que éste encamina los mensajes al canal de señalización de llamada del punto extremos destino. El método a emplear es transmitido del *gatekeeper* al punto extremo durante el intercambio de mensajes RAS, más precisamente en el mensaje de confirmación de admisión (ACF).

La recomendación H.225[3] indica que el *gatekeeper* podrá modificar, para luego retransmitir a su destinatario, terminar o iniciar mensajes Q.931 en el caso de estar operando en modo *gatekeeper-routed*. Esto último permite al *gatekeeper* brindar otro tipo de servicios a los establecidos como obligatorios y opcionales de este (2.3.4.1 y 2.3.4.2).

2.3.4.5. Protocolo H.245 en el gatekeeper

Este protocolo consiste en el intercambio de mensajes extremo-a-extremo entre los puntos extremos que se encuentran en una comunicación. Otra vez en el modo *gatekeeper-routed*, estos menajes podrían pasar a través de la mencionada entidad. Para ello y en forma similar al resto de los protocolos, se utiliza un canal de control H.245 que se encuentra permanentemente abierto durante una comunicación, al contrario de los canales de intercambio de tramas de medios.

2.3.5. Procedimientos de conexión

En lo que sigue se muestra con más detalle los pasos o fases involucrados en la creación de una comunicación H.323, el establecimiento de la comunicación de tramas de medios (voz o video) y la finalización de la llamada. Se asume que la red solo contiene dos *endpoints* Terminal 1 y Terminal 2, además del *gatekeeper* y se muestran dos formas de funcionamiento del *gatekeeper*, primero llamada directa y luego llamada encaminada por *gatekeeper* [4].

2.3.5.1. Llamada directa.

- Establecimiento de llamada

[3] En el punto 7.1 Use of Q.931 messages.

[4] En el ejemplo se encaminan los mensajes H.225 de señalización de llamada y no los mensajes H.245.

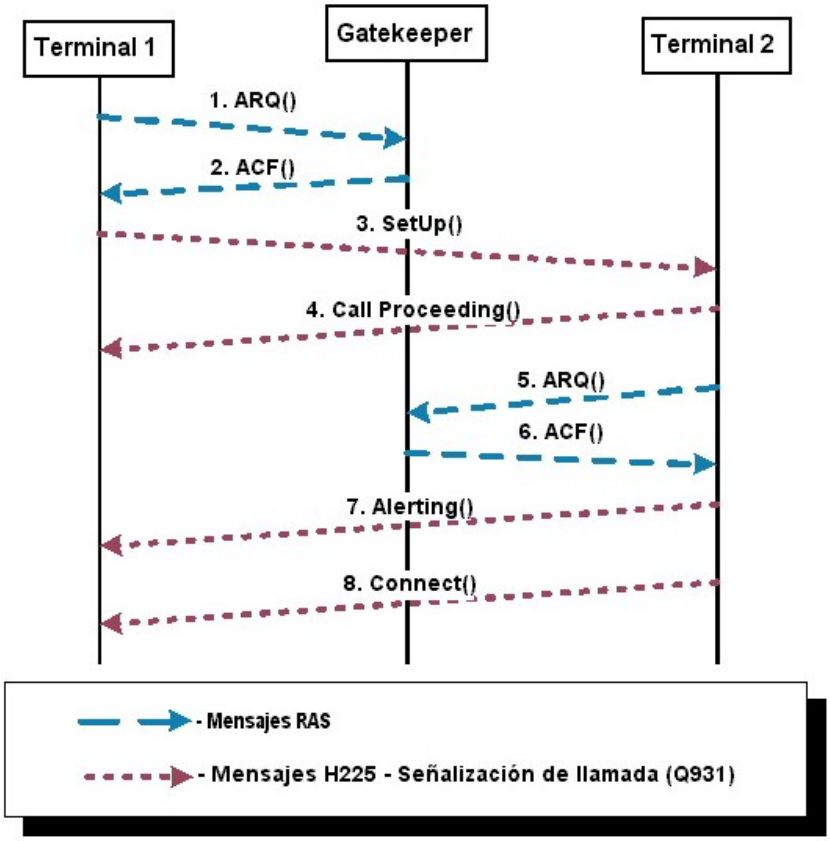

Figura 2.18: Establecimiento de llamada

(1) El Terminal 1 envía el mensaje RAS ARQ al *gatekeeper* solicitando autorización para el inicio de una nueva comunicación. Además, Terminal 1 solicita el uso del modo directo o modo encaminado por *gatekeeper* de señalización de llamada.

(2) El *gatekeeper* confirma con el mensaje RAS ACF la admisión del Terminal 1. Además, este indica en el mensaje ACF la utilización del modo directo.

(3) Terminal 1 envía un mensaje H.225 de señalización de llamada, de inicio (*setup*), solicitando a Terminal 2 una conexión.

(4) Terminal 2 responde con otro mensaje H.225 de señalización de llamada (*call proceeding*).

(5) Terminal 2 envía un mensaje ARQ al *gatekeeper* utilizando un canal RAS, también solicitando sea admitida una nueva comunicación de este.

(6) El *gatekeeper* responde a Terminal 2 con una confirmación, mensaje RAS ACF.

(7) Terminal 2 avisa a Terminal 1 sobre el establecimiento de la conexión con un mensaje H.225 de alerta (*alerting message*)

(8) Terminal 2 confirma el establecimiento de la conexión, con el envío del mensaje H.225 de conexión (*connect*).

- Señalización de control

A continuación se intercambian los mensajes de control H.245 siguientes:

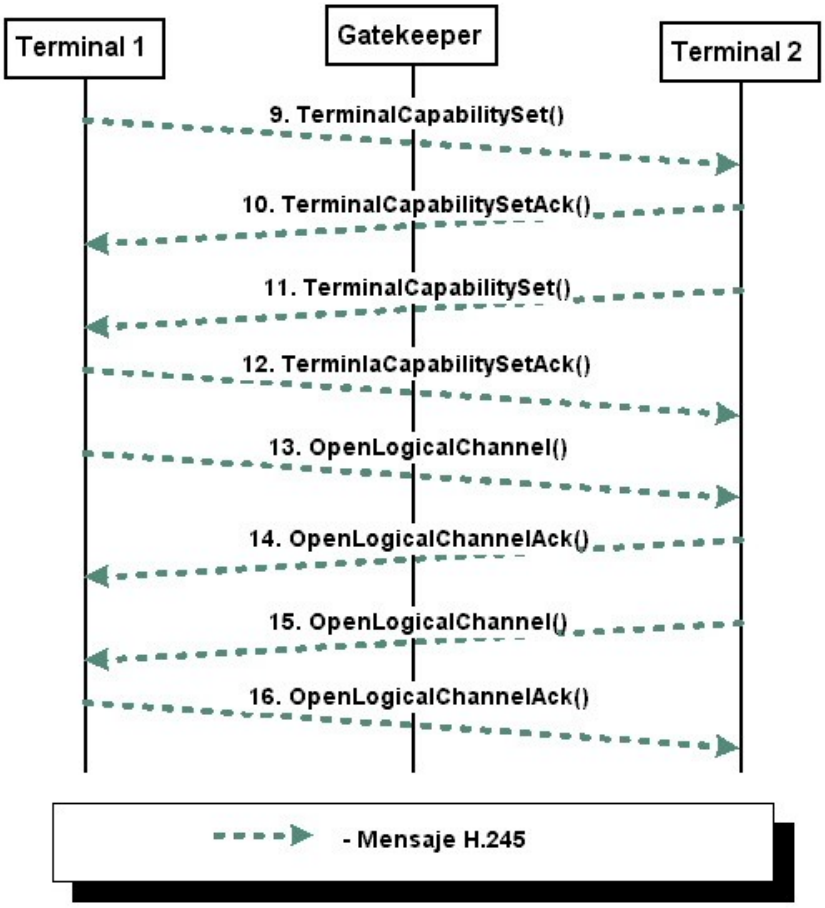

Figura 2.19: Control H.245

(9) Un canal de control H.245 es establecido entre Terminal 1 y Terminal 2. En él, Terminal 1 envía el mensaje *TerminalCapabilitySet* a Terminal 2 para intercambiar sus capacidades de procesamientos de tramas de medios.

(10) Terminal 2 contesta reconociendo las capacidades de Terminal 1 con el mensaje H.245 *TerminalCapabilitySetAck*.

(11) Ahora es el turno de Terminal 2 para su envío de capacidades, con el mensaje H.245 *TerminalCapabilitySet* a Terminal 1.

(12) Terminal 1 reconoce el mensaje de Terminal 2, con el propio H.245 *TerminalCapabilitySetAck*.

(13) Terminal 1 envía un mensaje H.245 *openLogicalChannel* con el fin de abrir un canal de medios con Terminal 2. En este mensaje se encuentra la dirección de transporte del canal RTCP.

(14) Terminal 2 reconoce el establecimiento del canal lógico unidireccional desde Terminal 1 a Terminal 2 enviando el mensaje *openLogicalChannelAck*. Incluido en este mensaje se envía: la dirección de transporte de RTP asignada por Terminal 2 para ser usada por Terminal 1 para el envío de tramas de medios RTP y la dirección de transporte de RTCP recibida de Terminal 1 en (13).

(15) Ahora Terminal 2 abre un canal de medios con Terminal 1 enviando el mensaje H.245 *openLogicalChannel*. En forma similar, va en este mensaje la dirección de transporte del canal RTCP.

(16) Ahora Terminal 1 reconoce el establecimiento del canal lógico unidireccional (Terminal 2 a Terminal 1) con el envío del mensaje H.245 *openLogicalChannelAck*. En éste se incluye: la dirección de transporte del canal RTP asignado por Terminal 1 para el envío de tramas de medios RTP por Terminal 2 y la dirección del canal RTCP recibida en (15).

A partir de este momento queda establecido una comunicación bidireccional de tramas de medios.

- Flujo de paquetes de audio.

Se llega a la comunicación de tramas conteniendo información de audio (por ejemplo).

(17) El Terminal 1 envía las tramas de medios encapsuladas en RTP a Terminal 2.

(18) El Terminal 2 envía las tramas de medios encapsuladas en RTP a Terminal 1.

(19) Terminal 1 envía mensajes RTCP a Terminal 2.

(20) Terminal 2 envía mensajes RTCP a Terminal 1.

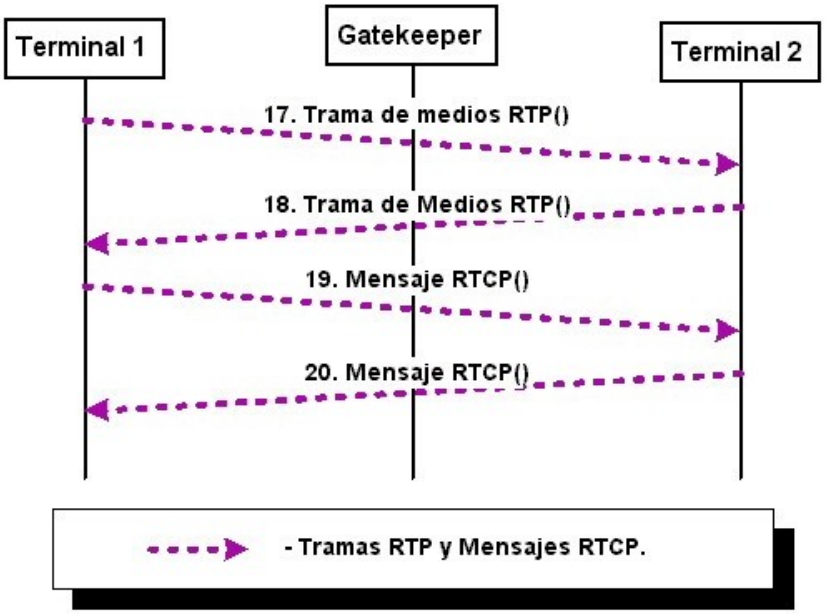

Figura 2.20: Intercambio de paquetes de medios.

- Liberación de la llamada

(21) Terminal 2 libera el canal H.245 abierto enviando el mensaje H.245 *EndSessionCommand* a Terminal 1.

(22) Terminal 1 también confirma la liberación enviando también el mensaje *EndSessionCommand* H.245 a Terminal 2.

(23) Terminal 2 completa la liberación de la llamada enviando el mensaje H.225 *Relese Complete* a Terminal 1.

(24) Ahora Terminal 1 y Terminal 2 desenganchan del *gatekeeper* enviando el mensaje RAS DRQ, notificando el fin de la comunicación.

(25) El *gatekeeper* desengancha a Terminal 1 y Terminal 2 y lo confirma enviando el mensaje RAS DCF a ambos.

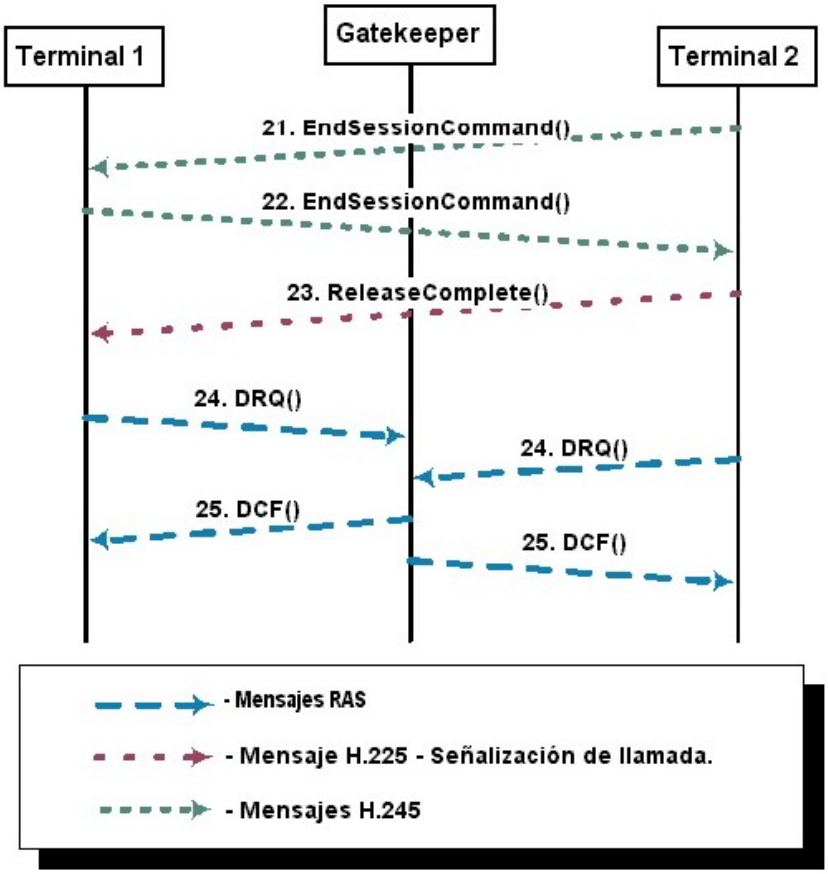

Figura 2.21: Liberación de llamada

2.3.5.2. Llamada encaminada por gatekeeper

- Establecimiento de llamada

(1) El Terminal 1 envía el mensaje RAS ARQ al *gatekeeper* solicitando autorización para el inicio de una nueva comunicación.

(2) El *gatekeeper* confirma con el mensaje RAS ACF la admisión del Terminal 1. Además, éste indica en el mensaje ACF la utilización del modo encaminado por *gatekeeper* e informa al Terminal 1 que la dirección de transporte del canal de señalización de llamada H.225/Q.931 es la de él mismo.

(3) Terminal 1 envía un mensaje H.225 de señalización de llamada, de inicio (*setup*), solicitando a Terminal 2 una conexión, hacia la dirección de transporte en el *gatekeeper*.

(4) El *gatekeeper* encamina el mensaje *setup* hacia el Terminal 2, sin antes haber realizado los cambios necesarios en dicho mensaje.

(5) El *gatekeeper* responde lo más pronto posible al Terminal 1 con un *call proceeding*.

(6) El Terminal 2 responde con otro mensaje H.225 de señalización de llamada *call proceeding*. A la llegada de este mensaje el *gatekeeper* puede tomar parte de la información que el mismo contiene y transmitirla al Terminal 1 utilizando un mensaje *facility*. Es el caso de recibir por ejemplo información de *fast connect*, o mensajes H.245 en túnel.

(7) Terminal 2 envía un mensaje ARQ al *gatekeeper* utilizando un canal RAS, también solicitando sea admitida una nueva comunicación de este.

(8) El *gatekeeper* responde a Terminal 2 con una confirmación, mensaje RAS ACF, anunciando el modo encaminado por *gatekeeper*.

(9) Terminal 2 avisa sobre el establecimiento de la conexión con un mensaje H.225 de alerta (*alerting message*), mediante el canal de señalización de llamada H.225/Q.931 previamente establecido.

(10) El *getekeeper* retransmite el mensaje *alerting* haciendo las modificaciones que sean necesarias, al Terminal 1.

(11) Terminal 2 confirma el establecimiento de la conexión, con el envío del mensaje H.225 de conexión (*connect*).

(12) El *gatekeeper* encamina el mensaje connect, haciendo las modificaciones que sean necesarias, al Terminal 1.

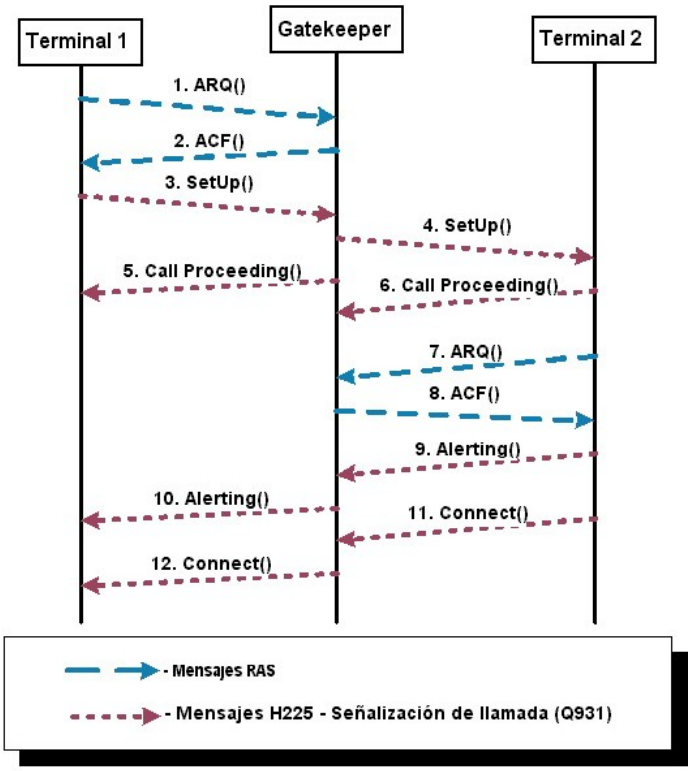

Figura 2.22: Establecimiento de llamada, encaminada por *gatekeeper*

- Señalización de control.

A continuación se intercambian los mensajes de control H.245, no se debe olvidar que el canal de señalización de llamada establecido con el *gatekeeper* por los dos terminales permanece abierto.

(13) Un canal de control H.245 es establecido entre Terminal 1 y Terminal 2. En él, Terminal 1 envía el mensaje *TerminalCapabilitySet* a Terminal 2 para intercambiar sus capacidades de procesamientos de tramas de medios.

(14) Terminal 2 contesta reconociendo las capacidades de Terminal 1 con el mensaje H.245 *TerminalCapabilitySetAck*.

(15) Ahora es el turno de Terminal 2 para su envío de capacidades, con el mensaje H.245 *TerminalCapabilitySet* a Terminal 1.

(16) Terminal 1 reconoce el mensaje de Terminal 2, con el propio H.245 *TerminalCapabilitySetAck*.

(17) Terminal 1 envía un mensaje H.245 *openLogicalChannel* con el fin de abrir un canal de medios con Terminal 2. En este mensaje se encuentra la dirección de transporte del canal RTCP.

(18) Terminal 2 reconoce el establecimiento del canal lógico unidireccional desde Terminal 1 a Terminal 2 enviando el mensaje *openLogicalChannelAck*. Incluido en este mensaje se envía: la dirección de transporte de RTP asignada por Terminal 2 para ser usada por Terminal 1 para el envío de tramas de medios RTP y la dirección de transporte de RTCP recibida de Terminal 1 en (17).

(19) Ahora Terminal 2 abre un canal de medios con Terminal 1 enviando el mensaje H.245 *openLogicalChannel*. En forma similar, va en este mensaje la dirección de transporte del canal RTCP.

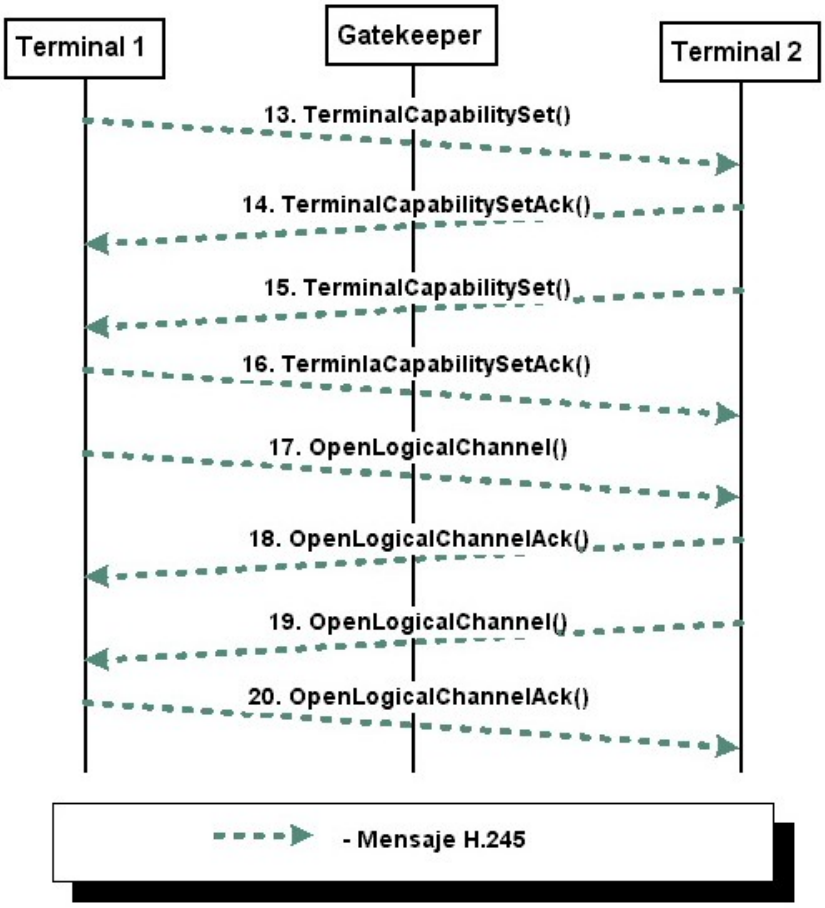

Figura 2.23: Control H.245, para llamada encaminada por *getekeeper*.

(20) Ahora Terminal 1 reconoce el establecimiento del canal lógico unidireccional (Terminal 2 a Terminal 1) con el envío del men-

saje H.245 *openLogicalChannelAck*. En este se incluye: la dirección de transporte del canal RTP asignado por Terminal 1 para el envío de tramas de medios RTP por Terminal 2 y la dirección del canal RTCP recibida en (19).

A partir de este momento queda establecido una comunicación bidireccional de tramas de medios.

- Flujo de paquetes de audio.

Se llega a la comunicación de tramas conteniendo información de audio.

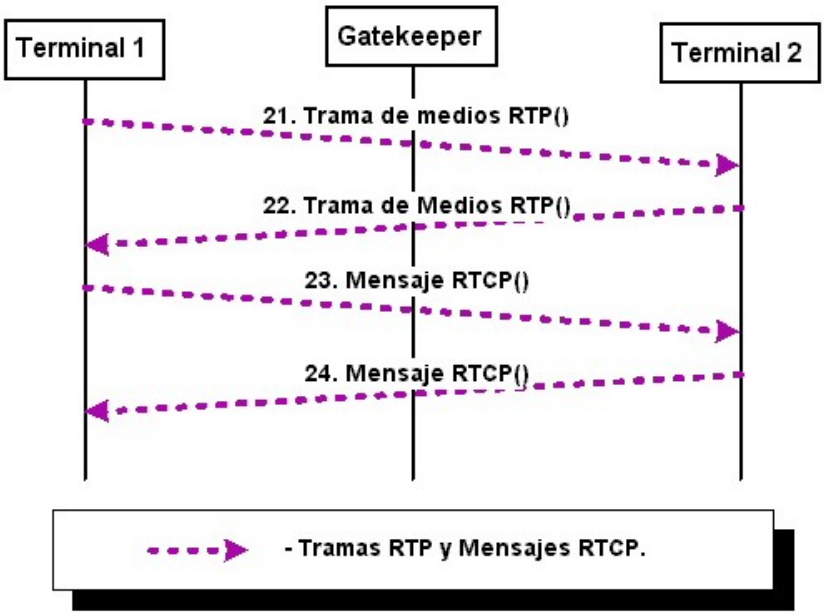

Figura 2.24: Intercambio de paquetes de medios, llamada encaminada por *gatekeeper*

(21) El Terminal 1 envía las tramas de medios encapsuladas en RTP a Terminal 2.

(22) El Terminal 2 envía las tramas de medios encapsuladas en RTP a Terminal 1.

(23) Terminal 1 envía mensajes RTCP a Terminal 2.

(24) Terminal 2 envía mensajes RTCP a Terminal 1.

- Liberación de la llamada.

(25) Terminal 2 libera el canal H.245 abierto enviando el mensaje H.245 *EndSessionCommand* a Terminal 1.

(26) Terminal 1 también confirma la liberación enviando también el mensaje *EndSessionCommand* H.245 a Terminal 2.

(27) Terminal 2 completa la liberación de la llamada enviando el mensaje H.225 *Relese Complete* hacia el *gatekeeper*, por el canal de señalización de llamada.

(28) El *gatekeeper* encamina el mensaje *Release Complete* hacia el Terminal 1, haciendo las modificaciones que sean necesarias.

(29y30) Ahora Terminal 1 y Terminal 2 desenganchan del *gatekeeper* enviando el mensaje RAS DRQ, notificando el fin de la comunicación.

(31y32) El *gatekeeper* desengancha a Terminal 1 y Terminal 2 y lo confirma enviando el mensaje RAS DCF a ambos.

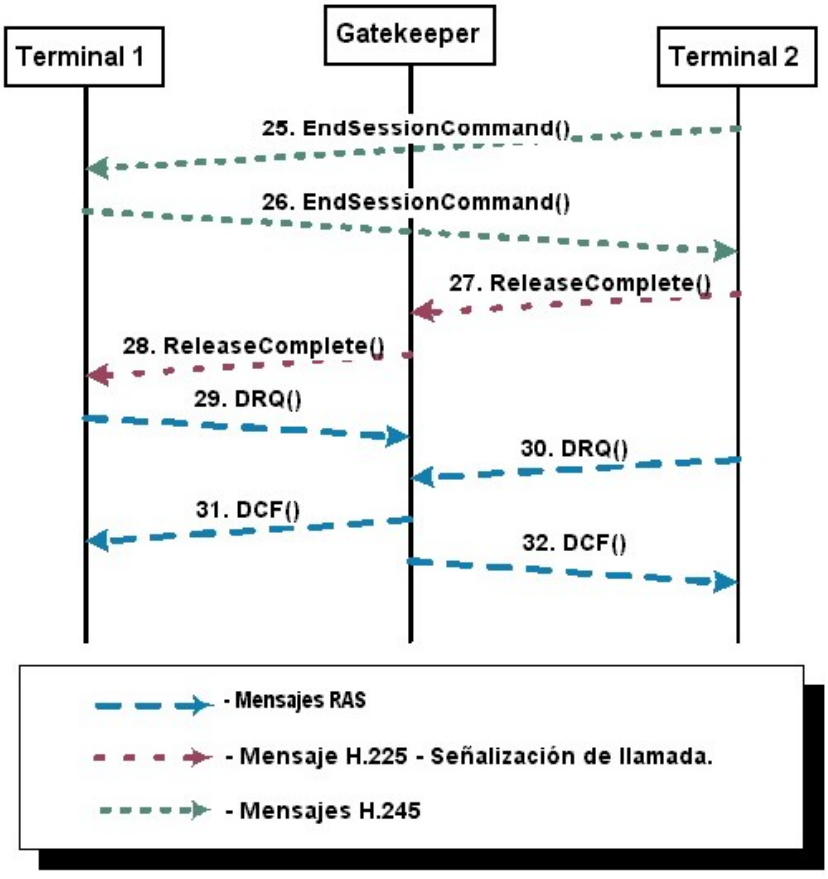

Figura 2.25: Liberación de llamada, encaminada por *gatekeeper*

2.3.6. Procedimientos de conexión rápida

En lo que sigue se detallarán un grupo de procedimientos previstos en la recomendación H.323 que permiten una mayor agilidad durante el establecimiento de una comunicación entre *endpoints*.

Estos procedimientos son opcionales según la versión de la recomendación, y pueden se aplicados en forma superpuesta para obtener un establecimiento aún más rápido. Si bien en general no consisten en la eliminación de los mensajes vistos en la sección 2.3.5, sino que el envío de alguno de éstos en forma anticipada encapsulados en otro mensaje, o la autorización previa.

2.3.6.1. Mensajes RAS con autorización previa

El *gatekeeper* tiene la posibilidad de autorizar en forma previa los mensajes ARQ de los *endpoint* de su zona de administración. Esto se realiza mediante el campo *preGrantedARQ* del mensaje RAS RCF (*RegistrationConfirm*). El mismo es una lista de valores booleanos, cada uno de ellos autorizando o negando un tipo de evento a producirse en el endpoint destino del RCF. Por ejemplo, se pude autorizar a iniciar una comunicación sin la necesidad de enviar un ARQ al *gatekeeper*, autorizar a responder una llamada sin el ARQ al *gatekeeper*, autorización de ancho de banda total empleado por el *endpoint*. También se pude imponer que los mensajes de señalización de llamada pasen por el *gatekeeper*, si el *endpoint* involucrado no ha usado el mensaje ARQ previo al inicio de una comunicación.

2.3.6.2. Fast Connect

Con este procedimiento un *endpoint* originante (y sólamente el originante), pude llegar a proponer un conjunto de canales de medios para su inmediata apertura en el propio mensaje de establecimiento (*setup*) H.225.

Para esto se utiliza el elemento o campo *fastStart* incluido en el dicho mensaje encapsulando mensajes de apertura de canales de medios H.245 (*openLogicalChannel*). Así, el originante propondrá una serie de canales y codecs asociados, para la comunicación que pretende establecer.

Si el endpoint destino decide aceptar alguno de los canales de medios propuestos, debe responder también incluyendo un elemento *fastStart* en alguno de los mensajes de señalización de llamada H.225/Q.931; *Call Proceeding*, *Progress*, *Alerting* o *Connect*. En este caso, el elemento *fastStart* sólo contendrá información de los canales que el *endpoint* destino ha decidido aceptar, no pudiendo realizar contrapropuestas o alterando parámetros ya establecidos por el originante, ver figura 2.26.

Pero no es obligatorio aceptar el *"Fast Connect"* por parte del destinatario de la comunicación, porque no soporta este procedimiento o cualquier otra razón. Para ello simplemente no se incluye el elemento *fastStart* en ninguno de los mensajes antes mencionados o directamente se emplea el elemento *fastConnectRefused* en alguno de estos.

Una vez que los canales han sido aceptados, los *endpoint* que interviene en la comunicación están en condiciones para comenzar el intercambio de

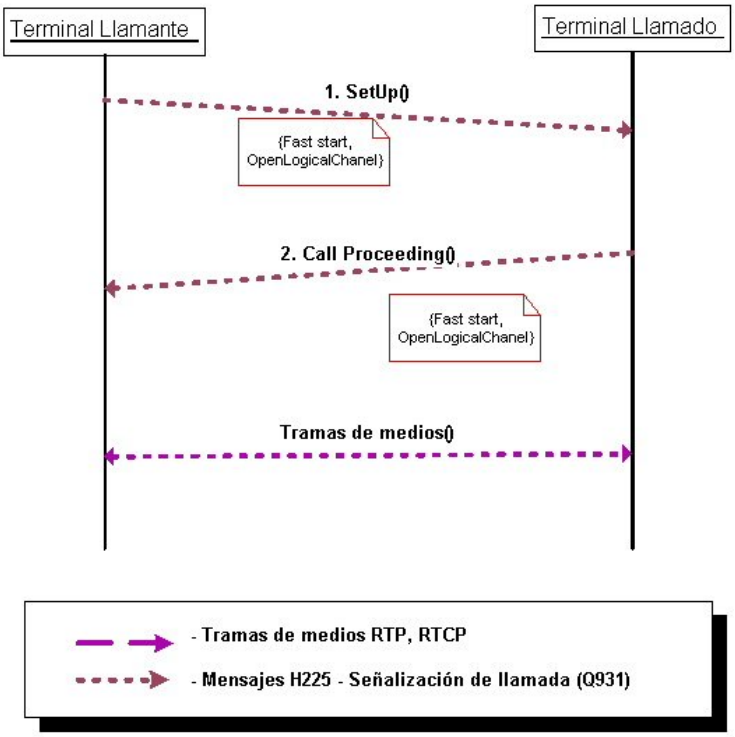

Figura 2.26: FastConnect

medios. Obsérvese que con ello se omite gran parte de los mensajes en la etapa de control H.245, ver figura 2.19 o 2.23.

Posterior a *"Fast Connect"* y si los *endpoint* lo requieren, también tiene la posibilidad de abrir el canal de control H.245, para el intercambio de otros mensajes de control. Pero la recomendación aconseja[5] la utilización del de H.245 vía túnel, que se explicará más adelante.

2.3.6.3. Encapsulado de H.245 en mensajes Q.931.

En este procedimiento se utiliza el canal de señalización de llamada, previamente abierto entre los *endpoints*, o entre *endpoint* y *gatekeeper*, para enviar mensajes H.245, en lugar de abrir un canal de control separado. Según la recomendación [H323], esto permite ahorro de recursos, sincronización de la señalización del control de la llamada, además de producir un menor tiempo de establecimiento.

Para esto se utiliza el elemento o campo *h245Control* de cualquiera de las unidades de datos *h323_uu_pdu*. En este campo se pueden copiar uno o más mensajes H.245 a ser transmitidos, por lo que el procedimiento es conocido como encapsulado o tunelización de mensajes H.245.

[5]Es obligatorio para *endpoints* que cumplen con H.323 versión 4 o posterior.

Además es empleado el campo *h245Tunneling* para anunciar que un *endpoint* posee la capacidad y quiere emplear este método de H.245 encapsulado. El terminal originante debe poner en TRUE este campo en el mensaje *setup*[6] y en todos los subsiguientes. De forma similar, el *endpoint* destino deberá también asignar TRUE a este campo en el mensaje de respuesta al *setup* y en los siguientes.

En caso de existir mensajes H.245 para enviar y no tener mensajes Q.931 para mandar, un terminal que haya sido habilitado por su par a emplear este procedimiento, podrá utilizar el mensaje *Facility* para encapsular. Además se advierte que el canal de señalización de llamada debe permanecer abierto durante toda la comunicación si se ha empelado tunelización de H.245 o, se deberá abrir un canal para este fin.

Para terminales que soporten versión 4 o posterior de la recomendación H.323 el tener habilitado el campo *h245Tunneling* es obligatorio.

2.3.6.4. Fast Connect y encapsulado de H.245

Los dos procedimientos anteriores pueden ser usados en combinación, pero para ello el *endpoint* originante debe iniciar el procedimiento *Fast Connect* enviando el campo *fastStart* como se dijo y solamente el campo *h245Tunneling* en TRUE. No debe usar el campo *h245Control*, ya que los mensajes que este incluya sobreescribirán el *fastStart*. Por otro lado el *endpoint* destino deberá también responder en el primer mensaje H.225/Q.931 con la bandera *h245Tunneling* y su respuesta al procedimiento *FastConnect*.

Así, todo otro mensaje H.245 que sea necesario intercambiar entre los participantes de la comunicación, podrá ser encapsulado en mensajes H.225/Q.931 posteriores. No olvidando que, al utilizar *Fast Connect*, el destino ya ha aceptado un grupo de canales y codecs asociado para el intercambio de medios.

2.3.6.5. Fast Connect en paralelo con tunelización de H.245

Existe también una posibilidad más, en la cual el *endpoint* originante podrá enviar mensajes H.245 en paralelo con el campo *fastStart* y este último no sea sobreescrito como se mencionó en el punto anterior. La utilidad de este procedimiento esta dada por la oportunidad que tiene este *endpoint* de incluir inmediatamente los primeros mensajes (*terminalCapabilitySet* y *masterSlaveDetermination*) del intercambio de control H.245. Esto aceleraría la conexión en caso de que estos mensajes sean requeridos por el *endpoint* destino, y solo tendría que responder a ellos encapsulando su respuesta H.245 en subsiguientes mensajes Q.931, o en caso de que el procedimiento *Fast Connect* falle.

Para este fin, se debe utilizar el campo *parallelH245Control*, que solo se encuentra presente en el unidad de H.225 *setup-UUIE*, que viaja dentro del mensaje *setup* Q.931 y, seteando la bandera *h245Tunneling* a TRUE. El campo *parallelH245Control* contendrá los mensajes H.245 que se desean enviar en forma de octetos crudos, un mensaje a continuación de otro.

[6]Esta claro que si el setup lleva mensajes H.245 encapsulados el campo **h245Tunneling** debe ser TRUE.

Si todo sale bien, el *endpoint* destino debe responder con un mensaje H.245 *terminalCapabilitySetAck* y *masterSlaveDeterminationAck* encapsulados en el canal de señalización del llamada H.225/Q.931, esta vez sí en el campo *h245Control*.

Debe aclararse una vez más que este mecanismo de señalización acelera el procedimiento visto en el punto 2.3.6.2, y que solo tiene sentido con la presencia del *fastStart*, pues si lo que pretende es solamente hacer tunelización de H.245 se debe emplear el procedimiento visto en 2.3.6.3

Capítulo 3

Bibliotecas de software para VoIP

3.1. Alternativas existentes en el comienzo del proyecto

Al momento de comenzar con el proyecto se investigaron las alternativas existentes para el desarrollo del mismo. Se buscaron distintas implementaciones de los stacks de protocolos que pudieran servir para el objetivo buscado. No sólo hubo que considerar la disponibilidad de las herramientas (código, bibliotecas de software), su funcionalidad y calidad, sino que también las posibilidades en cuanto a patentes para el uso de ellas y de la aplicación desarrollada.

Si bien el desarrollo de software y de comunidades de desarrolladores que intercambian código y conocimientos, se ve muy favorecida por la Internet, lo reciente de estos protocolos determina que los proyectos relacionados con ellos se encuentren en estados bastante primarios.

Las alternativas encontradas diferían bastante en varios aspectos. Las más importantes que merecen destaque en este documento son las siguientes: OpenH323, Vovida, J323, algunas JSR (*Java Specification Request*) de JCP (*Java Community Process*).

JCP constituye una comunidad donde los desarrolladores de Java participan en la especificación de JSRs (*Java Specification Requests*) para la posterior implementación de API o bibliotecas que hagan uso de Java. Existen varios grupos en los temas relacionados con VoIP. Existe una JSR de SIP, otra de Megaco y una de H.323, aunque el trabajo en esta última se encuentra en etapas iniciales, sin haberse desarrollado aún la especificación.

J.323 es un proyecto de IBM que brinda algún software para su utilización en desarrollo de aplicaciones de H.323. El código no es libre y las licencias restringen su uso a desarrollos personales y no comerciales. Si bien puede ser utilizado como elemento de prueba, al no tener el código, ni una biblioteca disponible sobre la que desarrollar una aplicación, no resultó de utilidad.

Los últimos dos proyectos considerados son de software libre, uno de

H.323 (OpenH323) y otro de SIP (Vovida). Las ventajas en cuanto a su utilización se basan en las mínimas restricciones que imponen sus licencias, y en el hecho de contar con las fuentes lo que permite, a través de su estudio, aprender mucho de normas y de programación.

El proyecto Vovida tiene licencia GPL (General Public Licence) mientras que OpenH323 usa la licencia Mozilla. La primera de ellas impone alguna restricción en el proyecto, en el sentido de que determina que el software debe seguir siendo libre. Mozilla no hace esta salvedad.

El proyecto Vovida está hecho en C++ y Java, mientras que OpenH323 está escrito en C++.

3.1.1. Desarrollos de software cliente para H.323

Si bien éstos no son una alternativa de partida para el desarrollo del presente proyecto se deben tener en cuenta dado que a lo largo de todo el desarrollo del mismo fue necesario ir realizando diferentes pruebas para lo que se necesitaron terminales H.323.

En Internet se tiene disponible diferentes terminales clientes H.323 de software, a continuación se listan algunos de ellos junto a un breve comentario y una referencia.

- **OpenH323 endpoint**s: OhPhone, OpenPhone y OpenAM, software de código libre basado en las bibliotecas OpenH323 y Pwlib. El segundo es idéntico al primero salvo que se presenta con una interfaz gráfica de usuario. *(http://www.openh323.org/code.html)*
- **GnomeMeeting**: clon de Netmeeting para Linux, basado en la OpenH323 *(http://www.gnomemeeting.org/)*
- **CPhone**: cliente GUI H.323 basado en QT *(http://cphone.sourceforge.net/)*
- **MyPhon**e: cliente H.323 para Windows, Audio, Video, Chat *(http://myphone.sourceforge.net/)*
- **XMeeting:** endpoint H.323 para Mac *(http://xmeeting.sourceforge.net/)*
- **J323 Engine**: Cliente H.323 Java para IBM, soporte JTAPI, ejecutable libre pero no el código fuente. J323 Engine es un software JavaTM que implementa las funciones para control de llamada y control de medios en un terminal H.323. J323 Engine incluye el estándar orientado a objetos, Java Telephony API (JTAPI), de forma que los desarrolladores pueden escribir sus propias interfaces de usuario o pueden integrar la funcionalidad de terminal H.323 a sus propias aplicaciones. *(http://www.alfaworks.ibm.com/tech/j323engine)*
- **Phone**: Cliente H.323 Java para IBM, ejecutable libre , pero código fuente no. *(http://www.alfaworks.ibm.com/tech/phone4java)*

Como se detalla en la sección 5.2, donde se detallan los terminales utilizados para las pruebas, básicamente, éstos fueron los del proyecto OpenH323 dado que también son de código libre y esto permitió realizar algunas pequeñas modificaciones de acuerdo a las necesidades que se presentaron.

3.1.2. Desarrollos de gatekeepers

Existen otros desarrollos de *gatekeepers* H.323 basados en la biblioteca OpenH323 como por ejemplo el proyecto Gnugk sobre el cual se puede encontrar más información el sitio web *http://gnugk.org*. Otro proyecto basado en la biblioteca es el OpenGatekeeper, sobre éste se pude obtener información en *http://openh323proxy.sourceforge.net/*.

3.1.3. Proyecto Vovida (SIP)

Vovida es un proyecto en el que se da una implementación del protocolo SIP para voz sobre IP.

Es un proyecto de código libre, según Vovida es una "plataforma de desarrollo y librerías de software para comunicaciones IP centralizado".

Puede ejecutarse sobre sistemas operativos Linux y Solaris, y hardware basado en Intel (I86). Implementa un softswitch SIP y provee facilidades básicas como forward de llamadas, bloqueo de llamadas, transferencia y llamada en espera. Así mismo provee librerías de software para implementar nuevas facilidades y aplicaciones en:

- C++
- Call processing language (CPL)
- Java telephony API (JTAPI)

Se puede obtener más información en la siguiente página:
http://www.vovida.org.

3.1.4. Ventajas y desventajas de su utilización

En este proyecto se optó por utilizar las bibliotecas del proyecto OpenH323. La utilización de estas bibliotecas presentó algunos aspectos favorables y otros no tanto, por ejemplo el que sea de código libre fue fundamental y determinante ya que da la posibilidad de utilización de sus funcionalidades de forma más clara y pudiendo modificarlas de acuerdo las necesidades que se tengan, lo cual contrarresta el hecho de que la biblioteca cuenta con escasa documentación y no demasiado clara. Otro factor importante a tener en cuenta es su característica multiplataforma, que da la posibilidad de generar un producto que pueda ser utilizado tanto en plataformas Linux como Windows, esto es una gran ventaja ya que amplía el campo de aplicación del programa, pudiendo generar el ejecutable de acuerdo a donde será utilizado.

Otro punto importante es el lenguaje de programación utilizado, los integrantes de este grupo de proyecto al momento de comenzar con él no teníamos conocimiento del lenguaje, si en cambio de JAVA, pero dado los puntos antes mencionados se decidió utilizar estas bibliotecas y por lo tanto se tuvo la necesidad de aprender el lenguaje de programación C++ y familiarizarse con él lo cual supuso un esfuerzo importante.

Además en la elección de OpenH323 fue determinante el que éste implementaba H.323 y, dado que el proyecto abarcaba algún servicio suplementario, se estimó que el uso de H.323 era más adecuado debido a que existía mayor trabajo normativo en esa área.

3.2. Proyecto OpenH323

3.2.1. Descripción

El proyecto OpenH.323 (http://www.openh323.org) apunta a crear un sistema completo interoperable que implemente el stack de protocolos ITU-T H.323 que pueda ser usado por desarrolladores privados y comerciales sin costo. Es código libre, y su utilización es impulsada a través de la licencia MPL (*Mozilla Public License*).

Implementa los protocolos H.225 (control de llamada), H.235 (seguridad), H.245 (control de medios), Q.931 (señalización de llamada), los codecs de audio : G.711, G.722, G.723, G.728, G.729, y los codecs de video H.261 y H.263.

Las bibliotecas fueron desarrolladas en el lenguaje de programación orientado a objetos C++, según sus desarrolladores la elección del lenguaje obedece a que éste da más posibilidades que el lenguaje C puro, en cuanto a la posibilidad de utilizar Java se argumenta que se eligió C++ por razones de performance y por ya estar familiarizado con él.

Según sus creadores el proyecto OpenH323 era necesario dado que implementaciones comerciales del stack de protocolos H.323 son caras y propietarias, lo cual limita las posibilidades de su utilización por parte de pequeñas compañías que deseen crear productos o servicios mediante dicho stack.

El proyecto está compuesto por las bibliotecas OpenH323 y PWLib, las cuales se detallan en las secciones siguientes.

3.2.2. Licencias

Como se mencionó anteriormente todo el software del proyecto OpenH323 esta registrado bajo licencia Mozilla.

La licencia MPL (http://www.mozilla.org/NPL/MPL-1.0.html) asegura tener acceso al código fuente. Mientras la biblioteca se utilice sin modificaciones, las aplicaciones que se desarrollen podrán tener cualquier tipo de licencia. En caso de modificaciones de la biblioteca, la licencia establece que dichos cambios deberán hacerse públicos bajo MPL.

La biblioteca ha sido compilada y utilizada ampliamente en plataformas Linux y Win32, pero también ha sido utilizada sobre plataformas Solaris, FreeBSD y BeOS.

No se necesita de hardware especial para su utilización.

3.3. Biblioteca OpenH323

La OpenH323 es una biblioteca de clases, de tipo Open Source, que permite el desarrollo de aplicaciones que sirven de plataforma para comunicaciones multimedia sobre redes de paquetes utilizando el stack de protocolos indicados por la recomendación H.323 [H323].

El lenguaje de programación usado por la biblioteca es C++ y, cabe destacar el uso extensivo de diferentes técnicas de programación orientada a objetos que sus desarrolladores han empleado, como ser herencia y polimorfismo.

Esto último la destaca como una herramienta muy cómoda para el programador cliente, una vez que se ha comprendido la aplicación de estas técnicas en la OpenH323 y, la finalidad de cada clase y su métodos, en la misma. Por desgracia este último punto, se ve dificultado por la escasa documentación y material de ejemplo que esta posee, ver sección 3.3.2.

A su vez la OpenH323 utiliza la biblioteca PWlib como base, utilizando herencia y/o composición emplea las clases de la PWlib como elementos constructivos. Este último hecho es el que le otorga la naturaleza multiplataforma a la OpenH323. Ya que empleando versiones para GNU/ Linux, Unix[1] o Microsoft Windows de la PWlib es posible desarrollar, compilar y ejecutar aplicaciones para cualquiera de estas familias de sistemas operativos.

En las siguientes secciones se buscará dar una explicación a las diferentes posibilidades que ofrece la biblioteca y explicar alguna de sus clases más utilizadas por el proyecto. Claro que no se pretende hacer una documentación exhaustiva de la OpenH323, pues esto supera los objetivos del presente documento.

3.3.1. Clases componentes

La versión de la biblioteca que se pasa a detallar es la 1.9.5[2]. Aunque este proyecto ha utilizado también la versión 1.11.7[3] sin necesidad de cambios en el código.

Existen alrededor de 1315 clases en la OpenH323, las cuales pueden ser agrupadas de acuerdo con las funciones que pueden cumplir, para realizar su estudio de forma más ordenada. Estos grupos de clases puede ser vistos en el cuadro 3.1.

[1] Recientemente se ha anunciado la portabilidad de la OpenH323 a Sun Solaris [VoxG].
[2] Debe ser usada con la versión 1.3.5 de la PWLib.
[3] Para la versión 1.4.11 de la PWLib.

Grupo.	Función.	
1.	Representación de Servicios y entidades de H.323.	
2.	Representación de un canal de protocolo H.323.	
3.	Representación de conexión.	
4.	Representación de tipos de datos.	
5.	Representación de mensajes de protocolos en H.323.	
	A)	Clases para H.225 RAS y H.225 señalización de llamada.
	B)	Clases para manejar solicitudes RAS.
	C)	Clases para el protocolo H.245.
	D)	Clases para codificar Q.931.
	E)	Clases para H.450.
	F)	Clases para H.235.
	G)	Clases para X.880.
	H)	Clases para T.38 y T120.
	I)	Clases para RTP.
6.	Representación de Codecs.	
7.	Representación de un canal para protocolo de red.	
8.	Hilos de proceso especializados.	

Cuadro 3.1: Grupos de Clases de la OpenH323.

Es posible también hacer un esquema jerárquico de las clases en la OpenH323, en cuya cúspide se encuentran las clases más complejas y con mayor número de funciones, son las que representan entidades de una red H.323 (3.3.1.1). Por el otro lado en la base de esta división jerárquica se pueden ubicar aquellas clases que cumplen un pequeño número de funciones y que generalmente son componentes de las primeras.

De acuerdo con la complejidad que desea alcanzar el programador cliente con su desarrollo deberá tomar clases de la cúspide de la pirámide o ir más hacia los niveles inferiores.

A continuación una explicación de los diferentes grupos de clases.

3.3.1.1. Representación de servicios y entidades de H.323

En este grupo podemos encontrar aquellas clases que directamente representan una entidad de la red H.323, como ser un endpoint y un gatekeeper, ver cuadro 3.2.

Clase	Función
H323EndPoint	Implementa un *endpoint* en la red H.323.
H323GatekeeperServer	Implementa un *gatekeeper* en la red H.323.

Cuadro 3.2: Entidades de red H.323.

La clase H323EndPoint contiene las funcionalidades básicas con las cuales es posible desarrollar un terminal de red H.323. Podrá recibir y enviar llamadas en forma muy rudimentaria. Para esto el programador cliente deberá primero heredar de esta clase y sobreescribir sus métodos. Luego se

deberá usar composición para unir las funcionalidades que se han obtenido al resto del proyecto; por ejemplo creando un proceso (clase que contiene el main) que la contenga e, iniciando los variables de H323EndPoint y los *listeners* (3.3.1.2) en forma apropiada. Es necesario subrayar que aunque parezca sencillo hacer esto, no debe olvidase que la mayoría de los métodos del H323Endpoint deben ser sobreescritos para que puedan cumplir con un cometido. Principalmente aquellos denominados *callback functions* por los desarrolladores de la OpenH323.

Un *callback function* es un método que siempre es disparado por un evento generado en un nivel inferior en la abstracción que la clase representa. O sea, un mensaje de inicio de llamada provocará el llamado a un método OnIncomingCall, esta debe ser sobreescrita para cumplir con todas las tareas necesarias para recibir esta nueva comunicación.

La clase H323GatekeeperServer contiene las funciones básicas que debe cumplir una entidad *gatekeeper* en una red H.323. Es el esqueleto necesario para que un programador cliente implemente rápidamente las funciones obligatorias de la entidad *gatekeeper* H.323 (2.3.4.1).

Nuevamente será necesario heredar de esta clase y sobreescribir sus métodos. Al igual que sucedía con la clase anterior habrá un grupo de métodos llamados *callback functions*, los cuales serán llamados en tiempo de ejecución cada vez que se active un evento dado, la llegada de un mensaje RAS. Por ejemplo el método OnRegistration cada vez que un terminal envíe un mensaje RegistrationRequest.

Luego es necesario crear un proceso (clase que contiene el main) que inicialice correctamente esta clase heredada de H323GatekeeperServer. Esto último implica setear aquellas variables que se considere necesario en el constructor y agregar (como mínimo) un *listener* (3.3.1.2) para el protocolo RAS.

Las funciones opcionales pueden ser implementadas con un poco más de trabajo e ingenio. Lo cual será detallado en el resto de este documento.

3.3.1.2. Representación de canales de protocolos de la H.323

Este grupo de clases tiene en común la función de escuchar mensajes del protocolo RAS, o del protocolo H.225/Q.931[4], en un canal de transporte dado (UDP y TCP respectivamente) y, proceder de alguna forma cuando se haya recibido y decodificado en forma satisfactoria uno de estos mensajes.

Las clases incluidas en este grupo son las mostradas en el cuadro 3.3.

[4]Por simplicidad notaremos de esta forma al protocolo de señalización de llamada H.225, encapsulado en paquetes Q.931, en el campo User-User IE .

Clase	Función
H225_RAS	Clase abstracta para protocolo RAS.
H323GatekeeperListener	Escucha mensajes RAS para el H323GatekeeperServer.
H323Gatekeeper	Escucha mensajes RAS para el H323EndPoint.
H323Listener	Clase abstracta para escuchar mensajes H.225/Q.931.
H323ListenerTCP	Escuchar mensajes H.225/Q.931.

Cuadro 3.3: Canales de protocolos de la H.323.

Como lo muestra el cuadro, H225_RAS es la clase abstracta creada para representar un canal de tipo RAS. De esta heredan H323GatekeeperListener y H323Gatekeeper. La primera de ellas especializada para implementar un canal RAS del *gatekeeper* y la segunda para el canal RAS de un *endpoint*.

Como ya se mencionó durante la inicialización del *gatekeeper* se debe crear un H323GatekeeperListener usando el método AddListener(). Para la inicialización del canal RAS del *endpoint* se precede en forma distinta, se usa el método FindGatekeeper() del H323EndPoint el cual internamente crea una instancia de la clase H323Gatekeeper.

En el caso de H323GatekeeperListener el AddListener() llama al método StartRasChannel() del primero, el cual crea e inicia un *thread* y lo asocia con la clase de tipo transporte (ver 3.3.1.7) que posee el *listener*. Este *thread* contiene un loop infinito que realiza la tarea de escuchar en este transporte asociado con el *listener*.

Cuando el thread consigue leer con éxito un mensaje lo eleva al H323GatekeeperListener llamando al método HandleRasPDU() el cual decodifica el mensaje y lo eleva un nivel más, hasta el H323GatekeeperServer.

El H323Gatekeeper tiene un funcionamiento similar. Aunque el *thread* es iniciado al crear la clase y este no se dedica a decodificar continuamente mensajes RAS si no que a controlar dos timers, uno para el envío de respuestas IRR, y el otro para el envío de RRQ ligeros.

Por último tenemos los *listeners* H.225/Q.931 del H323EndPoint. El H323Listener en realidad es una clase abstracta y la clase de interés realmente es la H323ListenerTCP. Esta última al ser creada, inicia un puerto TCP en la dirección de transporte que sea pasada como argumento en su constructor. Luego para abrir este puerto y comenzar a recibir mensajes se debe utilizar el método StartListener() del H323EndPoint. Este último es el que inicia el thread, ya que a su vez H323Listener hereda de PThread.

Una vez recibido un mensaje en el puerto iniciado, el thread del H323ListenerTCP se encarga de crear un nuevo canal de transporte TCP (H323TransportTCP), el cual está abierto durante toda la conexión y llevará mensajes desde y hacia el *endpoint*. Además de este, se crea otro nuevo thread (H225TransportThread) el cual será encargada de procesar el primer mensaje de inicio de una nueva conexión.

3.3.1.3. Representación de conexión

El siguiente par de clases (ver cuadro 3.4) tiene en común el hecho de representar una llamada y presentar una primera etapa de procesamiento de los mensajes RAS o H.225/Q.931 recibidos durante la misma.

Representan una llamada pues poseen un extenso grupo de variables que pueden albergar información de la comunicación (IP origen, IP destino, ancho de banda usado, etc) así como del estado de la misma.

Además, poseen métodos para procesar la llegada de cada mensaje durante establecimiento, transcurso y finalización de una llamada. En general estos métodos realizan variadas tareas; como ser el procesamiento de los campos en el mensaje dado y su posterior carga en las variables de la clase, actualización del estado de la llamada, aviso a la entidad superior en la jerarquía de la llegada del nuevo mensaje, e incluso el envío de una respuesta al otro extremo de la comunicación.

Clase	Función
H323GatekeeperCall	Representa una llamada a nivel del protocolo RAS.
H323Connection	Representa una llamada a nivel de H.225/Q.931.

Cuadro 3.4: Clases de tipo conexión.

Una nueva instancia de la clase H323GatekeeperCall, es creada cuando se recibe un nuevo mensaje ARQ. Esto es realizado por el el método CreateCall() del H323GatekeeperServer. Implica que si se desea modificar el comportamiento de la H323GatekeeperCall además de heredar y sobrescribir sus métodos, es necesario modificar CreateCall() para que instancie objetos de la nueva clase.

Después de la creación de la H323GatekeeperCall, la información de la llamada es cargada en esta. Para ello es llamado el método OnAdmission() desde H323GatekeeperServer::OnAdmission().

Por último conviene subrayar que por comunicación establecida el *gatekeeper* tendrá dos H323GatekeeperCall, una generada por el ARQ del terminal destino y otra por el ARQ del origen. Estas son guardadas en una lista que posee el H323GatekeeperServer y para obtener una referencia de esta se debe utilizar el método FindCall().

Por otro lado la H323Connection es creada por el H323EndPoint cada vez que recibe un mensaje H.225/Q.931 setup mediante el método CreateConnection() y, cada vez que el endpoint origina una nueva comunicación, desde el método InternalMakeCall().

Esta clase para el *endpoint*, es única por comunicación establecida y posee un poco más de inteligencia que la H323GatekeeperCall, ya que posee un grupo de métodos de tipo *callback* que son llamados con la recepción de cada nuevo mensaje H.225/Q.931 durante todo el transcurso de la llamada. Estos métodos cambian el estado de la conexión, crean nuevos *threads* para contol de protocolos como H.245 y H.450. O sea, si el objetivo es construir

un terminal es casi inevitable tener que utilizar esta clase como elemento constructivo.

Al igual que ocurría con el *gatekeeper* el H323EndPoint posee una lista (en realidad es un contenedor de tipo diccionario) de conexiones, la cual pude ser consultada mediante el método FindConnectionWithLock()

3.3.1.4. Representación de tipos de datos

Este grupo de clases se utiliza para representar información de algún tipo, ver cuadro 3.5. En el primer caso , se tiene una clase usada para manejar alias de usuarios. La H323TransportAddress, para manejar mediante un objeto tipo string (hereda de la clase PString de la PWLib), una dirección IP, puerto, y nombre de Host en caso de que esté disponible. Además presenta una interfaz que permite obtener rápidamente alguno de estos valores.

Clase	Función
H225_AliasAddress	Representar un alias de usuario.
H323TransportAddress	Representar dirección de transporte.
H323RegisteredEndPoint	Guardar información de terminales registrados.
OpalGloballyUniqueID, H225_GloballyUniqueID	Representar un identificador único de llamada.

Cuadro 3.5: Representación de datos.

El primer par de clases es muy usado por los objetos de la biblioteca para albergar justamente el alias de un usuario y el conjunto dirección IP y puerto, para identificar en forma completa un punto de acceso a un servicio. El formato usado para el string en H323TransportAddress es: ip$A.B.C.D:P, donde A, B, C y D representan el primer, segundo, tercer y cuarto octeto de la dirección IP en decimal, mientras que P es el puerto. También posee algunos métodos que permiten por ejemplo crear un *listener* TCP mediante una dirección de transporte, resolver el nombre de host a partir de una IP, entre otros.

La clase H323RegisteredEndPoint es utilizada en el H323GatekeeperServer para implementar una base de datos (diccionario en realidad) de los terminales registrados con este. Obviamente aquellos terminales que hayan solicitado ser registrados con el *gatekeeper* y este haya aceptado dicha solicitud.

El último par de clases son usadas en toda la biblioteca como una representación del identificador global único de llamada (guid) mencionado en la recomendación H.225[H225] . La H225_GloballyUniqueID es generada a partir de ASN.1 (ver 3.3.1.5) y se puede encontrar en cualquiera de los mensajes RAS como ARQ, DRQ, BRJ, etc., o en los mensajes H.225/Q.931 setup , connect, etc. Mientras que OpalGloballyUniqueID, es el objeto que generalmente usan las clases H323GatekeeperServer, H323Connection y H323EndPoint como identificador de llamada, ya que ofrece una interfaz

más amigable con el programador cliente, entre otros posee un método constructor que puede recibir la H225_GloballyUniqueId como argumento (a veces se conoce como un *wrapper*).

3.3.1.5. Representación de mensajes de protocolos en H.323

Los siguientes grupos de clases son los más numerosos dentro de la biblioteca openH323 y son los que se encontrarían dentro de la pirámide jerárquica en los niveles más bajos. Las clases en estos grupos tienen en común dos características.

La primera es el hecho de que las mismas son utilizadas para manejar todos los tipos de mensajes especificados por la recomendación H.323 [H323].

La segunda característica en común es que, en general, son generadas por un programa parser[5] a partir de archivos ASN.1. Cada definición realizada en ASN.1 del conjunto de protocolos H.323 tiene una representación en alguna de estás clases. Debido a esto, solo están constituidas por un constructor (destructor) y un grupo de campos o miembros de datos, los cuales son públicos y pueden ser accedidos directamente.

Este último hecho tiene sus excepciones en algunas clases que en realidad ya vienen definidas por la biblioteca y se utilizan para encapsular a las anteriores y permitiendo su mejor manejo. Se detallan más adelante.

A) Clases para el protocolo RAS y H.225/Q.931 señalización de llamada

En este grupo se encuentran todas las definiciones realizadas para los mensajes RAS y mensajes de señalización de llamada H.225/Q.931 en la recomendación H.225 [H225] anexo H. El grupo completo de clases puede ser visto en el cuadro 3.1.

[5]El asnparser.exe generado al copilar la PWLib.

H323RasPDU	H225_Content	H225_PresentationIndicator
H225_ANSI_41_UIM	H225_CryptoH323Token	H225_PrivatePartyNumber
H225_ANSI_41_UIM_system_id	H225_CryptoH323Token_cryptoEPPwdHash	H225_PrivateTypeOfNumber
H225_AddressPattern	H225_CryptoH323Token_cryptoGKPwdHash	H225_Progress_UUIE
H225_AddressPattern_range	H225_DataRate	H225_ProtocolIdentifier
H225_AdmissionConfirm	H225_DisengageConfirm	H225_PublicPartyNumber
H225_AdmissionConfirm_language	H225_DisengageReason	H225_PublicTypeOfNumber
H225_AdmissionReject	H225_DisengageReject	H225_Q954Details
H225_AdmissionRejectReason	H225_DisengageRejectReason	H225_QseriesOptions
H225_AdmissionRequest	H225_DisengageRequest	H225_RTPSession
H225_Alerting_UUIE	H225_EncodedFastStartToken	H225_RTPSession_associatedSessionIds
H225_AliasAddress	H225_EncryptIntAlg	H225_RasMessage
H225_AltGKInfo	H225_Endpoint	H225_RasUsageInfoTypes
H225_AlternateGK	H225_EndpointIdentifier	H225_RasUsageInformation
H225_AlternateTransportAddresses	H225_EndpointType	H225_RasUsageSpecification
H225_ArrayOf_AddressPattern	H225_EnumeratedParameter	H225_RasUsageSpecification_callStartingPoint
H225_ArrayOf_AliasAddress	H225_ExtendedAliasAddress	H225_RasUsageSpecification_when
H225_ArrayOf_AlternateGK	H225_FacilityReason	H225_RegistrationConfirm
H225_ArrayOf_AuthenticationMechanism	H225_Facility_UUIE	H225_RegistrationConfirm_preGrantedARQ
H225_ArrayOf_BandwidthDetails	H225_FastStartToken	H225_RegistrationReject
H225_ArrayOf_CallReferenceValue	H225_FeatureDescriptor	H225_RegistrationRejectReason
H225_ArrayOf_CallsAvailable	H225_FeatureSet	H225_RegistrationRejectReason_invalidTerminalAliases
H225_ArrayOf_ClearToken	H225_GSM_UIM	H225_RegistrationRequest
H225_ArrayOf_ConferenceIdentifier	H225_GatekeeperConfirm	H225_ReleaseCompleteReason
H225_ArrayOf_ConferenceList	H225_GatekeeperIdentifier	H225_ReleaseComplete_UUIE
H225_ArrayOf_CryptoH323Token	H225_GatekeeperInfo	H225_RequestInProgress
H225_ArrayOf_DataRate	H225_GatekeeperReject	H225_RequestSeqNum
H225_ArrayOf_Endpoint	H225_GatekeeperRejectReason	H225_ResourcesAvailableConfirm
H225_ArrayOf_EnumeratedParameter	H225_GatekeeperRequest	H225_ResourcesAvailableIndicate
H225_ArrayOf_ExtendedAliasAddress	H225_GatewayInfo	H225_ScnConnectionAggregation
H225_ArrayOf_FeatureDescriptor	H225_GenericData	H225_ScnConnectionType
H225_ArrayOf_GenericData	H225_GenericIdentifier	H225_ScreeningIndicator
H225_ArrayOf_H245Security	H225_GloballyUniqueID	H225_SecurityCapabilities
H225_ArrayOf_H248PackagesDescriptor	H225_GroupID	H225_SecurityServiceMode
H225_ArrayOf_IntegrityMechanism	H225_GroupID_member	H225_ServiceControlDescriptor
H225_ArrayOf_NonStandardParameter	H225_H221NonStandard	H225_ServiceControlIndication
H225_ArrayOf_PASN_ObjectId	H225_H245Security	H225_ServiceControlIndication_callSpecific
H225_ArrayOf_PASN_OctetString	H225_H248PackagesDescriptor	H225_ServiceControlResponse
H225_ArrayOf_PartyNumber	H225_H248SignalsDescriptor	H225_ServiceControlResponse_result
H225_ArrayOf_RTPSession	H225_H310Caps	H225_ServiceControlSession
H225_ArrayOf_RasUsageSpecification	H225_H320Caps	H225_ServiceControlSession_reason
H225_ArrayOf_ServiceControlSession	H225_H321Caps	H225_SetupAcknowledge_UUIE
H225_ArrayOf_SupportedPrefix	H225_H322Caps	H225_Setup_UUIE
H225_ArrayOf_SupportedProtocols	H225_H323Caps	H225_Setup_UUIE_conferenceGoal
H225_ArrayOf_TransportAddress	H225_H323_UU_PDU	H225_Setup_UUIE_connectionParameters
H225_ArrayOf_TransportChannelInfo	H225_H323_UU_PDU_h323_message_body	H225_Setup_UUIE_language
H225_ArrayOf_TunnelledProtocol	H225_H323_UU_PDU_tunnelledSignallingMessage	H225_StatusInquiry_UUIE
H225_BandRejectReason	H225_H323_UserInformation	H225_Status_UUIE
H225_BandWidth	H225_H323_UserInformation_user_data	H225_StimulusControl
H225_BandwidthConfirm	H225_H324Caps	H225_SupportedPrefix
H225_BandwidthDetails	H225_ICV	H225_SupportedProtocols
H225_BandwidthReject	H225_InfoRequest	H225_T120OnlyCaps
H225_BandwidthRequest	H225_InfoRequestAck	H225_T38FaxAnnexbOnlyCaps
H225_CallCapacity	H225_InfoRequestNak	H225_TBCD_STRING
H225_CallCapacityInfo	H225_InfoRequestNakReason	H225_TerminalInfo
H225_CallCreditCapability	H225_InfoRequestResponse	H225_TimeToLive
H225_CallCreditServiceControl	H225_InfoRequestResponseStatus	H225_TransportAddress
H225_CallCreditServiceControl_billingMode	H225_InfoRequestResponse_perCallInfo	H225_TransportAddress_ip6Address
H225_CallCreditServiceControl_callStartingPoint	H225_InfoRequestResponse_perCallInfo_subtype	H225_TransportAddress_ipAddress
H225_CallIdentifier	H225_InfoRequestResponse_perCallInfo_subtype_pdu	H225_TransportAddress_ipSourceRoute
H225_CallLinkage	H225_InfoRequestResponse_perCallInfo_subtype_pdu_subtype	H225_TransportAddress_ipSourceRoute_route
H225_CallModel	H225_Information_UUIE	H225_TransportAddress_ipSourceRoute_routing
H225_CallProceeding_UUIE	H225_IntegrityMechanism	H225_TransportAddress_ipxAddress
H225_CallReferenceValue	H225_LocationConfirm	H225_TransportChannelInfo
H225_CallTerminationCause	H225_LocationReject	H225_TransportQOS
H225_CallType	H225_LocationRejectReason	H225_TunnelledProtocol
H225_CallsAvailable	H225_LocationRequest	H225_TunnelledProtocolAlternateIdentifier
H225_CapacityReportingCapability	H225_McuInfo	H225_TunnelledProtocol_id
H225_CapacityReportingSpecification	H225_MobileUIM	H225_UUIEsRequested
H225_CapacityReportingSpecification_when	H225_NonIsoIntegrityMechanism	H225_UnknownMessageResponse
H225_CicInfo	H225_NonStandardIdentifier	H225_UnregRejectReason
H225_CicInfo_cic	H225_NonStandardMessage	H225_UnregRequestReason
H225_CircuitIdentifier	H225_NonStandardParameter	H225_UnregistrationConfirm
H225_CircuitInfo	H225_NonStandardProtocol	H225_UnregistrationReject
H225_ConferenceIdentifier	H225_Notify_UUIE	H225_UnregistrationRequest
H225_ConferenceList	H225_NumberDigits	H225_UseSpecifiedTransport
H225_Connect_UUIE	H225_PartyNumber	H225_VendorIdentifier
H225_Connect_UUIE_language		H225_VoiceCaps

Figura 3.1: Clases para H.225

En este grupo podemos por ejemplo encontrar una clase H225_AdmissionConfirm, la cual justamente representa un mensaje de confirmación de solicitud de admisión. Esta, está compuesta por un conjunto de miembros, por ejemplo m_destinationInfo que es de tipo H225_ArrayOf_AliasAddress, clase que también podemos encontrar en este grupo. Como ya se mencionó este miembro puede ser accedido directamente ya que es público.

También es posible encontrar los campos opcionales de este mensaje, como ejemplo podemos mencionar el e_destinationInfo.

En este grupo conviene destacar la clase H323RasPDU, que es utilizada para encapsular todos los posibles mensajes RAS. Además posee un grupo de métodos que permiten generar estos tipos de mensajes, por ejemplo el BuildAdmissionRequest() crea una clase H225_AdmissionRequest.

B) Clases para manejar solicitudes RAS

Este grupo de clases es utilizado por el H323GatekeeperServer para manejar solicitudes realizadas mediante mensajes RAS. Tienen en común el hecho de venir definidas en la biblioteca, o sea, no son generadas por el parser mencionado anteriormente.

El cuadro 3.6, muestra este grupo de clases.

Clases	Función
H323GatekeeperRequest	Clase abstracta para el manejo de solicitudes RAS.
H323GatekeeperARQ	Manejo de solicitudes de admisión.
H323GatekeeperBRQ	Manejo de solicitudes de ancho de banda.
H323GatekeeperDRQ	Manejo de solicitudes de desligamiento.
H323GatekeeperGRQ	Manejo mensajes de descubrimiento de *gatekeeper*.
H323GatekeeperIRR	Manejo de solicitudes de estado.
H323GatekeeperLRQ	Manejo de solicitudes de localización.
H323GatekeeperRRQ	Manejo de solicitudes de registro.
H323GatekeeperURQ	Manejo de solicitudes de borrado de registro.

Cuadro 3.6: Clases para manejo de solicitudes RAS.

La primer clase del cuadro es simplemente la clase abstracta de la cual heredará el resto. Cada una de las siguientes se utilizan para manejar los mensajes involucrados en cada una de las tareas que implican al *gatekeeper* soportar un canal RAS; solicitudes de admisión, solicitudes de registro, etc.

Por ejemplo, para cada solicitud de admisión que recibe el H323GatekeeperServer, será llamado el método de *callback* OnAdmission(). El cual recibe como parámetro un H323GatekeeperARQ. El programador cliente deberá buscar en este objeto la información recibida en el mensaje original. Para esto último dispone de la interfaz de la propia H323GatekeeperARQ, pero también deberá recurrir a uno de sus miembros públicos - arq - de tipo H225_AdmissionRequest. Para armar el mensaje de respuesta también se deberá manipular esta clase H323GatekeeperARQ, pues tiene un par de

miembros para las dos posibles respuestas; acf de tipo H225_AdmissionConfirm y arj de tipo H225_AdmissionReject.

C) Clases para el protocolo H.245

De forma similar al grupo A) se encuentran los mensajes y entidades generadas a partir de ASN.1 para el protocolo H.245 [H245].

En este grupo se puede encontrar, por ejemplo, la clase H245_TerminalCapabilitySet, la cual corresponde a la tabla de capacidades que envía un terminal durante el intercambio H.245 o durante el *Fast Connect*. En el intercambio H.245 se utiliza para esto el mensaje MultimediaSystemControlMessage, que como no podría ser de otra forma está representado en este conjunto por la clase H245_MultimediaSystemControlMessage.

La H323ControlPDU es la clase usada para encapsular [6] y generar este tipo de mensajes. Encapsular pues internamente posee un miembro de tipo H245_MultimediaSystemControlMessage, y generar pues tiene métodos pa ra generar cada tipo de mensaje de este protocolo, por ejemplo el BuildTerminalCapabilitySet() genera justamente un H245_TerminalCapabilitySet.

D) Clases para codificar Q.931

Este grupo está formado por el par de clases en el cuadro 3.7. La función del grupo es encapsular los mensajes de señalización de llamada adaptados de Q.931[Q931] para la recomendación H.323.

Clases	Función.
Q931	Representar mensajes Q.931.
H323SignalPDU	Encapsular y generar los mensajes Q.931.

Cuadro 3.7: Clases para Q.931.

La primera clase en el cuadro encapsula y genera todos los tipos de mensaje Q.931 adaptados a la recomendación H.323. Mientras que la segunda posee un miembro - justamente - de tipo Q.931 y es usada también como manejador de la primera.

Cada función *callback* de H323Connection y H323EndPoint recibe como argumento una clase del tipo H323SignalPDU. Por lo tanto la forma de analizar un mensaje de señalización de llamada, tiene como punto de partida esta última. A su vez, si el programador cliente se encuentra codificando a este nivel, la forma de generar un mensaje de respuesta es utilizar también la H323SignalPDU.

El procedimiento habitual para esto último es crear una instancia de H323SignalPDU y luego, por ejemplo, llamar al método BuildSetup(), por lo que se obtiene un mensaje setup, con los mínimos campos inicializados.

[6] Nuevamente estamos hablando de un tipo *wrapper*.

E) Clases para H.450

En este grupo se encuentran representados todos los mensajes, operaciones y procedimientos necesarios para implementar los servicios suplementarios definidos en H.450.1 a H.450.9. En esté último aspecto la biblioteca OpenH323 aún se encuentra en desarrollo, ya que no están presentes todos los servicios suplementarios especificados hasta hoy día por la recomendación H.323.

F) Clases para H.235

Este grupo es utilizado para implementar los procedimientos de seguridad y encriptado definidos en la recomendación H.235 [H235].

Del cuadro 3.2 las clases no generadas por el programa parser son; H235Authenticator, H235AuthProcedure1, H235AuthSimpleMD5.
La primera de ellas es abstracta y define una interfaz común para mecanismos de autenticación definidos en H.235.

H235AuthProcedure1	H235_EncodedPwdCertToken
H235AuthSimpleMD5	H235_EncodedReturnSig
H235Authenticator	H235_H235CertificateSignature
H235_AuthenticationBES	H235_H235Key
H235_AuthenticationMechanism	H235_HASHED
H235_ChallengeString	H235_IV16
H235_ClearToken	H235_IV8
H235_CryptoToken	H235_Identifier
H235_CryptoToken_cryptoEncryptedToken	H235_KeyMaterial
H235_CryptoToken_cryptoHashedToken	H235_KeySignedMaterial
H235_CryptoToken_cryptoSignedToken	H235_KeySyncMaterial
H235_DHset	H235_NonStandardParameter
H235_ECDSAlikeSignature	H235_Params
H235_ECKASDH	H235_Password
H235_ECKASDH_eckasdh2	H235_PwdCertToken
H235_ECKASDH_eckasdhp	H235_RandomVal
H235_ENCRYPTED	H235_ReturnSig
H235_EncodedGeneralToken	H235_SIGNED
H235_EncodedKeySignedMaterial	H235_TimeStamp
H235_EncodedKeySyncMaterial	H235_TypedCertificate

Figura 3.2: Clases para H.235

H235AuthSimpleMD5 es la clase creada por los programadores de la OpenH323 para implementar el mecanismo de autenticación basada en MD5. Mientras que, H235AuthProcedure1 implementa el mecanismo de autenticación especificado en el anexo D [7] de la H.235.

G) Clases para X.880

La entidad de control de los servicios suplementarios basados en H.450 que se debe encontrar en cada *endpoint* , debe utilizar un servicio denominado ROS (*Remote Operations Service*), basado en la Recomendación X.880

[7] Base line Procedure 1.

de la ITU. Este grupo de clases es generado automáticamente por el parser a partir de un archivo ASN.1, debido a esta dependencia.

H) Clases para T.38 y T.120

En este grupo se tiene las clases generadas automáticamente por el parser a partir de la especificación ASN.1 del protocolo T.38 y el protocolo T.120. El primero es utilizado por la recomendación H.323 para implementar el servicio de Fax. Mientras que el segundo implementa la transmisión de datos entre terminales H.323.

I) Clases para RTP

En este conjunto se encuentran las clases utilizadas por la OpenH323 para establecer sesiones RTP. Conviene destacar por ejemplo la clase RTP_ Session, cuyo cometido es, justamente, ofrecer a una interfaz al programador cliente que desee llegar a manipular directamente RTP. Mientras que la clase RTP_DataFrame, ofrece un objeto que encapsula tramas de datos correspondientes a este protocolo.

RTP_ControlFrame	RTP_UDP
RTP_DataFrame	RTP_UserData
RTP_JitterBuffer	H323_RTP_Session
RTP_Session	H323_RTP_UDP
RTP_SessionManager	H323RTPChannel

Cuadro 3.8: Clases para RTP

3.3.1.6. Representación de codecs

Tenemos en este conjunto (ver cuadro 3.3) todos los codecs y capacidades implementadas por la OpenH323 hasta la versión 1.9.5. Es importante destacar esto último, ya que la comunidad de programadores de la biblioteca también ha trabajado en los últimos tiempos en el área de implementar nuevos codecs y mejorar las posibilidades de los ya existentes.

Obviamente en este conjunto se encuentran los codecs obligatorios especificados por la H.323; como ser la clase H323_muLawCodec y la H323_A-LawCodec. Las cuales implementan los codecs de la G.711 Ley μ y Ley A respectivamente.

Hay que diferenciar en este conjunto las que son una representación de la capacidad (para los mensajes H.245), de aquellas clases que realmente implementan el codec. Las primeras siempre contiene en su nombre la palabra *capability*.

Las primeras heredan de H323Capability, clase abstracta que justamente ofrece una interfaz común para representar el codec en el *endpoint*.

G7231_File_Capability	H323_H263Codec
G7231_File_Codec	H323_LIDCapability
H323AudioCodec	H323_LIDCodec
H323Codec	H323_LPC10Capability
H323FramedAudioCodec	H323_LPC10Codec
H323StreamedAudioCodec	H323_T120Capability
H323VideoCodec	H323_T120Channel
H323VideoDevice	H323_T38Capability
H323_ALawCodec	H323_T38Channel
H323_CiscoG7231aLIDCapability	H323_T38NonStandardCapability
H323_Cu30Capability	**H323_muLawCodec**
H323_Cu30Codec	MicrosoftGSMAudioCapability
H323_G7231Capability	MicrosoftGSMCodec
H323_G7231Codec	MicrosoftIMAAudioCapability
H323_G726_Capability	MicrosoftIMACodec
H323_G726_Codec	MicrosoftNonStandardAudioCapability
H323_G729ACapability	NullVideoOutputDevice
H323_G729ACodec	SpeexCodec
H323_G729Capability	SpeexNarrow2AudioCapability
H323_GSM0610Capability	SpeexNarrow3AudioCapability
H323_GSM0610Codec	SpeexNarrow4AudioCapability
H323_H261Capability	SpeexNarrow5AudioCapability
H323_H261Codec	SpeexNarrow6AudioCapability
H323_H263Capability	SpeexNonStandardAudioCapability
H323Capability	PPMVideoOutputDevice

Figura 3.3: Codecs y capacidades.

3.3.1.7. Representación de un canal para protocolo de red

En este conjunto tenemos aquellas clases utilizadas por la OpenH323 para crear y manejar un canal de transporte de red. Es decir, permiten al programador cliente mantener un canal TCP/IP o UDP/IP [8]; crearlo, abrir el mismo, escuchar en él, escribir un mensaje en éste y, obviamente, cerrar el canal.

Clase	Función
H323Transport	Clase abstracta del grupo.
H323TransportIP	Implementa un canal de transporte IP para H.323.
H323TransportTCP	Implementa un canal de transporte TCP/IP para H.323.
H323TransportUDP	Implementa un canal de transporte UDP/IP para H.323.

Cuadro 3.9: Canales de Transporte.

[8]En realidad H323TransportTCP y H323TransportUDP, dan una interfaz para el manejo de estos protocolos, pero especializados para el trabajo con la recomendación H.323. Ver por ejemplo el método DiscoverGatekeeper() en el H323TransportUDP.

La primera clase del cuadro 3.9, ofrece una interfaz en común al resto de las clases de transporte y hereda de la clase PIndirectChannel de la PWLib.

La clase H323TransportTCP es especialmente utilizada por el *listener* del canal de señalización de llamada H.225 - H323ListenerTCP. Cuando un mensaje arriba en el puerto utilizado por el *listener*, una nueva instancia de la clase H323TransportTCP es creada para manejar este nuevo canal TCP que se está abriendo.

De forma similar es utilizada la H323TransportTCP por los *listeners* de RAS, H323GatekeeperListener y H323Gatekeeper.

3.3.1.8. Hilos de proceso especializados

En este grupo encontramos aquellas clases que fueron creadas para mantener un hilo de proceso separado, cumpliendo una misión especifica, ver cuadro 3.10. Se detallaran aquellas que presentaron mayor importancia para el proyecto.

Clase	Función
ToneThread	Generador de tono.
RTP_JitterBuffer	Implementa el jitter buffer RTP.
H323Listener	Clase abstracta para escuchar mensajes H.225/Q.931.
H323LogicalChannelThread	Inicio de canales lógicos, unidireccionales y bidireccionales.
H225CallThread	Inicio de conexión desde *endpoint* origen.
H323ConnectionsCleaner	Garbage collector de objetos H323Connection.
H225TransportThread	Inicio de conexión desde *endpoint* destino.
H245TransportThread	Inicio de canal de control de H.245.

Cuadro 3.10: Hilos de proceso especializados.

La clase H323ConnectionsCleaner tiene como función dormir la mayor parte del tiempo, hasta que se haya habilitado el semáforo de *wakeup*, luego inicia el proceso de borrado de las conexiones que se haya definido en el método CleanUpConnections() del H323EndPoint.

La clase H225CallThread es utilizada como hilo de proceso encargado de iniciar el primer intercambio de mensajes de establecimiento de llamada. Para el caso de la biblioteca, estamos hablando del primer setup de H.225/Q.931 y el ARQ que corresponda si el terminal está registrado con un *gatekeeper*. Esto está en el método H323Connection::SendSignalSetup(). Al final si no hay errores con este mensaje, deja a la H323Connection que se haya utilizado manejando el canal de señalización H.225/Q.931 establecido.

La H225TransportThread tiene una función similar a la anterior; se encarga de manejar el primer mensaje de establecimiento recibido por el *endpoint* y luego deja manejando la conexión a la H323Connection que se haya creado para soportar la nueva comunicación.

La H245TransportThread está encargada de realizar tareas de inicialización del canal de control H.245 de una comunicación y dejar a la H323-Connection de turno manejando este canal.

Por último la H323LogicalChannelThread sirve de inicializador de todos los canales lógico levantados luego de H.245 (incluidos canales da audio). Simplemente activa el loop de transmisión o recepción de cada canal lógico según sea el caso; canal unidireccional o canal bidireccional.

3.3.2. Documentación disponible

Desafortunadamente la documentación del proyecto OpenH323 y el material de ejemplo que esta posee es escaso. Además se suma el hecho de que la propia biblioteca se encuentra en constante desarrollo por lo que muchas veces los documentos que circulan por la Internet deja de estar al día con facilidad.

Las primeras fuentes de documentación son, obviamente, los propios sitios de Internet donde se encuentra el proyecto ([O323] y [VoxG]). También existe un pequeño material, tipo tutorial, el cual puede ser un punto de partida para comenzar a experimentar con las clases de la biblioteca [T323].

Pero el material de documentación más adecuado y actualizado para trabajar con las clases de la biblioteca se encuentra en los propios *headers* de la misma. En estos archivos los comentarios de las clases, métodos y miembros de datos, se encuentran escritos en formato DOC++. Lo que permite mediante su procesado obtener un conjunto de documentos html por los cuales es posible navegar entre las diversas clases, observar su herencia, etc.

Los programas que permiten hacer esto son dos; el DOC++ [DOCP], y el Doxygen [DGEN]. El último es el recomendado para las últimas versiones del proyecto OpenH323.

El material completo de documentación del proyecto OpenH323 que se entrega junto con este material fue generado con el DOC++ partiendo de la versiones 1.9.5 de la OpenH323 y la 1.3.5 de la PWlib.

3.4. Biblioteca PWLib

La función principal de esta biblioteca es servir de soporte para producir aplicaciones que puedan correr tanto en Windows como en los sistemas X-Window de Unix. Equivalence, la empresa donde surgió, la utilizaba regularmente para el desarrollo de sus programas. Una vez iniciado el proyecto OpenH323, y vista las ventajas de su utilización, se resolvió incorporarla al mundo del software libre.

La biblioteca se compone de algunos cientos de clases y no sólo sirve para la portabilidad de interfaces GUI entre distintos sistemas operativos, sino que brinda portabilidad de Entrada/Salida (E/S), multiprocesamiento (*multithreading*), portabilidad entre daemons de Unix y NT services, así como stack de protocolos de red y otros servicios. Todo esto está construido sobre clases contenedores básicas como: arrays, listas, listas ordenadas y

diccionarios (*hash tables*) que fueron desarrolladas antes del surgimiento de la Standar Template Library (STL), aunque con similares prestaciones.

El desarrollo de la biblioteca se mantiene en la actualidad incorporando nuevas clases y funciones. Este proyecto comenzó a trabajar con la versión 1.3.5, manteniendo esa versión durante el transcurso del proyecto, a pesar de que en la actualidad la versión es la 1.5.2. Es software libre y tiene licencia MPL lo que la convierte en una interesante herramienta de programación. Sin embargo, la documentación es bastante pobre y se reduce, principalmente, a comentarios dentro del código.

3.5. Arquitectura

El código está escrito en lenguaje C++ y las clases pueden ser agrupadas en las siguientes categorías:

- Contenedores
- E/S
- Threads y procesos
- GUI

3.5.1. Contenedores

Un contenedor es un objeto que contiene otros objetos; o en la mayoría de los casos, punteros a otros objetos. Es una solución usada en la programación orientada a objetos para lidiar con el problema de la creación de éstos en tiempo de ejecución. En los distintos lenguajes de programación existen distintos enfoques sobre cómo encarar el problema de los contenedores. En JAVA por ejemplo, se utiliza una jerarquía de clases basada en un único objeto lo que ayuda en el trabajo con contenedores dado que estos se diseñan para contener la clase base, y por ende, todas las demás. En C++ no existe una clase base de la que todas las demás hereden y se utilizan *templates*, de forma de que el propio compilador realiza los cambios necesarios para crear contenedores de los objetos específicos. La PWLib recoge en parte las dos soluciones, utilizando estructuras de clases basadas en el objeto PObject para algunas clases, así como *templates* y otras herramientas, en otros casos. El programador cliente dispone de macros para la definición de los distintos contenedores, y en base al sistema en el que se está compilando y al compilador utilizado, se decide si se utilizan *templates,* o no, abstrayendo al programador de la implementación.

PContainer es la clase de la que heredan las distintas implementaciones de los contenedores. Los principales servicios que brinda esta clase son: un contador del número de objetos que se contienen, y un chequeo de las referencias que existen al contenedor de modo que sólo se destruyan objetos cuando no existen más referencias a ellos. Heredando de esta clase se definen: listas, diccionarios, arrays, etc.

A partir de PContainer se definen, entre otras: PCollection y PAbstractArray. PCollection es un contenedor de clases que descienden de PObject.

En realidad contiene punteros a las instancias y no los objetos en sí. La vida del objeto en el contenedor debe ser considerada por el programador cliente ya que puede ser borrado automáticamente cuando se remueve o el contenedor es destruido, dependiendo de la configuración dada por el programador. Las distintas clases que heredan de PCollection determinan la forma en que se accede a los objetos, por ejemplo, el contenedor puede ser un array que permite un rápido acceso aleatorio a expensas de una lenta inserción o remoción, o puede ser una lista, con el comportamiento opuesto. Las clases más destacadas que heredan de PCollection son: PHashTable, PSet, PDictionary, PArray, PSortedList, PList, PStack, etc.

Otra clase a destacar que desciende de PContainer es PAbstractArray. Esta clase es un array de bloques arbitrarios de memoria. Estos pueden ser desde simples bytes hasta estructuras más complejas pero sin incluir clases que precisen de construcción o destrucción. Entre las clases que heredan de esta se destacan: PString y PBYTEArray. PString representa un array de caracteres, mientras que PBYTEArray es un array de caracteres sin signo. De esta última heredan dos clases muy interesantes para el desarrollo del proyecto: PPER_Stream y PBER_Stream, las cuales brindan funciones para codificar y decodificar trenes de bits según PER (*Packet Encoding Rules*) y BER (*Basic Encoding Rules*) de ASN.1, siendo muy útiles para trabajar con protocolos especificados según dicha norma.

3.5.2. Clases para E/S

Este grupo de clases permite el trabajo con los distintos métodos de entrada y salida existentes. La clase PChannel define la semántica de un canal de entrada/salida, pudiendo ser un puerto serial, una tubería (pipe), un socket de red, o simplemente un archivo. El modelo general del canal es que éste recibe o manda trenes de bytes. El acceso a estos trenes de bytes se realiza a través de funciones que permiten distintos tipos de transferencias: directas, por medio de buffers, o asíncronas. El modelo también conlleva el estado del canal, el cual puede estar abierto o cerrado. Una instancia de canal que está cerrado contiene información suficiente para describir el canal pero no utiliza o monopoliza recursos del sistema. Lo contrario sucede cuando el canal está abierto.

Entre las clases que heredan de PChannel se destacan: PSocket, PFile, PSoundChannel, PVideoChannel, PSerialChannel.

PSocket representa un canal de comunicaciones en una red y está basado en los conceptos de la biblioteca de sockets de Berkley. Es una comunicación bidireccional hacia un puerto en una máquina remota. PWLib contempla su uso con distintos protocolos de red, como son IP (PIPSocket) o IPX (PIPXSocket), y hasta existe una clase para representar un socket a nivel de Ethernet (PEthSocket). También existen las clases PTCPSocket y PUDPSocket para representar canales que utilizan transporte TCP y UDP respectivamente.

PFile y sus clases descendientes PTextFile y PStructuredFile representan un archivo de disco que puede ser portado para distintas arquitecturas de procesador. Este tipo particular de E/S tiene ciertos atributos. El modelo básico para los archivos es una secuencia de bytes que tiene un nombre y

que persiste dentro de una estructura de directorios. La transferencia de datos se realiza a la posición actual dentro del archivo.

Otra clase a destacar en este grupo es PInternetProtocol que representa un socket TCP/IP para nivel de aplicación. De ella heredan: PHTTP, PPOP3, PFTP, PSMTP. Estos protocolos intercambian comandos y respuestas en forma de texto de una manera estándar. Estas clases facilitan la implementación de clientes o servidores que hablen estos protocolos.

3.5.3. Hilos y procesos

En cuanto al soporte de procesos e hilos de ejecución, PWLib brinda apoyo para su implementación en los distintos sistemas operativos mientras que da funcionalidades para tratar con los procesos en sí, por ejemplo: parseo de argumentos, configuración de programas, trazabilidad, etc.

Un hilo de ejecución es un flujo independiente de instrucciones de procesador. Difiere de un proceso en tanto que a éste se le asocia un espacio de direcciones y una forma de disponer de recursos, mientras que los hilos comparten memoria y recursos en el ámbito del proceso en el que corren. Un proceso siempre tiene al menos un hilo. En la biblioteca un hilo se representa por la clase PThread de la que hereda PProcess la cual representa un proceso, un programa corriendo en el ámbito de un sistema operativo.

La implementación de un hilo es dependiente de la plataforma. No todos los sistemas operativos soportan hilos concurrentes dentro de un proceso, algunos ni siquiera soportan procesos concurrentes. A modo de ejemplo: MS-DOS no soporta *multi-threading* (hilos concurrentes) ni multiprocesamiento, Microsoft Windows tiene multiprocesamiento cooperativo pero no *multi-threading*, mientras que Windows NT y algunos Unixes son totalmente multiproceso y multi-thread con priorización de tareas. Si la plataforma no soporta *multi-threading* la biblioteca usa una técnica de rutinas cooperativas para implementarlo. Esto requiere que cada hilo de ejecución dentro de un proceso ceda voluntariamente el control a los otros hilos. Esto ocurre si el hilo es bloqueado dentro de una función de E/S en una función de PChannel o por medio de funciones específicas implementadas en PThread (Yield()).

Una clase interesante es PServiceProcess que es un proceso que corre como servicio background, o sea, un servicio en Windows NT, un demonio en Unix o una aplicación escondida en Windows.

La biblioteca también brinda los medios necesarios para la sincronización de hilos. Esto es muy necesario ya que pueden existir partes del código que no deben ser ejecutadas simultáneamente por varios hilos, o se precisa sincronizar la ejecución de los hilos entre sí. Esto se logra a través de semáforos definidos por medio de la clase PSemaphore. Dicha clase contiene un contador, un valor máximo para dicho contador y las funciones Wait() y Signal(). La función Wait() implementa una operación tal que si el contador del semáforo es mayor que cero, decrementa el contador y sale; si el contador es cero, se detiene el hilo de ejecución, que queda esperando que otro hilo llame a Signal(). La función Signal() desbloquea el primer hilo si existen hilos esperando; si esto no ocurre y el contador es menor que el máximo, se incrementa el contador. Las distintas combinaciones del contador con su

máximo, y el uso de las funciones Wait() y Signal() determinan el tipo de mecanismo de sincronización que se empleará.

La biblioteca dispone de varias clases que heredan de PSemaphore y que cubren los distintos mecanismos de sincronización. PMutex sirve para crear zonas de exclusión mutua, código que sólo puede ser ejecutado por un hilo a la vez. PSyncPoint se usa para bloquear un hilo de ejecución hasta que otro hilo lo habilite luego de ejecutada cierta parte del código. PCondMutex implica una zona de exclusión mutua pero sólo en el caso de que cierta condición sea satisfecha. PSyncPointAck se usa para enviar señales a otro hilo mientras se espera un reconocimiento del procesamiento de la señal. Se utiliza para iniciar una acción en otro hilo y esperar que la acción sea completada. Existen otras variedades: PWaitAndSignal, PReadWriteMutex, etc

También se brindan una serie de clases para tratar con contenedores y objetos en un ámbito de programación concurrente haciendo seguro el acceso a los objetos. En este contexto se define la clase PSafeCollection que constituye una colección *thread-safe* de la que a su vez heredan PSafeList y PSafeDictionary, variantes seguras de una lista y un diccionario. PSafeObject representa un objeto *thread-safe* y PSafePointer es una clase para manejar instancias del tipo PSafeObject con funciones que se encargan automáticamente de modificar los contadores de referencias, así como setear el estado del objeto; por ejemplo, si puede ser leído o no, si puede ser escrito, etc.

Por último existen toda una serie de clases para trabajar con procesos como: PConfig, que representa la configuración de un programa; PArgs y PArgsList, que ayudan en el parseo de argumentos; PTrace, para realizar trazas durante la ejecución de un programa; así como implementación de timers, relojes, etc.

3.5.4. GUI

Un aspecto interesante de la PWLib es la posibilidad de desarrollar una interfaz gráfica sin preocuparse del sistema operativo en el que correrá haciéndola independiente de éste. Existen muchas clases para implementar interfaces GUI aunque la biblioteca no está muy completa por el momento. La implementación para Win32 es bastante usable, así como la implementación de XLib, mientas que una implementación de Motif está en proceso. Esta parte de la biblioteca no ha sido muy estudiada dado que no se utilizó una interfaz GUI para el mismo.

3.6. Parser de ASN.1

Un componente fundamental de la biblioteca es un programa llamado *asnparser*. Dicho programa es un parseador de ASN.1 que crea código C++ a partir de las especificaciones ASN.1 de la norma. Esto permite un rápido desarrollo de las nuevas funciones que se incorporan en las nuevas versiones de la norma, abstrayendo al programador de los detalles engorrosos de la notación ASN.1. Estas herramientas son indispensables cuando se trabaja con ASN.1 haciendo virtualmente imposible el trabajo sin ellas.

Además, existen clases que definen los tipos de datos que se encuentran en la notación ASN.1. PASN_Object define el comportamiento común de todos los objetos ASN.1. De ella heredan toda una serie de clases con las características particulares de cada tipo de datos: PASN_Sequence, PASN_Real, PASN_Null, PASN_Enumeration, PASN_Integer, PASN_Array, PASN_Choice, PASN_Boolean, etc. Una vez corrido asnparser y en base a las especificaciones ASN.1 de la norma se crean automáticamente toda una serie de clases con los mensajes y los campos de éstos que heredan de las clases que representan los tipos de datos de ASN.1. Luego el programador se olvida de ASN.1 y trabaja con los mensajes y los campos en ellos como tipos de datos de su programa despreocupándose de la codificación de los mensajes para la transmisión o recepción. A su vez, tiene la ventaja que ante variaciones en la norma, el programador dispone automáticamente de los nuevos tipos de mensajes o campos como nuevas clases en su programa pudiendo utilizarlos a conveniencia.

3.7. Servicios que brinda a OpenH323

El desarrollo de PWLib se ha visto muy influenciado por la evolución de OpenH323, y muchas de las clases que se han ido incorporando responden a necesidades surgidas del trabajo en OpenH323. Como ejemplo de lo anterior se podría nombrar todo el trabajo realizado para *asnparser*, para el soporte de programación concurrente, etc. En cualquier desarrollo del stack de H323 es fundamental contar con herramientas de multiprocesamiento y parsers de ASN.1. Las ventajas de PWLib se basan en todas las posibilidades nombradas anteriormente, en el hecho de que permite portabilidad entre distintos sistemas operativos y en que se pueden desarrollar programas bastante complejos haciendo uso solamente de PWLib, lo que determina bastante independencia frente al uso de bibliotecas específicas de ciertos sistemas operativos y compiladores.

3.8. Documentación de PWLib

Lo mismo que se dijo respecto a la documentación de OpenH323 es aplicable a la documentación de PWLib. No existe un trabajo serio de documentación de la biblioteca limitándose ésta a comentarios en el código con un formato específico para extraerse a través de DOC++. A su vez, las clases más recientes no tienen ninguna documentación, y sólo se cuenta con el código para conocer su funcionamiento. La lectura del código fuente de ciertos programas muy simples que vienen como ejemplos constituyen la mejor aproximación a la comprensión de cómo usar las funciones de la biblioteca, con las dificultades que ello conlleva.

También cabe destacar la existencia de listas de correos que proveen los desarrolladores para el intercambio entre los usuarios, siendo una fuente de información importante cuando se tiene cierto conocimiento de la biblioteca, ya que el nivel usado en las discusiones presupone tal conocimiento[O323].

Capítulo 4

Desarrollo del programa

Introducción

El presente capítulo detallará el desarrollo de software realizado con la finalidad de construir la entidad *gatekeeper* de la recomendación H.323, implementando sus funcionalidades básicas y algunas opcionales. Como elementos constructivos se tomaron las clases de las bibliotecas incluidas en el proyecto OpenH323, las cuales fueron el centro del capítulo anterior.

Como ya fue dicho el lenguaje de programación utilizado fue C++ y la mayor parte del desarrollo fue hecho utilizando el ambiente de desarrollo de Microsoft Visual C++ 6.0. Aunque el código también fue portado a GNU/Linux en diferentes etapas del proyecto.

El capítulo contiene una primera sección en la cual son planteados los objetivos originales del proyecto y la forma en que se buscó alcanzar estos dividiendo el trabajo en cuatro etapas bien delimitadas.

Luego se pasa a una descripción mas detallada de cada etapa, con sus planteos específicos, dificultades encontradas y objetivos alcanzados. Cada etapa a su vez dejó nuevos requerimientos que fueron incorporados a las siguientes.

La última sección detalla el código final y como fueron implementados los distintos requerimientos del proyecto.

4.1. Análisis y división del proyecto en etapas

Después de un primer estudio de los objetivos originales planteados para el proyecto, se decidió dividir el desarrollo en cuatro etapas. Las primeras contendrían los requerimientos más básicos y servirían de plataforma para las subsiguientes etapas. Cada módulo o etapa daría como resultado un programa ejecutable, que podría ser probado con diferentes tipos de terminales.

Las cuatro etapas y sus objetivos originales se encuentran en el cuadro 4.1.

Con la evolución del proyecto, la finalización de cada módulo a su vez fue dejando nuevos requerimientos para los siguientes, que no habían sido considerados originalmente o que surgieron como necesidades obvias; log

Etapa del desarrollo	Objetivos buscados.
1.	El requerimiento principal de esta etapa es obtener un programa que cumpla con las primeras funciones básicas de un *gatekeeper*, manejo del protocolo RAS y resolución de direcciones. El programa conocerá y responderá mensajes a través del protocolo RAS, sin que ello implique que resuelva todos los aspectos internos involucrados. Por ejemplo: no necesitan estar implementadas políticas de admisión.
2.	En esta etapa se desarrollarán las funciones locales del *gatekeeper* que le permitan cumplir con las tareas que le son asignadas (por ejemplo: desarrollo de políticas de admisión, registro de llamadas, manejo de los usuarios registrados, etc.) En este proceso se juntarán estos objetivos con la señalización RAS anteriormente desarrollada.
3.	En esta etapa se cubrirán los aspectos relacionados con el manejo de las APDU de H.450 para su utilización por parte del *gatekeeper* como "proxy" de H.450. Se desarrollarán básicamente los aspectos relacionados con la recomendación H.450.1. Se incluirán aspectos de diseño en cuanto a los componentes que se deben desarrollar para cumplir con los servicios suplementarios.
4.	Se desarrollarán las funciones de la recomendación H.450.2 y H.450.3 que competen al *gatekeeper*, así como funciones adicionales que le sean indispensables internamente para cumplir con los servicios suplementarios; desvío de llamada y transferencia.

Cuadro 4.1: Etapas del proyecto.

de llamadas, archivo de configuración, etc. Pero se profundizará sobre estos replanteos en las siguientes cuatro secciones, cada una dedicada a una etapa de desarrollo específicamente.

4.2. Etapa uno

El objetivo de esta etapa consistió en poner en funcionamiento un *gatekeeper* básico por medio de los recursos provistos por la biblioteca. Se debía lograr un *gatekeeper* que implementara las funciones básicas que la norma le confiere: registración, admisión de llamadas y manejo de ancho de banda. Básicamente dicho *gatekeeper* debía procesar los distintos mensajes RAS que se intercambian con los terminales para que éstos se registren y puedan cursar llamadas. Esto implicaba que al finalizar el módulo los terminales debían registrarse con el *gatekeeper*, solicitar autorización para cursar una llamada entre ellos, alertar al gatekeeper sobre la desconexión de la llamada y desregistrarse con el *gatekeeper*. El *gatekeeper* también debía procesar

los mensajes de solicitud de cambio de ancho de banda por parte de los terminales. Se excluyó el procesamiento de los mensajes de localización (LRQ, *location request*) utilizados para la resolución de direcciones entre distintas zonas por no estar incluidos dentro de la definición del proyecto, si bien parte de dicha funcionalidad es provista por la biblioteca.

Si bien la biblioteca implementa muchas de estas funciones la dificultad radicaba en la puesta en funcionamiento del programa y en el uso de los recursos provistos por la biblioteca dada la poca información con que se contaba, así como el aprendizaje de las herramientas de programación utilizadas (entorno de programación, etc).

4.2.1. Las características obtenidas del módulo 1, interfaz de comandos

Como resultado del trabajo en este módulo se logró un programa que cumplía con los requerimientos fijados. El *gatekeeper* se invocaba desde la línea de comandos y se le podían pasar algunos argumentos que brindaban cierta variedad de configuración. Entre las variables que se podían configurar mediante los argumentos se encontraban: la dirección IP en la que atendería el *gatekeeper*, el identificador por el que se daría a conocer con los terminales, el puerto RAS, así como el nombre y el nivel de un archivo de traza. La no existencia de dichos argumentos implicaba la utilización por parte del *gatekeeper* de ciertos valores por defecto.

Las bibliotecas cuentan con buenas herramientas de debug y generación de trazas. Por esta razón, se agregó en este módulo la posibilidad de que el usuario dispusiera de un archivo de traza que, básicamente, muestra según el nivel que se desee, las distintas acciones que realiza el *gatekeeper*, así como los mensajes que intercambia con las otras entidades. Dicha herramienta es muy útil ante problemas puntuales de interoperabilidad. Cada vez que se ejecuta el programa se genera un archivo de traza que detalla el funcionamiento del *gatekeeper* y los distintos mensajes que procesa, quedando guardado con un nombre que detalla la fecha y la hora del comienzo de la ejecución o, con el nombre que el usuario especifique, creándose un archivo distinto por cada ejecución o sobreescribiéndose el mismo si el usuario lo desea.

La existencia de dicha traza es opcional y es configurable a través de una bandera que se setea en la compilación del programa. Si la versión ejecutable no dispone de trazabilidad (dada la configuración durante su compilación), los argumentos que refieren al nombre y nivel del archivo de traza no son tenidos en cuenta.

Para constatar el correcto funcionamiento del programa se realizaron pruebas con dos terminales. Mediante dos computadoras personales conectadas a través de un cable UTP cruzado se probó el funcionamiento de dos terminales implementados en software y el *gatekeeper*. Una de las computadoras corría un terminal, mientras que en la otra corrían el *gatekeeper* y otro terminal. Los terminales utilizados fueron openphone y ohphone. Los dos son implementados con la biblioteca OpenH323 y se diferencian, básicamente, porque el primero dispone de una interfaz gráfica de usuario (GUI). Se realizaron algunas modificaciones al código del ohphone para que

pudiera solicitar modificaciones de ancho de banda ya que ninguno de los dos terminales lo soportaba con las versiones utilizadas.

4.3. Etapa dos

En esta segunda etapa del proyecto se buscó sumar un grupo de funcionalidades obligatorias del *gatekeeper*, que se detallan a continuación, así como incorporar algunas que surgieron como una clara necesidad de la primera etapa de desarrollo.

Todos los objetivos iniciales de está etapa fueron alcanzados con éxito, dejando de lado la posibilidad de implementar la autenticación de los usuarios, debido a que requería mayor tiempo de estudio del protocolo H.235 del que se disponía.

4.3.1. Los objetivos y funcionalidades alcanzadas

Los objetivos fundamentales comprendieron la adición de tres nuevas funcionalidades al *gatekeeper*: políticas de admisión de registro, políticas de admisión de llamadas tal cual fue especificado previamente en los objetivos del proyecto y, un archivo de registro de llamadas (Log de llamadas). Asimismo, contempla dos nuevas funcionalidades que surgieron como necesidades obvias del modulo 1; la posibilidad de configurar el *gatekeeper* mediante un archivo de texto externo al ejecutable y la posibilidad de contar con una interfaz de comandos y despliegue de información durante la ejecución del mismo.

También se estudió la posibilidad de implementar la autenticación con el *gatekeeper* durante el registro de los terminales; mediante *"alias* de usuario" y *"password"*, pero esto último ha quedó postergado para siguientes etapas de desarrollo debido a la necesidad de adicionar tiempo extra de estudio al tema.

4.3.1.1. Políticas de admisión de registro

En este etapa del desarrollo la zona administrada por el *gatekeeper* se delimitó de acuerdo con los siguientes criterios:

1. De acuerdo a la dirección IP del host desde el cuál se desea registrar el terminal.
2. De acuerdo a un alias propio del usuario.
3. Combinaciones de las anteriores

Una vez recibido el GRQ del terminal, se chequea la información en este paquete y se verifica que pertenezca a la zona de administración habilitándose su registro.

Estas zonas serán configuradas a través del archivo de configuración, el cual se detalla mas adelante.

4.3.1.2. Políticas de admisión de llamadas, clases de servicio

Se definió clases de servicios a las cuales los usuarios registrados pertenecerán y que determinarán que tipo de facilidades les son admitidas. Cada clase de servicio tiene asociados distintos parámetros, los cuales determinan cada clase en particular. Los parámetros y sus posibles valores son los siguientes:

Entrantes = si/no

Salientes = si/no

lista_de_restricciones = no/lista de terminales separados por coma.

Los dos primeros parámetros indican la posibilidad que tiene el usuario de recibir y realizar llamadas, respectivamente. El tercero es una lista de terminales a los cuales no le será permitido llamar. Estos parámetros, son almacenados en el archivo de configuración del *gatekeeper* en el sector corres-pondiente a una clase de servicio.

Luego a cada usuario, también registrado en el archivo de configuración, se le asignará una de estas clases.

Toda esta información es leída al inicio del programa y guardada en estructuras internas para su posterior consulta durante la llegada de mensajes ARQ.

4.3.1.3. Log de llamadas

Se implementó un registro (log) de llamadas establecidas a través del *gatekeeper*, el cual es generado a partir de los mensajes RAS que lleguen al *gatekeeper* del tipo ARQ/ACF y DRQ/DCF.

El log es un archivo de texto que se genera en el mismo directorio donde reside la aplicación, cuyo nombre es; GKMOD2.LOG. Cada línea del log se corresponde con una única conexión efectuada a través del *gatekeeper*, luego de que la misma haya sido finalizada por los terminales involucrados.

A su vez, cada línea o registro del log contiene la información de la conexión detallada a continuación:

1. Origen de la conexión.
2. Destino de la conexión
3. Fecha y hora de la conexión
4. Fecha y hora del fin de la conexión
5. Ancho de banda solicitado para la conexión

Esta información es presentada en la línea de registro un campo a la vez y separados por un carácter delimitador, por ejemplo una coma. Así el formato de cada línea es el siguiente:

Origen,Destino,Fecha_hora_de_inicio,Fecha_hora_de_fin,Ancho_de_Banda

Origen: n caracteres que contienen el alias del terminal e IP, con formato:

AAAAA:IIIIII

AAAAA : alias del terminal, número de caracteres variable, según el alias del terminal. Diferentes alias separados por espacios blancos.

IIIIII : IP del terminal, 4 grupos de 3 caracteres separados por punto.

Destino: n caracteres que contienen el alias del terminal e IP, con formato:

AAAAA:IIIIII

AAAAA : alias del terminal, número de caracteres variable, según el alias del terminal. Diferentes alias separados por espacios blancos.

IIIIII : IP del terminal, 4 grupos de 3 caracteres separados por punto.

Fecha_hora_de_inicio: un total de 15 caracteres que dan fecha y hora de inicio de la conexión, de la forma:

AAAAMMDD HHMMSSPP

AAAA : Año expresado con 4 caracteres.

MM : Mes, 2 caracteres.

DD : Día, 2 caracteres:

Fecha y hora separados por un único espacio en blanco.

HH : hora, 2 caracteres.

MM : minutos, 2 caracteres.

SS : segundos, 2 caracteres.

PP : dos caracteres para identificar a.m. o p.m.

Fecha_hora_de_fin: un total de 15 caracteres que dan fecha y hora de finalización de la conexión, de la forma:

AAAAMMDD HHMMSSPP

AAAA : Año expresado con 4 caracteres.

MM : Mes, 2 caracteres.

DD : Día, 2 caracteres:

Fecha y hora separados por un único espacio en blanco.

HH : hora, 2 caracteres.

MM : minutos, 2 caracteres.

SS : segundos, 2 caracteres.

PP : dos caracteres para identificar a.m. o p.m.

Ancho_de_Banda: Total de 5 caracteres para mostrar el ancho de banda bidireccional solicitado para la conexión, en unidades de 100 bits por segundo.

4.3.2. Otras características obtenidas en módulo 2

En lo que sigue se realiza una descripción de las dos nuevas características incorporadas al *gatekeeper* durante esta etapa.

4.3.2.1. Archivo de configuración

Al iniciarse el *gatekeeper*, éste carga desde un archivo de configuración un conjunto de datos que determinarán su forma de funcionamiento.

Dicho archivo es del tipo INI y esta dividido en diferentes secciones, las que se detallan a continuación.

A. Sección General

En esta sección se encuentran parámetros generales del *gatekeeper*, los cuales se aplican en caso de no ser proporcionados en la línea de comandos al iniciar el programa.

Los campos presentes en esta sección son:

traza : nombre del archivo de traza

nivelTraza : nivel de la traza.

dirección IP : dirección IP de la interfaz en la que estará escuchando el *gatekeeper*.

puertoRAS : puerto TCP donde escuchará mensajes RAS.

nombreGK : identificador del *gatekeeper*.

B. Sección Zona

Esta sección contiene los datos necesarios para la determinación de la zona a ser atendida por el *gatekeeper*.

Contiene los siguientes campos:

modo :

La zona (terminales que se pueden registrar con el *gatekeeper*) puede ser definida mediante direcciones IP, mediante alias, o ambos, esto se puede elegir con el modo de funcionamiento, asignándole los valores 0,1 y 2 respectivamente.

redIP :

Es la dirección de la subred a la cual deben pertenecer los terminales.

redmascara :

Máscara utilizada para saber si una dirección IP determinada pertenece o no a la subred indicada en el campo anterior.

Estos dos últimos campos pueden repetirse varias veces y se diferenciarán agregando un número en forma creciente al final del nombre del campo, por ejemplo redIP1, redIP2, etc.

Dentro de los alias pueden haber nombres de usuario alfanuméricos, o números telefónicos según la recomendación E.164, en caso de que sean de éstos últimos existen otros campos que intervienen en la determinación de la zona y se detallan a continuación.

chequeo_cantidad_de_digitos :

Puede tomar los valores si o no, esto indica si se controlará o no la cantidad de dígitos del número, es decir serán números de largo fijo o no.

cantidad_de_digitos :

Es un número entero que indica el tamaño (cantidad de dígitos) de los números o alias.

chequeo_caracteristica :

Se pueden definir características de forma análoga a lo que es una subred, de forma tal que todos los números (alias) cuyos primeros dígitos coincidan con la característica pertenecerán a la zona.

Este campo indica si existirá o no característica, y puede tomar los valores si o no.

caracteristica :

En caso de existir característica, ésta es definida en este campo, pudiendo ser más de una separadas por coma.

C. Sección Clase

Luego se tienen varias secciones donde cada una define una clase de servicio, a la que pertenecerá un cierto grupo de usuarios, cada usuario deberá pertenecer a una y sólo una clase de servicio. Estas se diferencian entre si agregando un número (consecutivo creciente) al final del nombre, por ejemplo clase1, clase2, etc.

Los campos presentes en cada una de ellas son:

tipo:

Identifica el tipo de clase de servicio, en este módulo del programa el tipo será fijo y tendrá el valor servicio_básico

entrantes:

Indica si los usuarios pertenecientes a esta clase de servicios podrán recibir llamadas o no, los valores posibles para este campo son si y no.

salientes:

De forma similar al anterior, este indica si los usuarios podrán efectuar llamadas, los valores posibles son si y no.

lista_de_restricciones:

Los usuarios pueden tener uno o un grupo de destinos restrictos, es decir usuarios a los que no se los permite llamar, si no existen restricciones para los usuarios de esta clase, este campo contendrá el valor no, en caso contrario este campo contendrá los usuarios restrictos, ya sean alias o identificadores E.164, en este último caso si se trata de todos los números con determinada característica no es necesario especificarlos por extensión sino que se puede dar la característica seguida por el caracter '*'.

suplementarios1:

suplementarios2:

En este módulo los campos anteriores contendrán el valor no, son campos reservados que se utilizarán posteriormente cuando se implementen ciertos servicios suplementarios.

Las diferentes clase de servicio se pueden lograr mediante combinaciones de los valores de los campos de estas.

D. Sección Usuario

Se tiene una sección por usuario, con los siguientes campos.

nombre:

Es un string que representa el nombre del usuario, es opcional.

alias:

Contendrá el alias con el que se registrará el usuario en caso de funcionar en modo alias, En caso de ser necesario se podrá asignar un grupo de terminales por su prefijo seguido del carácter '*'.

direccionIP:

Contendrá la dirección IP del usuario para el caso de funcionamiento en modo IP.

clase:

Contendrá la clase de servicio a la que pertenecerá el usuario, este campo solo podrá tener un valor, en caso de encontrarse vacío el usuario se podrá registrar pero no podrá realizar ninguna acción.

A continuación se da un ejemplo de un archivo de configuración para este módulo.

```
[general]
;traza=traza.txt
nivelTraza=4
;direccionIP=192.168.46.6
direccionIP=127.0.0.1
puertoRAS=1719
nombreGK=GatekeeperH323Modulo2
[zona]
; modo = 0 soloIP
; modo = 1 soloAlias
; modo = 2 AliasEIP
modo=1
redIP1=127.0.0.0
redMascara1=255.255.255.0
redIP2=192.168.46.0
redMascara2=255.255.255.0
chequeo_cantidad_de_digitos=no
cantidad_de_digitos=4
chequeo_caracteristica=no
caracteristicas=12,33,222
[clase1]
;solo admite llamadas entrantes
tipo=servicio_basico
entrantes=si
salientes=no
lista_de_restricciones=no
suplementarios1=no
suplementarios2=no
usuarios= 333*,usuario1,usuario2
[clase2]
;admite llamadas entrantes y salientes sin restricciones
tipo=servicio_basico
entrantes=si
salientes=si
lista_de_restricciones=no
suplementarios1=no
suplementarios2=no
usuarios=usuario3,usuario5
[clase3]
;admite llamadas entrantes y salientes con restricciones
tipo=servicio_basico
entrantes=si
salientes=si
;la lista de restricciones son alias o ID E.164
lista_de_restricciones=12*,usuario5,usuario1
suplementarios1=no
suplementarios2=no
usuarios=usuario4,12*,332*
```

Cuadro 4.2: Ejemplo de Archivo de Configuración del módulo 2

4.3.2.2. Interfaz de comandos

Una vez iniciado el programa gkmod2, éste abre una sencilla consola para el ingreso de comandos y la obtención de información del *gatekeeper* en tiempo real. Esto se consigue a través del siguiente grupo de comandos:

RCG [-a archivo_de_configuración]

Este comando permite la recarga del archivo de configuración, por defecto será el de nombre "config.ini" ubicado en el mismo directorio que el programa, en caso de no ingresar el parámetro opcional -a y el camino completo a un nuevo archivo de configuración.

Esta opción permite cambiar la configuración del *gatekeeper*; habilitación de nuevos usuarios, modificación de valores en las secciones de usuarios, etc, y luego activar estos cambios en el archivo de configuración, por medio de la recarga del mismo.

Los cambios que se hagan sobre usuarios registrados, tomaran efecto la próxima vez que estos establezcan una nueva conexión, o vuelva a registrarse con el *gatekeeper*. Esto es, si el usuario A mantiene una conexión con un destino - usuario B - el cual pasa a estar dentro de la lista de restricciones del usuario A, esta conexión no es interrumpida, sino que las próximas son inhabilitadas.

LOG

Comando que permite habilitar y deshabilitar el despliegue del Log de llamadas por consola. Funciona como conmutador que cambia el estado de la opción de salida del Log de llamadas. Si el Log esta inhibido, luego de aplicar este comando se habilita y viceversa.

Para confirmar el estado en el cual se encuentra esta opción, luego de ingresar el comando se muestra si la opción esta habilitada o no, mediante los mensajes;

>Log por pantalla habilitado

o

>Log por pantalla inhibido

Luego de habilitado el Log y luego de la finalización de cada conexión se desplegará información de esta, con el siguiente formato:

Datos llamada terminada.
Origen (Alias:IP) =
Destino (Alias:IP) =
Fecha_hora_de_inicio =
Fecha_hora_de_fin =
Ancho_de_Banda (x100 b/s) =

El formato de cada campo respeta la misma forma que el Log de llamadas por archivo.

HLP

Simplemente muestra una breve ayuda sobre los comandos disponibles y su formato correcto.

TRZ [-n nuevo nivel traza]

Cambia el nivel de traza al nuevo nivel indicado por el comando. El parámetro -n es opcional y si no está presente se deja el nivel de traza en el nivel mínimo

FIN

Comando usado para salir de la ejecución del programa.

4.3.2.3. Inicio del programa

Para poner en funcionamiento el *gatekeeper* se debe ejecutar el archivo gkmod2, pudiéndose pasar un grupo de parámetros en su inicialización a través de línea de comando según se detalla a continuación, en caso de que no le sean proporcionados tomará ciertos valores por defecto definidos previamente.

La forma de los parámetros es la siguiente;

gkmod2 [- h help] [- i dirección IP de la interfaz] [- n identificador del gatekeeper] [- p puerto del canal RAS] [- t nombre del archivo de traza] [- T nivel de la traza] [-f archivo de configuración]

Opciones. Los parámetros requeridos por el programa Gkmod2 deben ser proporcionados mediante línea de comandos con las siguientes opciones.

- i Especifica la dirección IP de la interfaz en la que estará escuchando el *gatekeeper*. Si no se proporciona una, por defecto ésta será 127.0.0.1.

- n Especifica el identificador que tendrá el *gatekeeper*, es un string de caracteres por el cual se lo reconocerá.

- p Puerto TCP donde escuchará mensajes RAS, por defecto tomará el 1719.

- t Especifica el nombre del archivo donde se escribirá la traza durante el funcionamiento.

- T Determina el nivel de la traza, esto es el nivel de detalle, pudiendo tomar valores del 1 al 6, siendo el último el de mayor detalle. Por defecto tomará 1.

- f Especifica el nombre del archivo de configuración que debe cargar. Por defecto tomará el archivo config.ini

4.4. Etapa tres

4.4.1. Replanteo de los objetivos

En principio, los objetivos de este módulo eran el estudio y manejo de las APDU de H.450. El anterior trabajo con la biblioteca permitió un conocimiento más acabado de sus características, determinándose que ésta no disponía de los elementos necesarios para manejar la señalización de llamada a través del *gatekeeper*. Dado que dicho comportamiento era imprescindible para la implementación de los servicios suplementarios se replantearon los objetivos de este módulo para dedicarlo a implementar el enrutamiento de la señalización. Todo lo relacionado con los servicios suplementarios se trasladó al módulo 4.

La señalización que debía rutear el *gatekeeper* se restringía a H.225, no incluyéndose H.245 ya que los mensajes de H.450 se cursan a través de Q.931/H.225.

También se agregaron nuevas funcionalidades y se realizaron modificaciones y correcciones a lo hecho anteriormente, corrigiendo ciertos aspectos de configuración que, del trabajo con el *gatekeeper*, resultaron poco satisfactorios.

4.4.2. Las características obtenidas del módulo 3

Se modificó la forma en que se generaba el log de llamadas aprovechando la nueva información que se obtiene al manejar la señalización H.225/Q.931. Si el *gatekeeper* rutea la señalización de llamada, el log se genera a partir de los mensajes "connect" y "release complete" de Q.931, obteniéndose datos más precisos.

Se cambió la forma como se define la zona que administra el *gatekeeper*. Ahora, sólo existen dos modos de funcionamiento: "soloAlias" y "aliasEIP". El primero determina la zona a través de una lista de alias que se pueden registrar con el *gatekeeper*, mientras que el segundo, no sólo toma en consideración los alias, sino que, además, chequea las direcciones IP que utilizan los terminales.

También se cambió la forma como se representan los rangos de números en el archivo de configuración. Ahora, cada caracter punto (".") en el nombre de alias identifica un dígito, cuando el alias comienza por un dígito ("0" a "9") o por un caracter punto (".").

Por último, se agregó la opción de que la traza pueda ser desplegada por consola.

Todos estos cambios reflejan modificaciones y mejoras en cuanto a las funciones que realiza el *gatekeeper* así como a la interfaz de usuario. Evidentemente, esto fue posible mediante cambios en la arquitectura del *software*, no visibles para el usuario.

Cabe hacer notar, que se logró compilar el programa para su utilización con el sistema operativo Linux mediante pequeños cambios que adecuan el código al nuevo compilador.

4.4.2.1. Ruteo de la señalización de llamada H.225/Q.931 por gatekeeper

A partir de este módulo se podría decir que el *gatekeeper* se comporta como un "proxy" de Q.931, pasando los mensajes que intercambian los terminales. De todas formas, no hay que olvidar que el *gatekeeper* actúa como una nueva entidad de Q.931 y, por lo tanto, debe cumplir ciertos requerimientos que le impone la norma. A modo de ejemplo: los mensajes entre dos entidades Q.931 referentes a una misma llamada tienen todos el mismo CRV(*call reference value*), el cual es distinto al utilizado en los mensajes con otra entidad. Esto determina que los mensajes que se reciben de un terminal deban ser mandados al otro con un CRV diferente. Entonces, los mensajes recibidos por el *gatekeeper* desde un terminal H.323 también sufren modificaciones al ser enviados al terminal destino, siempre respetando las recomendaciones H.323 y H.225.

También se podrían enumerar toda una serie de procedimientos que se deben efectuar al recibir ciertos mensajes, que no se limitan, simplemente, al reenvío de mensajes al terminal destino. Ver en sección 4.6.7.2 los métodos OnReceivedSignalSetup y OnReceivedCallProceeding en la clase GKConnection.

4.4.2.2. Cambios en el archivo de configuración

Se realizaron las siguientes modificaciones en el archivo de configuración:

Sección General

En esta sección se agregaron los siguientes campos:

- ruteadoPorGK : valor booleano que determina si la señalización de llamada Q.931 es ruteada por el *gatekeeper* o no
- puertoQ931 : valor del puerto donde el *gatekeeper* recibirá la señalización Q.931.

Sección Zona

Ahora, el campo "modo" sólo puede tener dos valores: "0", "soloAlias", o "1", "aliasEIP". De esta manera se racionaliza la configuración del *gatekeeper* descartando el modo "soloIP" que no reflejaba un modo de funcionamiento acorde a los servicios que brinda el *gatekeeper*. O sea, no tenía sentido restringir la registración de terminales basándose sólo en la dirección IP de éstos, cuando el *gatekeeper* brinda clases de servicios basadas solamente en alias.

Se eliminaron de esta sección los campos: "chequeo_cantidad_de_dígitos", "cantidad_de_digitos", "caracteristica", "chequeo_caracteristica". De esta forma, todos los usuarios que conforman la zona están definidos en las secciones de usuario, sin que esto disminuya, o afecte, las distintas posibilidades de configuración que antes se lograban, logrando, a su vez, un archivo de configuración homogéneo y entendible.

Sección Usuario

En las secciones de usuario se sigue manteniendo la posibilidad de definir rangos de números que se utilizan como alias para los terminales. Se podrían asociar estos rangos de números a la "característica" antes utilizada. La diferencia respecto a versiones anteriores radica en la forma como se definen estos rangos. Cada sección de usuario puede reflejar una característica al definirse ésta en el campo "alias". La cantidad de dígitos de cada característica puede ser variable y está dada por la cantidad caracteres que conforman el "alias". De esta manera, la parte variable de cada rango de números definidos está determinada por un caracter punto (".") que indica, como una expresión regular, un dígito cualquiera. A modo de ejemplo: el "alias" "23.." abarca los números que van desde el 2300 al 2399.

Sección Clase

La nueva forma de notación se generaliza a todo el archivo de configuración. Por lo tanto, en las secciones que definen las clases de servicio, dentro del campo "lista_de_restricciones", también se utiliza el caracter "." para indicar los rangos de números restrictos.

4.4.2.3. Cambios en la inicialización e interfaz de comandos

Además de los parámetros que ya se podían pasar al inicio del programa, se agregaron dos más para setear las nuevas funciones del *gatekeeper*.

Se puede determinar mediante un argumento si el *gatekeeper* rutea la señalización de llamada y, en ese caso, se puede indicar el puerto en el que se escuchará la señalización. Por defecto se utiliza el puerto estándar "1720".

A modo de ejemplo, el siguiente comando invoca el *gatekeeper* de forma que rutee la señalización de llamada utilizando el puerto 8989, tomando como archivo de configuración "config.ini" para el resto de la opciones:

>Gkmod3 -r -p 8989 -f config.ini

También se realizaron algunas modificaciones en la interfaz de comandos. Ahora es posible cambiar el despliegue de la traza hacia la consola, con la opción "-c" del comando "trz". Para volver a desplegarla en un archivo se utiliza el mismo comando con la opción "-a".

4.5. Etapa cuatro

4.5.1. Planteo de los objetivos

El objetivo original presentado para esta etapa de desarrollo fue la adición del soporte de servicios suplementarios para el *gatekeeper*, servicios previstos por la recomendación H.450. Se pensaba implementar aquellas funcionalidades necesarias en el *gatekeeper* para tener el servicio de transferencia de llamada (de acuerdo con H.450-2 [CTSS]) y el servicio desvío de llamada (de acuerdo con H.450-3 [CDSS]), o que el *gatekeeper* cumpliera con una función de proxy de señalización para aquellos terminales que no soportaran dichos servicios a través de H.450.

También fue especificado en los objetivos originales de esta etapa, que la implementación de los mismos estaría condicionada por los tiempos necesarios para llevar a cabo las etapas anteriores.

4.5.1.1. Dificultades encontradas

El primer problema fue la imposibilidad de cumplir con la primera previsión de estudiar e implementar las funcionalidades básicas de H.450 en la tercera etapa de desarrollo. Esto implicó postergar todo el análisis de los requerimientos de H.450 para la etapa cuatro, la cual ya se encontraba muy limitada en tiempo.

Luego, un primer análisis de los requerimientos necesarios para implementar H.450, en particular el servicio de transferencia de llamada, mostró

la necesidad del manejo de H.245 por parte del *gatekeeper*. Es decir, el *gatekeeper* debía encargarse de encaminar H.245 al igual que lo estaba realizando con H.225-Q.931. Esto era una dificultad realmente mayor dado el tiempo extra necesario para sumar esta nueva funcionalidad, por lo que quedó prácticamente descartado el servicio de transferencia de llamada.

Por último, pero no menos importante, se plantearon dificultades a la hora de encontrar terminales que implementaran servicios suplementarios utilizando H.450. El equipo Cisco 827, que se estaba utilizando para las pruebas, requería una actualización de IOS y otras features pagas. El equipo BCM de Nortel, que también se utilizó durante algunas pruebas, posee una implementación propietaria de servicios suplementarios como desvío y transferencia de llamada. Mientras que el terminal OpenPhone que posee el proyecto OpenH323 utiliza H.450 pero únicamente para la transferencia de llamada.

4.5.1.2. Replanteo de los objetivos

Por todo lo expuesto en el punto anterior se debió realizar un nuevo planteo de los objetivos originales, tratando de bajar los requerimientos necesarios para implementar los servicios requeridos.

La primera decisión fue quitar el servicio transferencia de llamadas como objetivo, debido principalmente al problema ya expuesto con la señalización H.245.

Como segundo punto se decidió implementar el servicio de desvío de llamada, pero utilizando como herramienta fundamental el manejo por parte del *gatekeeper* de la señalización de llamada H.225/Q.931.

Esta posibilidad si bien, no está basada en la recomendación H.450, no deja de cumplir con la norma H.323, ya que en esta última se encuentra la posibilidad de que el *gatekeeper* desvíe la señalización o la modifique en caso de ser necesario para la implementación de servicios, o funcionalidades extras.

Además, también está previsto la utilización del mensaje *Facility* de H.225 para la implementación de los mencionados servicios. De hecho el terminal OpenPhone implementa desvío y transferencia de llamada mediante este mensaje.

Por lo tanto, se reformuló esta etapa para que nuestro *gatekeeper* permitiera;

- Implementar los tres tipos de desvío de llamadas[1] ;
  - Desvío de llamada incondicional de un terminal hacia un tercero. (SSDI - Servicio suplementario desvío incondicional)
  - Desvío de llamada en caso de terminal destino ocupado.(SSDO - Servicio suplementario desvío ocupado)
  - Servicio de Desvío de llamada en caso de que el terminal destino no conteste la llamada, luego de vencido un tiempo dado. (SSDNC - Servicio suplementario desvío no contesta)

[1]Habitualmente, cuando se habla de desvío de llamada, se está considerando estos tres tipos. Incluso la recomendación H.450-3 lo hace de esta forma.

- Configurar a cada uno de los usuarios o grupos definidos en la zona del *gatekeeper* las tres posibilidades de desvío de llamada.

4.5.2. Las características obtenidas del módulo 4

Los tres tipos de desvío de llamada fueron concretados con éxito, así como la posibilidad de configurar este servicio suplementario a los usuarios de la zona administrada por nuestro *gatekeeper*.

Lo que implicó que el *gatekeeper* ya no solo debía actuar como pasarela de los mensajes H.225/Q.931 sino como terminador e iniciador de estos mensajes. Pues, con cada llamada iniciada, éste debía; chequear si se tenía configurado un servicio de desvío, en caso afirmativo debía proceder a reiniciar una llamada cuando correspondiera el desvío. O sea, terminal A (Ver figura 4.1), llama al terminal B el cual tiene configurado en el *gatekeeper* desvío en caso de no contesta. Luego de vencido del tiempo configurado para el desvío hacia terminal C, la llamada es iniciada hacia este último y finalizada hacia el terminal B.

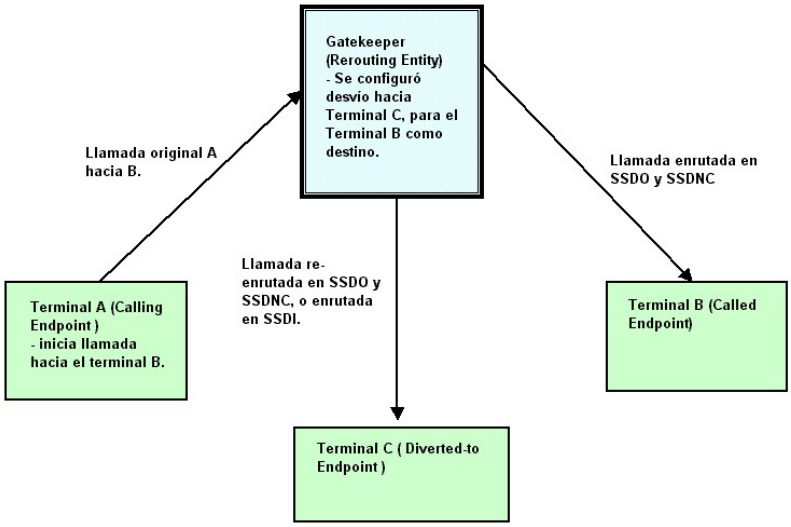

Figura 4.1: Desvío de llamada.

4.5.2.1. Servicios de desvío configurables en gatekeeper

Como primer paso se procedió a modificar el archivo de configuración y su posterior carga en el programa para albergar estas nuevas posibilidades. Para lo cual se agregaron cuatro nuevos parámetros a la configuración de usuario o grupo de usuarios, los cuales se detallan en el cuadro 4.3.

Los primeros 4 parámetros ya fueron explicados en la sección 4.3.2.1.

El quinto contiene el alias del usuario al cual se pretende desviar en caso de que el terminal del usuario configurado declare estar ocupado.

```
[usuario3]
nombre=José Pérez
alias=jperez,jp@gkpro.net,7301
direccionIP=192.168.100.54
clase=clase2
forwardBusy=3402
forwardUnconditional=no
forwardNoReply=3402
NoReplyTimer=10
```

Cuadro 4.3: Nuevos parámetros de usuario

El parámetro forwardUnconditional, define justamente, el alias al cual se desvía en forma incondicional, cuando un terminal intenta comunicarse con el usuario configurado en esta sección.

Los últimos dos parámetros indican alias de usuario al cual desviar y el tiempo de espera por una respuesta (mensaje *connect*), para el usuario configurado en esta sección.

4.5.2.2. Desvíos de llamada implementados por gatekeeper

Para lograr las tres variaciones de desvío mencionadas mas arriba, se utilizó únicamente la señalización H.225/Q.931, como ya se dijo.

Se agregó la inteligencia necesaria en el programa para analizar todos los mensajes *setup* que llegan al *gatekeeper*, determinar si es aplicable alguno de los desvíos, y generar los mensajes necesarios para realizar cada desvío. Todo esto sin la intervención de los usuarios durante la llamada. El *gatekeeper* opera de acuerdo a la configuración previamente guardada en su archivo de inicio.

En el desvío incondicional, cuando el mensaje *setup* llega al *gatekeeper* y es analizado simplemente se encamina la llamada hacia el Terminal C, en lugar del destino original, Terminal B.

Para el caso de desvío ocupado, el mensaje *setup* es encaminado hacia el Terminal B primero y se permite que éste determine si está ocupado (mediante un mensaje *ReleaseComplete*, causa *UserBusy*). Luego se termina la llamada con el Terminal B y se procede a iniciar una nueva llamada con el Terminal C.

Por último, en el caso desvío no contesta, cuando se recibe el mensaje *alerting* desde el Terminal B se dispara un timer con el tiempo configurado para este desvío en el archivo de inicio. Luego de expirado este tiempo, se inicia la terminación de la llamada original hacia el Terminal B, y luego se inicia la nueva llamada hacia el Terminal C. En caso de que el mensaje *connect* llegue desde el Terminal B, antes del vencimiento del timer, la llamada se completa con el terminal destino original, el B.

4.5.2.3. Manejo de señalización de desvío para OpenPhone

También se habilitó la posibilidad del *gatekeeper* de interpretar el mensaje *Facility* enviado por los terminales OpenPhone para realizar un desvío. En esta caso es el programa que interpreta el desvío anunciado por el mensaje *Facility* y lo realiza sin la intervención del terminal originante.

4.6. Arquitectura del programa

En esta sección se detalla la arquitectura del código resultante implementado. Algunas de las opciones de diseño se basan en ciertas premisas que se adoptaron al inicio del proyecto. Por ejemplo, se convino en utilizar la biblioteca sin modificarla sobreescribiendo procedimientos y clases para lograr las funciones requeridas. Otras opciones de diseño se explican por la evolución temporal del proyecto y por ciertas condicionantes impuestas por la biblioteca utilizada.

En la figura 4.2 se muestran las clases de software más importantes y sus relaciones. Se tomará dicho diagrama como punto de partida para la explicación de la implementación del programa.

El programa se basa en la clase GKPro la cual representa la entidad *gatekeeper*. Dicha clase contiene ciertos atributos donde se referencian los datos de los usuarios de la zona, de las redes desde las cuales se pueden registrar, y de las clases de servicio que el *gatekeeper* brinda a sus usuarios.

El atributo listaDeRedes es una instancia de la clase PList y constituye una lista de las clases RedIP. En dicha lista se agrupan las instancias de la clase RedIP, cada una de las cuales representa una de las redes habilitadas, desde las cuales se pueden registrar los usuarios.

ListaDeUsuarios es una hashtable (diccionario) que almacena las instancias de la clase Usuario referenciadas por una key del tipo PString el cual es el alias del usuario. De esta forma, en base al alias por el que se identifica el usuario se puede referenciar a la instancia que lo representa y a su información. Se crea una instancia de la clase Usuario por Usuario definido en el archivo de configuración (usuario individual o grupo de usuarios definidos por un prefijo), representando cada instancia los usuarios habilitados a registrarse y no los registrados en sí. Cuando se pide algún servicio al *gatekeeper* (registración, admisión de llamadas, etc), éste busca en estas estructuras, en base a las cuales determina si brinda los servicios. La información respectiva al usuario registrado se almacena en otras estructuras brindadas por la biblioteca que se diferencian de las antedichas.

ListaDeClases es una variable del tipo PList que contiene instancias de la clase ClaseDeServicio. Cada instancia representa un tipo de servicio que brinda el *gatekeeper*. Cada usuario, o mejor dicho, la representación en *software* del usuario, tiene una referencia a la instancia de la clase de ClaseDeServicio que representa los servicios que tiene habilitado.

Una instancia de GKPro contiene a su vez una instancia de GKEndPoint que representa de manera más general un *endpoint* de H. 323. Estas instancias contienen referencias a las instancias de la clase GKConnection, cada una de las cuales representa una conexión con otra entidad. En el caso del *gatekeeper*, esta parte del modelo tiene gran importancia para la

implementación de la señalización de llamada por parte del *gatekeeper*. Por medio de estas clases, el *gatekeeper*, otro *endpoint* a nivel de señalización, establece conexiones con los terminales representándose estas en *software* vía la clase GKEndPoint. Dado que cada llamada cuya señalización pasa por el *gatekeeper* implica una conexión por cada terminal, cada instancia de GKEndPoint contiene una referencia a la instancia par que representa la conexión con el otro terminal.

El modelo destaca las clases más importantes desarrolladas en el proyecto, su relacionamiento, y algunas dependencias con las clases de la biblioteca. Evidentemente existen muchas clases de la biblioteca involucradas que por razones de simplicidad y para que se logre un entendimiento primario no se incluyen.

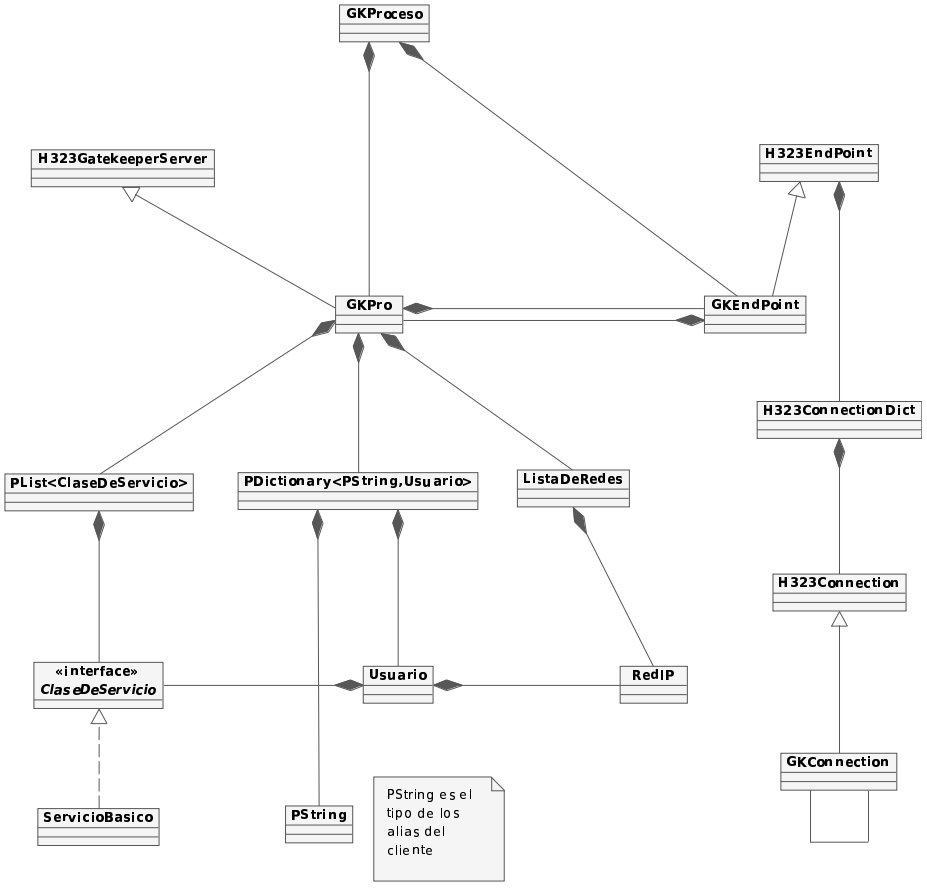

Figura 4.2: Diagrama de clases

4.6.1. Proceso e inicialización del programa

Como punto de partida para la aplicación *gatekeeper*, se tomó la clase PProcess. Heredamos de esta clase [2] y luego sobrescribimos el método Main(), el cual será llamado al ejecutar el programa. Por lo tanto, dentro del mencionado método lanzamos todas las tareas de inicialización de nuestro *gatekeeper,* para finalmente dejar habilitada la interfaz de comandos (ver sección A.1.3).

Las tareas de inicialización son las siguientes;

1. Obtención de los argumentos pasados al programa y almacenamiento para posterior consulta.
2. Carga de la información disponible en el archivo de configuración.
3. Inicialización del archivo de traza.
4. Inicialización de la instancia GKEndPoint.
5. Inicialización de la instancia GKPro.

En lo que sigue se da un detalle de cada tarea de inicialización.

4.6.1.1. Argumentos de entrada

Para obtener las opciones y parámetros pasados por el usuario al iniciar el programa, se utiliza el método PProcess::GetArguments(), el cual devuelve una lista de argumentos en un objeto de tipo PArgList. Luego se llama al método PArgList::Parse() pasándole como argumento la forma en que se desea interpretar las opciones y parámetros recién cargados, éste interpreta la línea de argumentos del programa y la decodifica.

Luego, las opciones y parámetros pasados al *gatekeeper*, están disponibles en el objeto PArgList para ser consultados con los métodos PArgList::HasOption() y PArgList::GetOptionString().

4.6.1.2. Carga de archivo de configuración

El archivo de configuración es imprescindible, por ello, sino se encuentra disponible en el directorio del ejecutable o no es indicado en los argumentos iniciales del programa, éste se detiene.

Entonces, cuando está disponible, se inicializa un objeto de tipo PConfig con la ruta completa o relativa hasta el archivo de configuración. Luego, se podrá leer la información guardada en este archivo mediante los métodos; PConfig::HasKey() y PConfig::GetString().

4.6.1.3. Inicialización de archivo de traza

El programa utiliza un archivo de traza para volcar los diferentes eventos que ocurran durante la ejecución, para la depuración de problemas del código o de las comunicaciones del *gatekeeper* con los terminales H.323.

[2] La clase PProcess nos ofrece una interfaz sencilla con la consola del sistema operativo, ver sección 3.10.

Para crear dicho archivo e inicializar el nivel de traza que éste tendrá, se utilizan los argumentos pasados al programa o el archivo de configuración, siendo este último el de menor prioridad.

El nivel de traza es usado para indicar que tan profundo se desean obtener datos de los eventos ocurridos durante la ejecución del programa. Con un nivel bajo, ´1´ o ´2´, se obtendrá mayormente el ingreso y salida de algunos métodos; mientras con un nivel ´5´o mayor, se podrá obtener hasta una copia de los mensajes recibidos por el *gatekeeper*.

Existe la opción de pasar esta traza a la consola del *gatekeeper*, ya sea desde el inicio del programa después de haber ingresado a la misma. Sin embargo, se ha comprobado que utilizar una traza en nivel alto por consola consume demasiados recursos.

4.6.1.4. Inicialización de GKEndPoint

La siguiente tarea es la creación de un objeto de tipo GKEndPoint y la posterior inicialización, llamando al método GKEndPoint::InicializoEP(), ver figura 4.3. Como parámetros de este último van la lista de argumentos y el archivo de configuración.

Las diferentes tareas que cumple este objeto se irán detallando más adelante, pero la que compete en este caso es el mantenimiento de un canal de señalización de llamada H.225-Q.931. Dicho canal es iniciado por el método mencionado, si hay encaminamiento de señalización por *gatekeeper*. De los argumentos y archivo de configuración se obtiene el puerto y dirección IP donde crear el puerto. Para esto se utiliza el método H323EndPoint::StartListener() (ver sección 3.2 y 3.3.1.2) ya que, como lo muestra el diagrama de clases, GKEndPoint hereda de H323EndPoint.

Como antes, son prioritarios los parámetros de configuración pasados como argumentos al inicio del programa frente a los contendidos en el archivo de configuración, y si no se dispone de ninguno de estos datos se utiliza el puerto por defecto (1720) y la loopback.

Si no hay problemas la inicialización continúa, de lo contrario un mensaje de error es desplegado y se termina la ejecución.

4.6.1.5. Inicialización de GKPro

La clase GKPro hereda de H323GatekeeperServer y es la siguiente en ser creada e inicializada (método GKPro::InicializoGK()), nuevamente a través de la lista de argumentos y el archivo de configuración.

La inicialización de GKPro involucra varias tareas;

1. Inicio de canal RAS.
2. Inicialización de la zona atendida por el *gatekeeper*.
3. Inicialización de las clases de servicio.
4. Inicialización de los usuarios a ser atendidos por el *gatekeeper*.

La tarea 1. implica la creación de un H323GatekeeperListener y la posterior adjudicación de éste al GKPro mediante el método AddListener() (ver sección 3.3.1.2). Para el puerto y dirección IP se utiliza un criterio idéntico

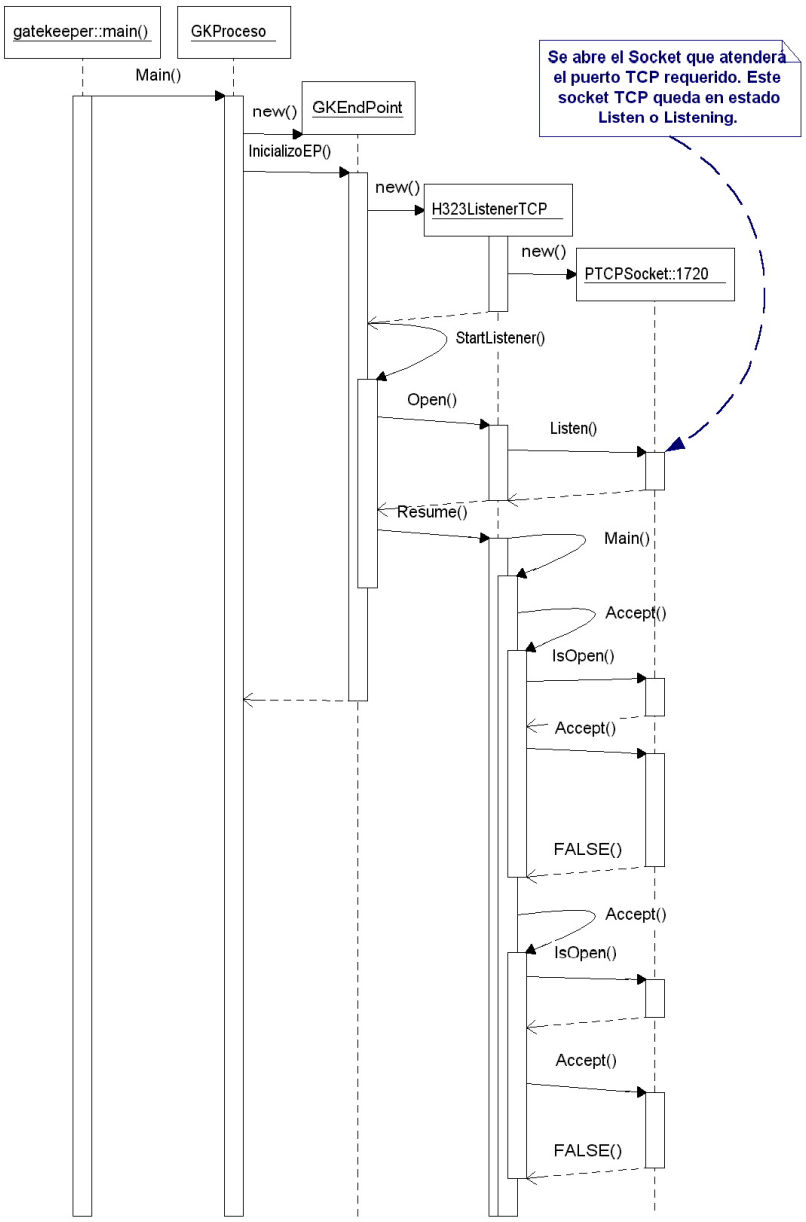

Figura 4.3: Inicio canal de señalización de llamada.

al empleado con el GKEndPoint, prioridad uno a los argumentos y luego al archivo de configuración, por defecto se utiliza el puerto 1719 y la loopback.

También es cargada la opción de encaminamiento de *gatekeeper*, a verdadero o falso, según el valor encontrado en el archivo de configuración o la

lista de argumentos iniciales.

La segunda tarea toma datos exclusivamente del archivo de configuración; éstos son el modo de funcionamiento y las diferentes zonas que atenderá el *gatekeeper*, las cuales son guardadas en un contenedor de tipo PList<RedIP>.

La tercera tarea, implica tomar del archivo de configuración cada una de las clases de servicio, generar una clase ServiciosBasicos que contendrá los datos de cada clase de servicio definida en él y, guardarlos en un contenedor de tipo PList<ServiciosBasicos>. Esta etapa en realidad es procesada completamente por el método GKPro::InstanciarClases().

La última tarea involucra la carga de la configuración de cada usuario en objetos de tipo Usuario, para su posterior almacenamiento en un container de tipo PDictionary<PString,Usuario>. Como su nombre lo indica, éste es un diccionario que puede ser accedido usando como clave de índice algunos de los alias de usuario especificado en el archivo de configuración. Conviene aclarar que se crea un único objeto Usuario por cada ítem de usuario en dicho archivo y luego se crean tantos punteros en el diccionario, como alias tenga este. De forma similar si el alias es en realidad una expresión regular engloba una lista de números, las claves de entrada para el diccionario son justamente estos números. Por ejemplo; si se utilizó las expresión regular "235.", para un grupo de terminales, se crea un único Usuario para contener su configuración y el diccionario puede ser accedido por cualquiera de los alias de usuario desde "2350" hasta el "2359", todas estas claves apuntan a la misma información.

Otra aclaración sobre punteros es que la clase Usuario posee uno a la clase ServiciosBasicos que le corresponda al usuario por su configuración. O sea, la configuración de un tipo de servicio en ServiciosBasicos estará asignada a tantos usuarios como se haya indicado en el archivo de configuración.

4.6.2. Interfaz de comandos

Como ya se dijo, al finalizar todo el proceso de inicialización del *gatekeeper*, el método GKProceso::Main() deja el programa en una consola de usuario, la cual permite hacer algunas tareas muy sencillas, ver sección 4.3.2.2 .

Esta consola se implementa sencillamente con: un "loop infinito", la lectura continua de las líneas escritas por el usuario, una interpretación de estas líneas en comando y argumentos en forma similar a lo hecho con la lista de argumentos inicial del programa y, la llamada a los métodos que implique cada comando.

Para la recarga del archivo de configuración se llama al método GKPro::InicializoGK() nuevamente, pero en realidad no se ejecuta todo el proceso mencionado en la sección 4.6.1.5, se excluye de la recarga la tarea 1 (inicialización de canal RAS).

En el caso de ejecutar el comando que muestra los terminales activos se llama al método GKPro::TerminalesRegistrados() y para obtener las llamadas en curso GKPro::LlamadasEnCurso(), ambos métodos consultan las estructuras internas del H323GatekeeperServer para obtener la información requerida y se le da un formato de salida adecuado.

Finalmente el comando de salida, realiza un *break* en el "loop infinito" y a partir de ese punto se realizan las tareas de salida de la aplicación *gatekeeper*; borrado de información de usuario, clase de servicios, bajada de *listener*s, etc.

La realización de una sencilla consola fue la opción natural para permitir una interfaz con el programa. Existieron otras opciones más sofisticadas que se consideraron; como una interfaz GUI, e incluso una interfaz HTTP, esta última implementada parcialmente. Pero dado que no era el objetivo principal del proyecto se terminó por utilizar esta primera opción.

4.6.3. Usuarios

Los usuarios que pueden utilizar el *gatekeeper* se definen en el archivo de configuración como ya se explicó anteriormente. Al iniciarse el *gatekeeper*, se leen los datos de los usuarios y se cargan en estructuras con las que se trabaja durante la ejecución del programa. Cualquier modificación en el archivo de configuración: agregar o borrar nuevos usuarios, cambios en sus características; puede ser incorporada al funcionamiento del *gatekeeper* mediante el comando "rcg" de la interfaz de comandos, el cual borra las estrcturas internas y las arma nuevamente con los nuevos datos.

La clase Usuario tiene distintos atributos que determinan las características y los servicios que dispone cada usuario en particular. Existen atributos informativos que determinan el usuario: listaDeAlias, nombre y dirección IP. Si bien listaDeAlias se utiliza durante la inicialización y en la recarga del archivo de configuración, los dos últimos no son utilizados en el funcionamiento del *gatekeeper*. Se los incluye para disponer de la información de manera estructurada y para su posible utilización en modificaciones posteriores. Otros atributos son fundamentales durante la ejecución del programa, a saber: un puntero a la instancia que representa la clase de servicio a la cual pertenece el usuario y la información sobre las características de los servicios suplementarios de que dispone. Durante el diseño se barajaron distintas posibilidades sobre la forma de implementar los servicios suplementarios. Dado que existe cierta información que es única para cada usuario se decidió incluirla dentro de esta clase. En el caso de que el usuario disponga de algún tipo de transferencia, qué tipo de transferencia, el número al que se debe transferir y el tiempo que se espera en el caso de fuera una transferencia por no contestar, se almacena en variables de la clase usuario que se consultan durante la ejecución del programa.

El atributo listaDeAlias es un PStringArray que incluye todos los strings que se le configuran al usuario en el archivo "config.ini". Dichos alias son incluidos durante la inicialización del *gatekeeper* en PDictionary-<PString,Usuario>listaDeUsuarios, asociándose cada alias con la instancia de la clase Usuario que determina al cliente. Esta solución implica que cada alias que se le configura al cliente en el archivo de configuración pasa a ser un índice en el diccionario asociado con la clase del usuario; si el cliente tiene muchos alias, bastará que al registrarse con el *gatekeeper* use alguno de ellos para que al chequear la variable listaDeUsuarios se le permita su registración y los servicios posteriores. O sea, una vez inicializada la variable listaDeUsuarios, ésta es la variable que se consulta durante la ejecución del programa para saber los usuarios del *gatekeeper*.

Las funciones de la clase Usuario son para el acceso a los atributos. Como se ve esta clase básicamente encapsula la información de usuario siendo indispensable durante la ejecución del programa para la implementación de los servicios suplementarios.

4.6.4. Zonas

El *gatekeeper* define las zonas a las que brinda servicio en base a dos pautas: los alias de los usuarios y las direcciones IP de éstos. El alias del usuario es fundamental, y el *gatekeeper* sólo brindará servicio a los usuarios que se registren con al menos un alias de los que tiene configurado en el archivo de configuración. Este modo de funcionamiento se define como soloAlias. En el otro modo de funcionamiento, aliasEIP, no sólo se tiene en cuenta la información de alias del usuario, sino que también, se chequea la dirección IP que el terminal anuncia como su dirección de señalización comparándola con ciertas redes que se habilitan en el *gatekeeper*. Si alguna de estas condiciones no se cumpliera el *gatekeeper* rechazará la registración.

Estos modos de funcionamiento y las zonas que atiende el *gatekeeper* son tenidas en cuenta durante el pedido de registración al *gatekeeper*. Una vez registrado un cliente, e incorporado éste a las estructuras internas, no se consulta su pertenencia a la zona para peticiones posteriores. De todas maneras, la verificación está implícita ya que si está registrado pertenece a la zona y si su alias o dirección de señalización se modificara tendría que registrarse nuevamente.

Durante el funcionamiento del programa la forma en que se implementa dicho comportamiento es la siguiente. La clase GKPro contiene dos variables que determinan la zona que atiende el *gatekeeper*: listaDeUsuarios y listaDeRedes. Como ya se ha explicado, listaDeUsuarios asocia los alias que se pueden registrar con las instancias de las clases Usuario; listaDeRedes es una lista de las clases RedIP (PList(RedIP)). RedIP es una clase que se creó para encapsular la información de las redes que pertenecen a la zona del *gatekeeper*. Contiene dos atributos que determinan la dirección de red y la máscara de cada red en particular (RedIP::direcccionRed y RedIP::mascaraRed). Esto permite que se pueda definir las zonas con la granularidad que se quiera pudiéndose llegar a definir zonas de hasta una sola IP utilizando una máscara 255.255.255.255.

El método fundamental que dispone esta clase es ChequeoIP que tomando como argumento una dirección IP determina si ésta pertenece a la red definida por la instancia. El chequeo se realiza de la forma habitual haciendo un AND bit a bit entre el argumento y la máscara para ver si coincide con la dirección de red.

Ya se explicó cómo se inicializa la estructura listaDeUsuarios, en cuanto a listaDeRedes, se define durante la inicialización del *gatekeeper* en la función InicializaGK() y se crea una lista de clases RedIP con una cantidad igual a las definidas en el archivo de configuración.

Ante un pedido de registración mediante un paquete RRQ que llegue al *gatekeeper*, se llama a la función OnRegistration() de la clase H323GatekeeperServer. Dicha función se sobrescribió en nuestra implementación en la clase GKPro para realizar los chequeos de zona correspondiente. En ella se revisa la lista de alias que presenta el terminal llamándose a la función

AliasHabilitado() de GKPro para compararla contra la lista de usuarios del *gatekeeper*. En el caso de que se esté en el modo de funcionamiento "alias-EIP" se llama a la función IPHabilitada() de GKPro que, a su vez, llama ChequeoIP() para cada una de las instancias de listaDeRedes. Si el cliente supera dichas pruebas se lo registra en el *gatekeeper* (ver figura 4.4).

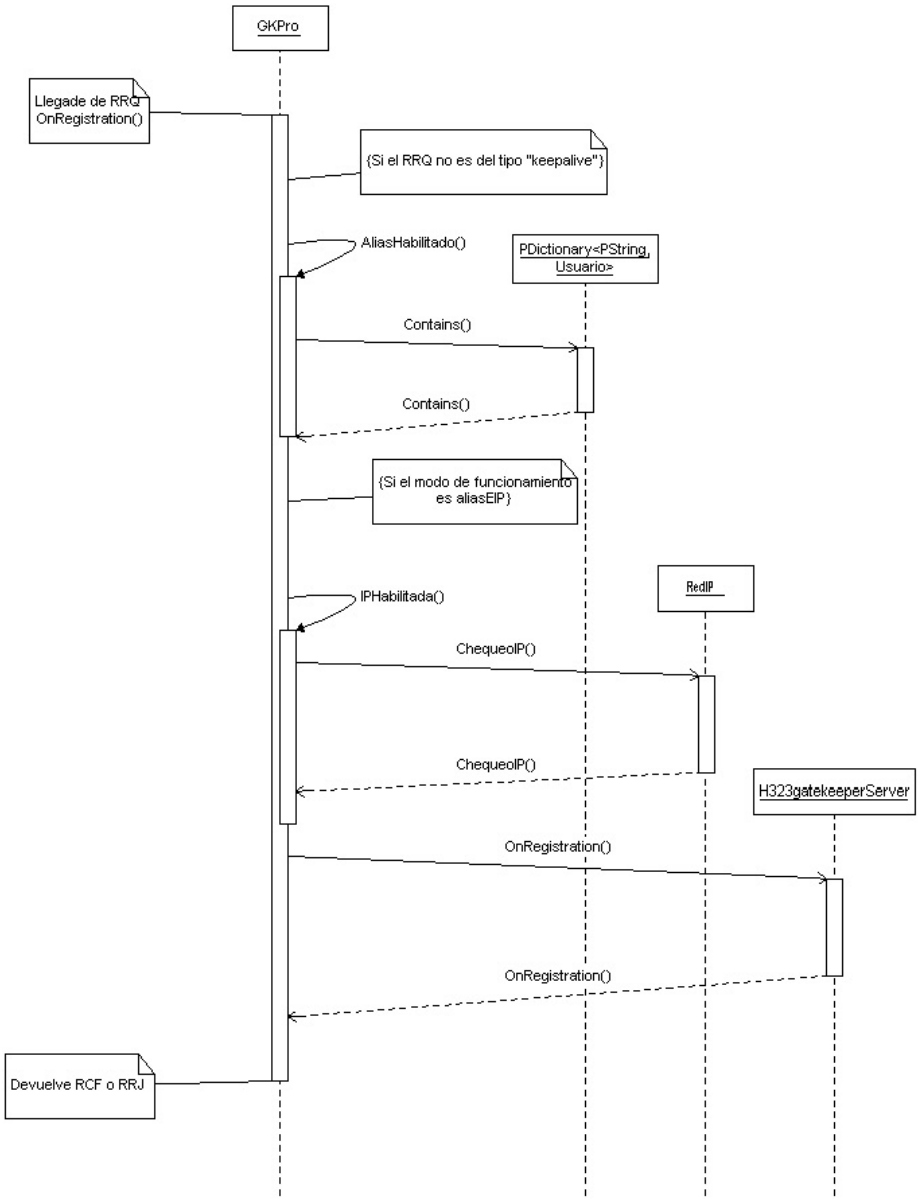

Figura 4.4: Registro de terminal

4.6.5. Clases de servicios

Cada usuario del *gatekeeper* tiene asociado un tipo de servicio que determina las funcionalidades que se le ofrecen. En el archivo de configuración se definen distintos tipos o clases de servicio, cada una identificada por un número, y luego se asocia cada usuario, o grupo de usuarios con una clase determinada. Para la implementación de tal comportamiento se creó una clase de *software* llamada claseDeServicio que representa el tipo de servicio brindado al cliente.

En un principio se pensó en definir dicha clase de *software* dejando la posibilidad de que a futuro se pudieran agregar otros tipos de servicio representados por otras clases, pero que mantuvieran la interfaz de esta clase original. Por esto se pensó en definir dicha clase como una interfaz de la que heredaran las distintas clases que implementaban los tipos de servicio concretos. La idea era representar los servicios suplementarios con una clase diferente que heredara de la clase original. Luego se desechó esta idea porque se constató que la implementación era más fácil y sencilla a través de datos incluidos en la clase Usuario. Sin embargo, se mantuvo la idea original, dejándose la posibilidad de que en el futuro se pueden agregar nuevos tipos de servicio mediante este método. A modo de ejemplo: se podría implementar un tipo de servicio que controlara a los usuarios según la hora del día en que intentan hacer una llamada, de modo de restringir las llamadas en ciertos horarios. Esta solución implica una interfaz dada por la función EstaHabilidado() que debería llamarse para cada clase en especial y que tuviera un comportamiento distinto según el tipo de servicio y no habría que modificar ningún código mientras que se utilice dicha función como una forma de habilitación para hacer llamadas.

La clase de *software* ServicioBasico caracteriza un tipo de servicio brindado por el *gatekeeper* en el que se puede limitar tanto las llamadas entrantes como salientes que hacen los usuarios. Las variaciones de estos parámetros determinan un tipo de servicio al que el usuario está suscrito. De esta manera puede haber una clase de servicio en la cual los usuarios no puedan hacer llamadas, pero sí recibirlas; otra en la cual, tengan limitado ciertos números para hacer llamadas, etc.

El software desarrollado mantiene una analogía con la realidad planteada. Existe un clase de software llamada ServiciosBasicos que hereda de ClaseDeServicio y que posee atributos de modo que las distintas instancias de esta clase, de acuerdo a los valores de sus atributos, determinan una clase de servicio brindada al cliente. Los atributos son booleanos que determinan si los usuarios pertenecientes a esas clases pueden recibir llamadas (llamadasEntrantes), hacer llamadas (llamadasSalientes), y en este último caso, si tiene algún número restricto al que no pueden llamar (restricciones). También existe otro atributo del tipo PStringList que es una lista de strings para almacenar los alias a los que no puede llamar.

Las instancias de dichas clases que representan las clases de servicio que presta el *gatekeeper* se crean en la función InicializaGK() de GKPro de acuerdo a los valores del archivo de configuración los cuales mantienen un paralelismo con los atributos de la clase. En dicha función se llama a la función InstanciarClases() la cual lee del archivo de configuración la cantidad de clases definidas y crea una instancia de la clase de *software* inicializa-

da con los atributos correspondientes. Dichas instancias se agrupan en el atributo del GKPro listaDeClases que es del tipo PList(ClaseDeServicio). Luego, y también en la inicialización del *gatekeeper* se asocia cada alias de listaDeUsuario con la clase correspondiente de listaDeClases.

Las instancias de estas clases se crean y se mantienen incambiadas durante la ejecución del programa a menos que el administrador decida modificar la configuración de ellas y haga una recarga del archivo de configuración.

La implementación de las políticas de admisión, en nuestro caso la clase de servicio brindada el cliente, se realiza de la siguiente manera. La biblioteca implementa la recepción de mensajes RAS y su clasificación mediante diversos mecanismos anteriormente explicados. Cada vez que se recibe un mensaje se llaman a una serie de funciones (*callback functions*) en clases situadas en niveles más cercanos a la interfaz de usuario. Tal es el caso de la función OnAdmission() implementada en la clase H323GatekeeperServer. Para modificar su comportamiento se sobreescribió dicha función y lo que se hace es determinar, en base a la información que viene en el ARQ, la clase a la que pertenece el cliente. Una vez determinada se llama a la función EstaHabilitado() de dicha clase la que determina la aceptación de la llamada o no. Dado que pueden existir clases de software que hereden de la interfaz ClaseDeServicio() este método asegura que la implementación de la función EstaHabilitado() será distinta según la política de admisión que se implemente. En nuestro caso, y dado que sólo se implementó la clase ServiciosBasicos la función EstaHabilitado() chequea en base a la información que se le pasa como parámetro (el mensaje ARQ) si el mensaje viene de un terminal llamante o llamado. En base a esto y a los valores de los atributos de la instancia, se decide si se acepta el requerimiento. En el caso de que hubieran restricciones se llama a la función ChequeoRestricciones() que compara la solicitud del cliente contra la lista de alias restrictos.

Una vez realizados estos chequeos se llama a la implementación de OnAdmission() de la biblioteca que realiza controles posteriores y registra en tal caso al cliente (ver figura 4.5).

4.6.6. Canal RAS

La biblioteca se encarga de brindar las herramientas para la implementación del canal RAS. El programador cliente se debe encargar básicamente, de sobrescribir ciertas funciones para adaptar el comportamiento del aplicativo a sus necesidades.

La inicialización del canal RAS se realiza, también, durante la función InicializoGK() de la clase GKPro. Como ya se ha explicado anteriormente, se leen los parámetros de configuración del archivo de configuración (a saber: dirección IP y puerto del canal RAS), se crea un objeto del tipo H323TransportUDP y se llama a la función AddListener() de H323GatekeeperServer que se encarga de iniciar el hilo de ejecución necesario para atender el canal RAS.

La tarea fundamental en esta área, y dado que la biblioteca tiene la mayoría de sus funciones implementadas consistía en modificar ciertas funciones. De esta manera se eligió sobreescribir cuatro funciones fundamentales de H323GatekeeperServer: OnDiscovery(), OnRegistration(), OnAdmi-

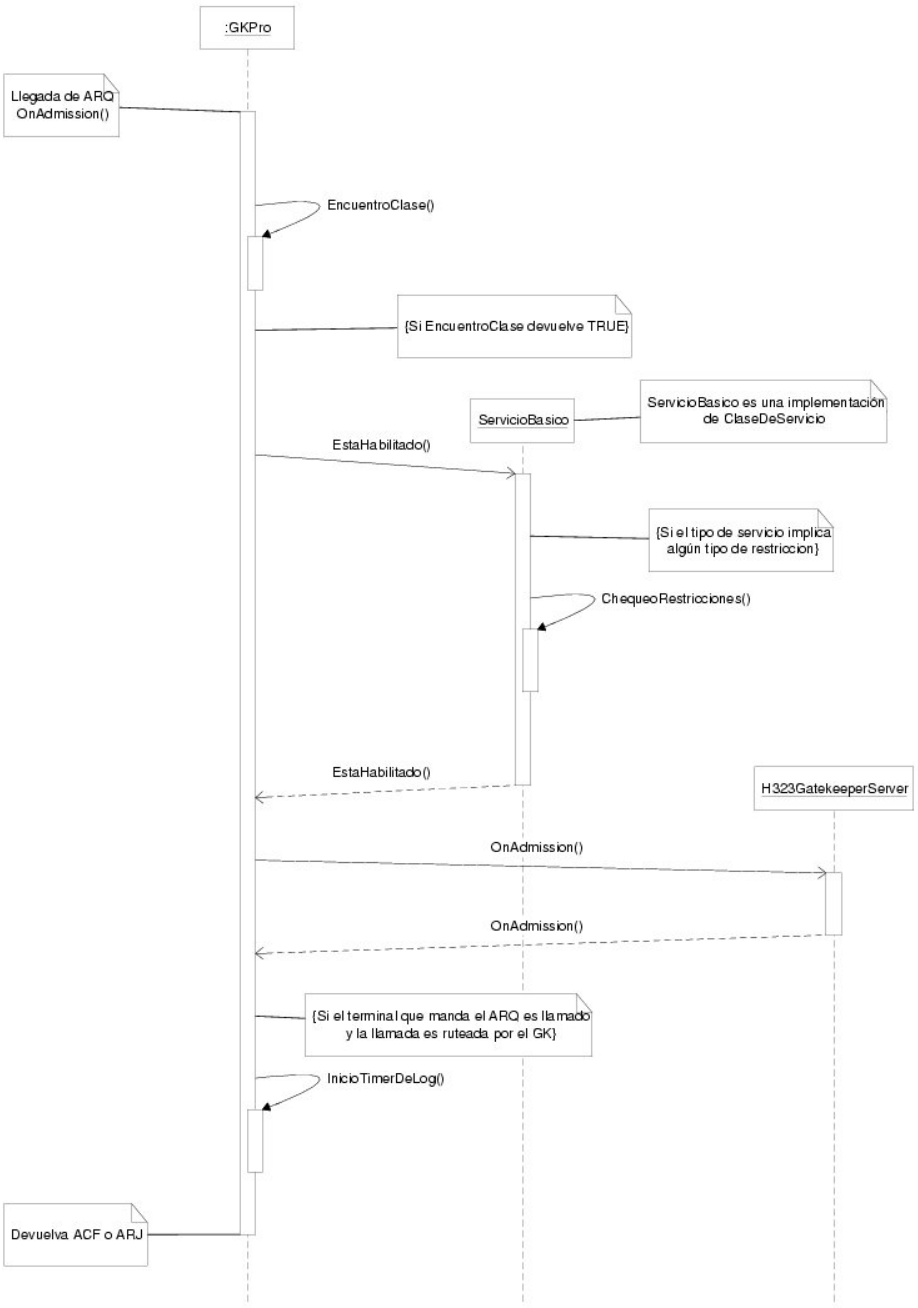

Figura 4.5: Admisión de llamada

ssion(), OnDisengage(). Estas funciones, del tipo *callback*, se llaman cuando se recibe un mensaje, GRQ, RRQ, ARQ y DRQ, respectivamente.

Las acciones que se realizan en cada una de estas funciones ya se ha explicado, basta decir que OnRegistration() se utiliza para implementar las políticas de registración con la consecuente definición de la zona del *gatekeeper*; OnAdmission() implementa las políticas de admisión de llamadas, mientras que OnDisengage() es importante en cuanto al log de llamadas. En el caso de que el *gatekeeper* no curse la señalización de llamadas, la forma de saber, de manera aproximada, los datos para conformar el log, es mediante los mensajes ARQ y DRQ que llegan al *gatekeeper*. Por tanto, en la función OnDisengage(), en tal extremo, se toman los elementos necesarios y se escribe en el log de llamadas mediante el llamado a la función ImprimoLog() de GKPro.

4.6.7. Manejo del canal Q.931

Un estudio detallado de la biblioteca OpenH323, en cuanto al manejo de los mensajes Q.931, permitió determinar las clases que se ocupan de esta tarea, y su funcionamiento. Sin embargo, estas clases están pensadas para implementar la señalización entre terminales H.323, no poseen todas las funcionalidades necesarias para el *gatekeeper*. Por lo tanto, se optó por heredar de las clases apropiadas de la biblioteca y realizar los cambios necesarios para cumplir con los nuevos requerimientos, sobrescribiendo métodos y creando nuevos. Las clases creadas fueron: GKEndPoint y GKConnection, que heredan de H323EndPoint y H323Connection respectivamente.

Básicamente la clase GKEndPoint es capaz de mantener y administrar conexiones Q.931, con los dos[3] terminales que intervienen en una llamada. Una vez recibida una nueva conexión, la clase GKEndPoint crea dos nuevos objetos GKConnection que representan las conexiones que mantiene con los terminales participantes de la llamada. Una GKConnection para el terminal que origina la llamada y otra para el terminal que recibe la comunicación.

4.6.7.1. La clase GKEndPoint

Como ya se mencionó anteriormente la principal función de esta clase es establecer y mantener el canal de señalización de llamada con los terminales H.323. Este canal es establecido con el procedimiento de la sección 4.6.1.4, y depende de la clase H323ListenerTCP.

Esta última clase es la encargada de iniciar y establecer el canal TCP a la llegada de mensajes Q.931, para el cual es creada una instancia de la clase H323TransportTCP. La clase anterior, mediante el método HandleFirstSignallingChannelPDU(), se encarga del análisis y de procesar el primer mensaje Q.931, luego se llama al método OnIncomingConnection de H323EndPoint. A partir de esta última, es creada la GKConnection. El método llamado para realizar esta última operación es H323EndPoint::-CreateConnection() la cual es justamente sobrescrita para modificar su comportamiento por defecto, la creación de una H323Connection.

[3]En caso de desvío se puede dar momentáneamente la existencia de tres conexiones con el *gatekeeper*.

La creación de una clase GKConnection que representa de alguna forma la conexión es - justamente - la segunda función de importancia de la GKEndPoint. Además, posee las funcionalidades necesarias para mantener un lista de conexiones establecidas y, luego a ser terminadas. Esto es, una vez creada la GKConnection se la incorpora al PDictionary H323ConnectionDict, el cual pude ser consultado mediante FindConnectionWithLock(). Luego cuando la conexión es terminada por el terminal o el *gatekeeper*, el objeto no es inmediatamente borrado sino que pasa a una segunda lista , la connectionsToBeCleaned. Esta última es recorrida por una especie de *garbage collector*, la H323ConnectionsCleaner, que inicia el proceso de borrado de las GKConnection en la lista antes mencionada. Todas estas funcionalidades son ofrecidas por defecto por la clase H323EndPoint.

4.6.7.2. La clase GKConnection

Como fue explicado con anterioridad, una nueva GKConnection es creada cada vez que el *gatekeeper* debe establecer un canal de señalización Q.931 con un terminal, justamente durante el encaminamiento de mensajes H.225/Q.931. La principal tarea de esta clase (junto con otros objetos que la componen) es la de mantener este canal y recibir/transmitir mensajes por él. Además, una vez recibido un menaje, este puede ser interpretado y disparar una acción determinada.

Entonces, siguiendo las etapas de un establecimiento basado en H.225/Q.931, el primer mensaje que recibe esta clase es un *setup*. Esto sucede durante su creación desde el terminal que pretende originar una comunicación. Este evento dispara el método OnReceivedSignalSetup() (ver figura 4.6) el cual fue previamente sobrescrito para realizar las siguientes acciones;

1. Determinar hacia que dirección IP y puerto debe establecerse una nueva conexión. Incluye una etapa de análisis de un posible desvío de llamada.

2. Creación de la mencionada conexión mediante el método GKEndPoint::MakeCall(), obteniendo una nueva GKConnection relacionada con el terminal/usuario llamado.

3. Asociación de ambas conexiones utilizando punteros. Por esto son denominadas conexiones pares, origen y destino de una llamada. De esta forma siempre será posible llamar desde una GKConnection, a los métodos de su conexión par. Por ejemplo el WriteSignalPDU() el cual escribe un mensaje Q.931 en el canal TCP asociado.

4. Armado y envío del mensaje *callProceeding* hacia el terminal originante.

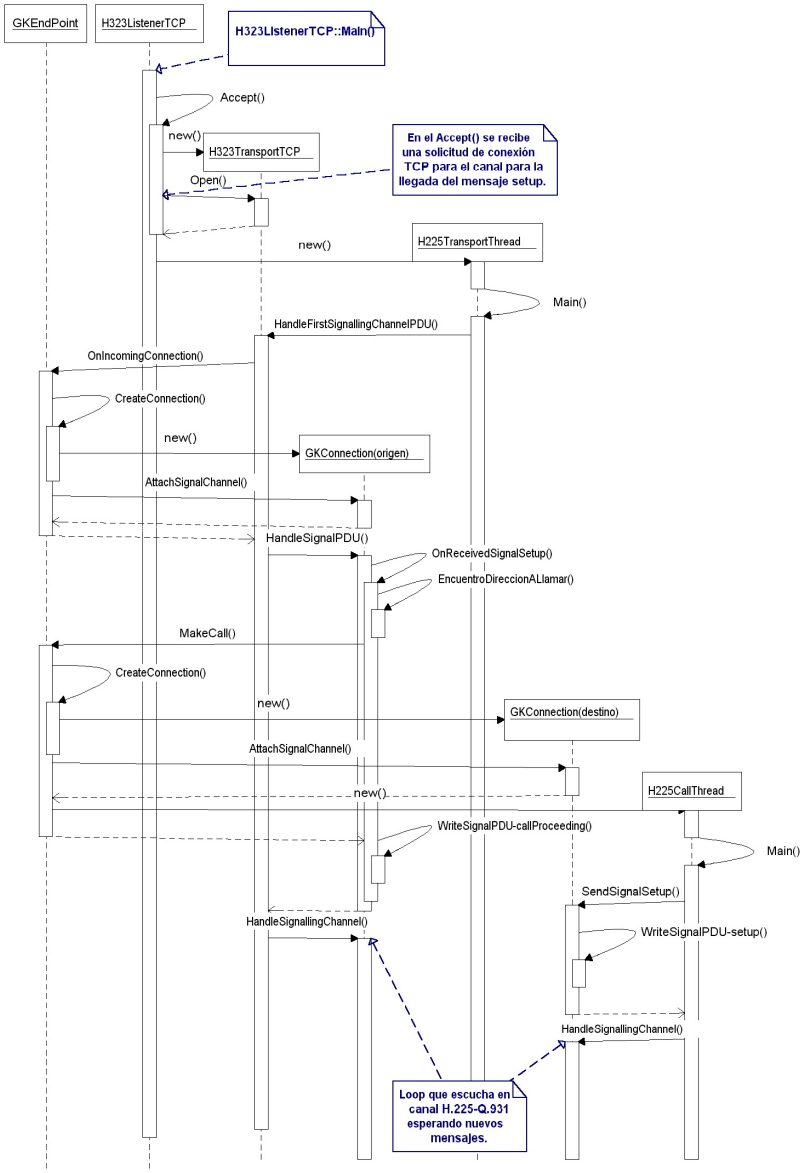

Figura 4.6: Creación de las GKConnection pares

Es fundamental aclarar que el segundo paso implica además, el envío de un nuevo mensaje *setup* hacia el terminal llamado. Esto es realizado en el método GKConnection::SendSignalSetup() el cual, tomando como base el mensaje recibido, genera un nuevo *setup***.** Éste contiene un nuevo campo *Call Reference Value*, generado para este tramo de la conexión, así como una adecuación de los campos *CalledPartyNumberIE* y *RedirectingNumberIE* en caso de desvío.

Luego de esta primera etapa de recepción y transmisión del *setup***,** ambos objetos GKConnection quedan a la espera de nuevos mensaje Q.931, para su interpretación y retransmisión hacia el otro extremo de la conexión.

El cuadro 4.4 muestra qué métodos de la GKConnection son llamados con la llegada de los mencionados mensajes y qué acciones son disparadas.

Mensaje recibido	Método llamado	Acciones tomadas
Setup	OnReceivedSignalSetup	1.Determinación Ip:puerto destino. 2.Llamada a MakeCall(). 3.Asociación conexión Par. 4. Envío del CallProceeding**.**
Setup Acknowledge	OnReceivedSignalSetupAck	1.Se regenera el mensaje. 2. En caso de túnel H.245 se incluye el campo provisionalRespToH245Tunneling. 3. Se reenvía el mensaje.
Call Proceeding	OnReceivedCallProceeding	Se chequea y guarda campos fastStart y h245Control.
Progress	OnReceivedProgress	1.Se regenera el mensaje. 2. En caso de túnel H.245 se incluye el campo provisionalRespToH245Tunneling 3. Se reenvía el mensaje.
Alerting	OnReceivedAlerting	1.En caso de SSDNC se inicia timer. 2.Se regenera el mensaje. 3. En caso de túnel H.245 se incluye el campo provisionalRespToH245Tunneling. 4. Se reenvía el mensaje.
Connect	OnReceivedSignalConnect	1.Se regenera el mensaje. 2. En caso de tener campos fastStart y h245Control se incluyen. 3. Se reenvía el mensaje.
Facility	OnReceivedFacility	1. Si es un callForwarded inicio un desvío con GKEndPoint::OnConnectionForwarded(). 2. Se regenera el mensaje. 3. Se reenvía el mensaje.

Cuadro 4.4: Callback functions en el objeto GKConnection.

Mensaje recibido	Método llamado	Acciones tomadas
Release Complete	OnReceivedReleaseComplete	1. Si causa Busy y hay SSDO inicio desvío. 2. Se regenera el mensaje. 3. Se imprime log. 4. Se reenvía el mensaje. 5. Se inicia la destrucción del la GKConnecion.
Notify	OnReceivedSignalNotify	1.Se regenera el mensaje. 2. En caso de túnel H.245 se incluye el campo provisionalRespToH245Tunneling 3. Se reenvía el mensaje.
Status	OnReceivedSignalStatus	1.Se regenera el mensaje. 2. En caso de túnel H.245 se incluye el campo provisionalRespToH245Tunneling 3. Se reenvía el mensaje.
Status Inquiry	OnReceivedStatusEnquiry	1.Se regenera el mensaje. 2. En caso de túnel H.245 se incluye el campo provisionalRespToH245Tunneling. 3. Se reenvía el mensaje.
Information	OnReceivedSignalInformation	1.Se regenera el mensaje. 2. En caso de túnel H.245 se incluye el campo provisionalRespToH245Tunneling. 3. Se reenvía el mensaje.

Cuadro 4.5: Callback functions en objeto GKConnection (cont.)

El diagrama de secuencia en la figura 4.7 muestra la interacción de las clases durante la llegada de un mensaje Connect. Una idea similar se aplica a la lectura y encaminamiento del resto de los mensajes H.225/Q.931.

4.6.8. Desvíos

A continuación se da una breve explicación de la forma en que se implementaron los desvíos de llamadas mencionados en la sección 4.5.1.2.

Como fue dicho en la sección 4.5.2.2, la llegada del mensaje *setup* inicia el proceso de análisis del desvío de llamada. Este análisis es lanzado desde el método GKConnection::OnReceivedSignalSetup() llamando a EncuentroDireccionALlamar(). En esta última se toma el mensaje recibido y obtiene el usuario/terminal destino de la llamada. Luego verifica si este destino tiene alguno de los desvíos configurados (SSDI, SSDO y SSDNC). Al final extrae de la configuración de desvío el terminal/usuario destino del desvío con su correspondiente dirección de transporte y asigna variables de

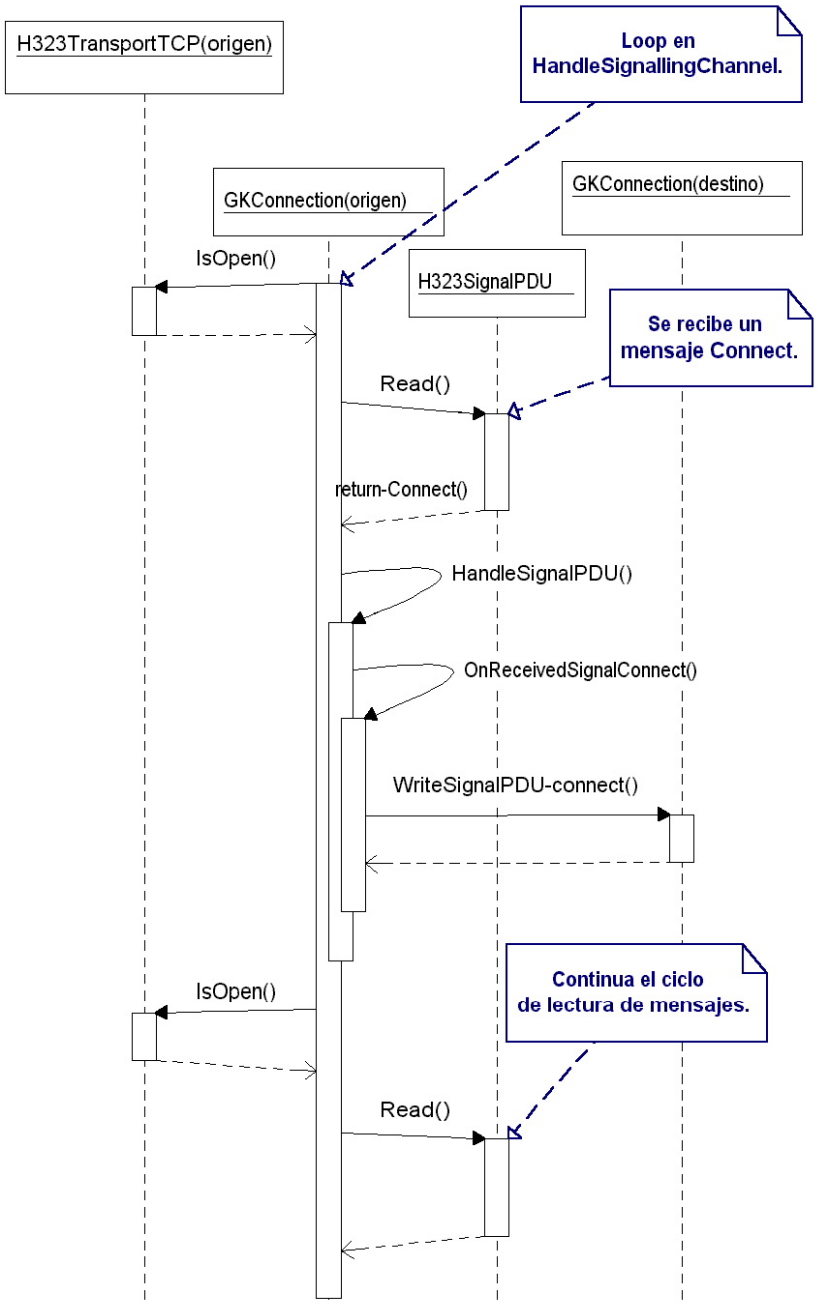

Figura 4.7: Lectura y encaminamiento de mensaje H.225/Q.931.

estado en el GKConnection que indican si hay desvío en la llamada y de qué

tipo se trata.

A la salida de este método la GKConnection::OnReceivedSignalSetup() se encuentra en alguna de las posibilidades mostradas por el cuadro 4.6.

Luego se crea la GKConnection par y a través de esta se envía el nuevo mensaje *setup* al terminal destino del desvío si corresponde o al terminal original.

En todos los casos de desvío los campos *fastStart* y *h245Control* son guardados en GKConnection y luego son transmitidos usando el mensaje *Connect*. Esto está perfectamente habilitado por la recomendación H.323 y permite prevenir que los terminales inicien los canales H.245 y RTP antes de establecer si la llamada es desviada o no. Si estos canales fueran establecidos con el terminal destino original, y luego resultara que la llamada es desviada, estos canales deben ser dados de baja. El problema reside en el hecho de que el *gatekeeper* no posee la posibilidad de mensajería H.245 para realizar las tareas mencionadas.

4.6.9. Programación concurrente

Uno de los aspectos más problemáticos de toda la implementación fue el manejo de los múltiples hilos de ejecución. La biblioteca OpenH323 es eminentemente *multithread* y toda implementación basada en ella deber ser muy cuidadosa con los detalles y problemas surgidos por la programación concurrente.

Se generaron varios bugs y comportamientos erróneos que no eran fáciles de identificar. Además, dichos bugs no eran reproducibles en todos los casos dependiendo de aspectos tales como: las características de la máquina donde corría el programa, las otras entidades H.323 que conformaran la zona y hasta el tipo de versión compilada que se corriera. Por ejemplo: era mucho más factible que se produjera un bug con la versión release que con la versión debug. La propia naturaleza de esta última, más grande y con más elementos adicionales para permitir el debug determinaba distinta velocidad de ejecución de los hilos y un comportamiento más estable del programa. Un trabajo de estudio detallado permitió identificar varias carreras críticas, o comportamientos disímiles en base a los tiempos de ejecución y a la interacción de los distintos hilos.

Como solución se implementaron distintos semáforos de forma de sincronizar los distintos hilos: GKConnection::sincronizaSetup, GKPro::mutexDeImprimoLog.

4.6.9.1. GKConnection::sincronizaSetup

Este tipo de semáforo del tipo PSyncPoint se utiliza para sincronizar dos hilos de ejecución cuando se rutea la señalización de llamada a través del *gatekeeper*. Cuando esto sucede, existe un hilo de ejecución y una instancia de GKConnection por cada terminal que cursa una llamada. Además las instancias representantes de las conexiones de una llamada están referenciadas entre sí por medio del atributo conexionPar. Cuando el *gatekeeper* recibe un mensaje *Setup* a través de una conexión con un terminal, debe crear una nueva conexión con el terminal llamado y enviar otro mensaje

Configuración del terminal/usuario destino	Información de salida	Estado GKConnection	Acciones a tomar
No hay desvío	Dirección de transporte es la del terminal destino original	No hay desvío	Se encamina la llamada normalmente
Hay SSDI (y ninguno, alguno o todos los otros tipos de desvío)	Dirección de transporte la del terminal destino del desvío	Hay desvío incondicional	Se inicia una llamada con el terminal destino a desviar y no con el destino original
Hay SSDO	Dirección de transporte la del terminal destino original y se guarda en variable la del terminal destino del desvío ocupado	Hay desvío en caso de destino ocupado	Se encamina la llamada hacia el terminal llamado original: Si este retorna un mensaje de ocupado se encamina la llamada hacia el terminal destino del desvío
Hay SSDNC	Dirección de transporte la del terminal destino original, se guarda en variable la del terminal destino del desvío no contesta y valor del tiempo de espera	Hay desvío no contesta en caso de vencer el timer	Se encamina la llamada hacia el terminal destino original, y se inicia un timer con el valor en segundos obtenido. Vencido el timer, se inicia una nueva llamada hacia el terminal destino del desvío
Hay SSDNC y SSDO configurado	Dirección de transporte la del terminal destino original, se guarda en variable la del terminal destino del desvío ocupado el no contesta y valor del tiempo de espera	Hay desvío ocupado y no contesta	Tiene prioridad el desvío ocupado. Pero si llega un **alerting** se aplica el desvío no contesta

Cuadro 4.6: Desvío en la GKConnection

Setup. Estas acciones se desencadenan en GKConnection::OnReceivedSignalSetup() por medio de la función GKEndPoint::MakeCall(). Dicha función crea el nuevo hilo de ejecución, una nueva clase GKConnection y llama a la función GKConnection::SendSignalSetup() de la instancia creada que es la que envía el *Setup* correspondiente. GKEndPoint::MakeCall() devuelve un puntero a la nueva GKConnection creada. La implementación determina que ese puntero debe ser guardado en el atributo conexionPar de la primera GKConnection y viceversa. El problema surge porque ahora hay dos hilos de ejecución independientes corriendo simultáneamente. La función SendSignalSetup() corre en el hilo de la conexión hacia el terminal llamado, mientras que MakeCall() corre en el hilo de la conexión llamante. A su vez, en la función SendSignalSetup() se referencia a la instancia GKConnection llamante por medio del atributo conexionPar. Dada la independencia de los hilos podría suceder (y sucede si no se pone el semáforo) que en SendSignalSetup() se referencia a la conexionPar sin que todavía se hayan inicializado los valores correctos (sin que se hayan asignado los punteros entre las dos instancias respectivas de GKConnection). Por esta razón, se utiliza este semáforo para sincronizar los dos hilos de ejecución, de modo que en SendSignalSetup() se hace esperar el hilo de ejecución hasta que en el otro hilo se haya asignado correctamente el atributo conexionPar.

La misma acción debe desarrollarse cuando se desvía una llamada creándose una tercera conexión por medio de GKEndPoint::MakeCall(). El mecanismo es similar en esta ocasión al considerar las dos conexiones resultantes.

4.6.9.2. GKPro::mutexDeImprimoLog

Este tipo de semáforo se implementó mediante un PMutex de la biblioteca PWLib. Lo que se precisaba era crear una zona de exclusión en el código de modo que dos hilos de ejecución no intentaran imprimir el log a la vez, o mientras lo estaba haciendo el otro. Esto se debe a que la escritura del log se produce cuando llega el mensaje *ReleaseComplete* o *DRQ* desde los terminales. Las distintas implementaciones de éstos determinan que el orden y tiempo de llegada de esos mensajes al *gatekeeper* sea muy variable. Por esta razón se utiliza este semáforo que permite que el primer hilo que ingresa a la función GKPro::ImprimoLog, mediante un llamado a la función Wait(), detenga la posible ejecución de la misma parte del código por otro hilo. Antes de retornar de la función se llama a la función Signal() que permite que los demás hilos continúen la ejecución y no se produzca un deadlock. A su vez, se realizan los chequeos necesarios para no ejecutar dos veces la escritura del log.

Capítulo 5

Pruebas realizadas

5.1. Introducción

En este capítulo se describirán las diferentes pruebas realizadas con el *gatekeeper* durante el proyecto, se detallan brevemente los éxitos y problemas encontrados, así como los resultados finales obtenidos.

El objetivo de estas pruebas fue doble: primero comprobar el correcto funcionamiento de las diferentes facilidades que se fueron sumando a las etapas del proyecto; mientras que el segundo objetivo fue buscar la compatibilidad del *gatekeeper* con un variado conjunto de terminales, los cuales se detallan más adelante.

En las primeras pruebas de comprobación generalmente se utilizaban los propios terminales del proyecto OpenH323 (ver cuadro 5.1), utilizando los diferentes módulos del *gatekeeper* a medida que estos iban siendo completados.

Luego, cuando se comprobó que el módulo en cuestión había obtenido cierta estabilidad y funcionalidad, se pasaba a las pruebas con el resto de los terminales, enfocando cumplir el segundo objetivo de interoperabilidad. Éste fue alcanzado en la mayoría de los casos aunque también se presentó algún problema que no pudo ser resuelto.

Básicamente todas las pruebas fueron realizadas en una LAN con direcciones privadas 192.168.1.0/24, el *gatekeeper* corriendo en PCs basados en procesadores Intel o AMD con diferentes sistemas operativos; Microsoft Windows 98, Millennium, NT y XP, así como GNU/Linux basados en distribuciones como: Debian 3.0, Mandrake 9.0 y Slackware 9.1.

También fueron realizadas pruebas de carga en el módulo III utilizando una Sun Ultra1 con sistema operativo GNU/Linux de la distribución Debian 3.0 en la que se ejecutó el *gatekeeper*, y fueron utilizados generadores de llamada de código libre también pertenecientes al proyecto OpenH323 (listados en quinto lugar en el cuadro 5.1) con los que se puede programar la realización de un determinado número de llamadas “simultáneas” , de forma que se configuró uno como llamante y otro como llamado.

Se probó con los dos modos de funcionamiento sabidos, llamada directa y llamada ruteada por *gatekeeper*, como era de esperar se obtuvo una mayor carga posible en el modo de llamada directa dado que requiere de menor

procesamiento.

Con el *gatekeeper* funcionando sólo con la señalización RAS soportó cargas de 30 y más llamadas simultáneas[1]. Mientras que cuando se probó ruteando las señalización Q.931 por el *gatekeeper* el número de llamadas simultáneas posibles bajó a las 10 o 12.

La utilización de este variado grupo de arquitecturas y sistemas operativos muestran la portabilidad que adquirió el proyecto al trabajar con el conjunto de bibliotecas OpenH323.

Como elemento adicional a las pruebas, y muy importante a la hora observar el funcionamiento y de encontrar los problemas de señalización que fueron surgiendo, se utilizó el *sniffer* de código libre Ethereal[ETH] , que basa su decodificación del grupo de protocolos de la recomendación H.323, en las bibliotecas del proyecto OpenH323. Además, fue de gran utilidad el uso de las trazas generadas por el propio *gatekeeper* con lo que se tiene un seguimiento de la ejecución del código en una situación de funcionamiento normal, no en el debugueo del código en el cual los tiempos de ejecución son diferentes por la propia información de debug. En algunos casos también resultó útil el registro de eventos o log generado por el equipo Cisco utilizado.

5.2. Terminales de prueba

Para la realización de las diferentes pruebas se emplearon como terminales los que se detallan en el cuadro 5.1.

Empleándose generalmente en la pruebas de comprobación el grupo de 5 terminales que abre la tabla, mientras que los restantes 4 fueron utilizados para la interoperabilidad, o mejor dicho, se comprobó y busco la interoperabilidad con estos últimos. Los resultados fueron variados y se muestran en la sección 5.4.

Cabe hacer la siguiente puntualización: los últimos tres equipos mencionados en el cuadro no son terminales en el sentido estricto de la recomendación H.323, es decir teléfonos H.323 nativos, si no que son *gateways* y se comportan como tal. Por esto último hubo siempre que considerar este factor y trabajar para que la interoperabilidad además abarcara este nuevo elemento de la arquitectura H.323.

La búsqueda de esta interoperabilidad se debió a la necesidad de obtener una comprobación del proyecto contra un tercer elemento que generase y recibiese mensajes de los protocolos involucrados en la recomendación H.323, pero que no basara su funcionamiento en las bibliotecas del proyecto OpenH323.

[1]Se entiende por simultaneas aquellas llamadas que se inician dentro de una ventana de 10 segundos máximo. Se empleó para esta prueba el generador de llamadas Callgen323 que provee el propio proyecto OpenH323.

Terminal	Fabricante	Módulo probado
OpenPhone	Proyecto OpenH323	I, II , III y IV
OhPhone	Proyecto OpenH323	I, II , III y IV
SimpH323	Proyecto OpenH323	I, II , III y IV
SimpH323 modificado[2]	Proyecto OpenH323 y Grupo proyecto Gatekeeper H323	I, II , III y IV
Callgen323	Proyecto OpenH323	III y IV
NetMeeting 3.01	Microsoft	I y II
Multitech	Multitech	II y III
Cisco 827	Cisco	II , III y IV
BCM	Nortel	II , III y IV

Cuadro 5.1: Terminales de prueba usados

5.3. Ejemplo de una prueba

5.3.1. Descripción de la prueba

En esta parte del documento se describirá brevemente, a modo de ejemplo, una de las pruebas realizadas al final del proyecto referente al módulo IV, con lo cual se comprueba implícitamente el cumplimiento de las especificaciones de los módulos anteriores dada la dependencia de las etapas subsiguientes respecto de las anteriores.

Como fue dicho anteriormente se realizaron pruebas con el *gatekeeper* funcionando sobre diferentes sistemas operativos y diferentes plataformas de hardware, la prueba que aquí se describe es sólo una de ellas, siendo el protocolo de pruebas idéntico para los restantes casos, salvo las pruebas de carga realizadas en el módulo III mediante los generadores de llamada.

Como se explicó en el punto **4.5.1.2** donde se describe el replanteo del problema que fue necesario hacer y qué alternativa se tomó para este módulo, se implementaron tres tipos de desvío de llamada, configurables para cada usuario en el archivo de configuración (ver A.1.2.4).

Estos son:

1. Desvío incondicional (SSDI)
2. Desvío en caso de no contesta (SSDNC)
3. Desvío en caso de ocupado (SSDO)

La prueba que se detalla aquí consistió en utilizar el *gatekeeper* y algunos terminales H.323 de software derivados de la biblioteca, conjuntamente con un equipo Cisco en el que se tenían dos terminales H.323 en una red LAN, y establecer llamadas entre ellos y realizar desvíos.

Datos de los equipos utilizados:

- Se implemento una red LAN privada con direcciones IP privadas con máscara de 24 bits del tipo 192.168.1.x compuesta por dos PC's y el equipo Cisco que se describen seguidamente conectados mediante un HUB Planet.

- Se ejecutó el programa *gkmod4.exe* en una laptop Toshiba Satellite la cual posee un procesador Mobile Intel Celeron de 2.0GHz y 256MB de memoria sobre el Sistema Operativo Windows XP Professional.
- Se utilizaron dos terminales H.323 de la biblioteca un llamado *Openphone* en el cual se tiene GUI (el listado en primer lugar en el cuadro 5.1) y otro llamado *Simple* (el cuarto en el cuadro 5.1) el cual es muy similar al anterior pero sin GUI, este último fue modificado y recompilado de forma de que utilizara otro puerto diferente para el canal RAS de forma de evitar conflictos dado que se ejecutaron ambos terminales en la misma PC con una única tarjeta de red. Dicha máquina es una PC Intel Pentium IV con un procesador de 1.4GHz y 128MB de memoria corriendo el Sistema Operativo Windows ME.
- Se utilizó un Router Cisco modelo 827 (octavo del cuadro 5.1) corriendo la versión IOS 12.2 la cual implementa H.323 versión 4 el cual posee cuatro puertos de voz FXS (*Foreign Exchange Station*), con lo cual el equipo puede ser utilizado como un *gateway* H.323 con cuatro terminales telefónicos tradicionales conectados.

Se realizó un protocolo de pruebas en el que se registraban con el *gatekeeper* los distintos terminales con determinados alias de usuario, donde a un determinado usuario se le configuraba un desvío determinado a probar. Posteriormente se realizaba la llamada comprobando la admisión de la misma de acuerdo a la clase de servicio del usuario y el destino de la misma, el ruteo de ésta, la realización del desvío correspondiente y el establecimiento de los canales de audio. A tales efectos se confeccionaron tres planillas las que se iban llenando con los resultados obtenidos de forma de ir cambiando la situación en forma ordena y sistemática. Se repitió el proceso cambiando el tipo de terminal originante para el desvío en cuestión y a su vez se iba cambiando el terminal destino.

Una vez probado un desvío se repitió el mismo procedimiento con los diferentes tipos de desvíos.

Notación: cuando se hable de terminal A se hace referencia al originante de la llamada, terminal B hace referencia al destino (intermedio) el cual tiene configurado un desvío, y terminal C habla del destino último.

Se debe tener en cuenta que el caso de llamadas entre los terminales conectados al Cisco (A y B en la puertos de voz) no se puede probar dado que si la llamada es entre terminales detrás del *gateway,* éste los comunica internamente y la señalización no pasa por el *gatekeeper*, esto es así por como lo implementa el equipo Cisco. Por lo que se usó la configuración en la que los terminales A y C sean los conectados al equipo Cisco de manera que la señalización pase por el *gateleeper*.

5.3.2. Resultados de la prueba

De las pruebas se concluyó que el *gatekeeper* cumple con los requerimientos, desviando las llamadas en los casos en que corresponde, y dando

las señalización adecuada al terminal llamante, es decir ocupado o llamando de acuerdo al caso.

También se probó con situaciones no típicas que se detallarán posteriormente en las cuales el comportamiento detectado fue el esperado dada la solución brindada en la implementación.

1. Primero cabe destacar que como se implementaron los servicios suplementarios solo se permite un nivel de desvío, es decir si un terminal (B) llamado tiene un desvío configurado hacia un terminal C y a su vez éste tiene un desvío programado hacia otro terminal D, éste último desvío no es tenido en cuenta. Esto se implemento así por un criterio de diseño. Esta implementación tiene como consecuencia que se elimina la posibilidad de que se produzcan loops como por ejemplo que el terminal D sea el llamante (A). Esto fue probado comportándose como se esperaba.

2. Otro caso diferente probado fue el siguiente: el terminal A llama al terminal B, éste tiene configurado un desvío no contesta hacia el terminal C, mientras se está llamando el terminal A recibe señal de llamada, expirado el timer se pasa a desviar la llamada hacia el terminal C, estando éste ocupado el terminal A pasa a recibir señal de ocupado. Esto es así por diseño, fue probado resultando el comportamiento el esperado.

3. En el módulo 2 se implementaron las políticas de admisión de llamadas, donde dependiendo de quienes sean los involucrados en la llamada y los permisos que tengan se la admite o no, este control se hace a partir del mensaje *AdmissionRequest*. Esto es contemplado en parte en caso de existir desvíos debido a su implementación. Se presenta un problema en el control de las restricciones, éste es el caso de que un terminal A tenga restricto llamar al terminal C, pero si llama a un terminal B y éste tiene configurado un desvío hacia el terminal C la llamada terminará estableciéndose.

5.4. Resultados generales

En el siguiente cuadro (5.2) se muestra como fue el comportamiento de los distintos terminales probados a lo largo del proyecto, tomando como base tres elementos. La compatibilidad y funcionalidad con la señalización RAS generada por el *gatekeeper*, la compatibilidad con la señalización de llamadas H.225/Q.931, y la compatibilidad con los servicios de desvío generados a partir de la cuarta etapa del proyecto.

El cuadro muestra que se alcanzó casi completamente la compatibilidad deseada con los diferentes terminales utilizados.

Los problemas mayores como siempre surgieron con los equipos *gateways,* BCM Nortel, Multitech y Cisco 827.

Dichos problemas se generaron básicamente cuando se buscó generar llamadas desde y hacia equipos de distintos fabricantes. Por ejemplo llamadas que se iniciaban en el equipo Multitech y tenían como destino extensiones en el *gateway* BCM.

Terminal	Resultados		
	Compatibilidad - RAS	Compatibilidad - H.225/Q.931	Compatibilidad con Desvíos
OpenPhone	Compatible	Compatible	Compatible
OhPhone	Compatible	Compatible	Compatible
SimpH323	Compatible	Compatible	Compatible
SimpH323 modificado[3].	Compatible	Compatible	Compatible
Callgen323	Compatible	Compatible	Compatible
NetMeeting 3.01	Compatible	No fue probado	No fue probado
Multitech	Compatible	Con problemas	No fue probado
Cisco 827	Compatible	Compatible	Compatible
BCM	Compatible	Compatible (problemas de codecs con el Cisco 827 y Multitech)	Compatible

Cuadro 5.2: Resultados obtenidos

Muchas veces las dificultades provenían por la incompatibilidad de los codecs empleados por los equipos y no por problemas de señalización. Constatándose incluso problemas de diálogo H.245 entre el equipo BCM de Nortel y el Cisco 827. Estos últimos no conseguían ponerse de acuerdo a la hora de abrir los canales de voz mediante el protocolo H.245 y los paquetes de voz sólo viajaban en uno de los sentidos. Debido a que el *gatekeeper* no encamina este último protocolo, el problema está fuera del ámbito del presente proyecto y no se realizaron mayores avances para buscar su solución .

Por otro lado, el equipo Multitech, por razones ajenas al presente proyecto dejó de funcionar y no pudo ser empleado en la pruebas finales de desvío de llamada.

Apéndice A

Documentación del Programa

A.1. Manual del administrador

El presente manual es aplicable al programa ejecutándose tanto sobre sistemas operativos Windows (98/ME/NT/2000/XP) o sobre plataformas Linux; las diferencias radican en la forma en que fue compilado, por más información sobre esto ver apéndice B.

Este programa implementa un *gatekeeper* según la recomendación ITU-T H.323 versión 4, el cual realiza las funcionalidades básicas conferidas en dicha recomendación así como ciertas funcionalidades definidas como opcionales en ella, como por ejemplo, la implementación de políticas de admisión de llamadas, manejo de señalización de llamada H.225/Q.931, implementación de servicios suplementarios (desvío de llamadas), etc.

Las políticas de admisión de llamada son implementadas en base a los identificadores de los usuarios, ya sean alias o números según E.164, donde cada uno de ellos tendrá determinadas posibilidades de realizar y recibir llamadas, así como restringir los destinos hacia los cuales les es posible llamar. Además cada usuario particular podrá tener configurados determinados servicios suplementarios los que estarán disponibles en caso de que la señalización de llamada este siendo procesada por el *gatekeeper*. El que la señalización de llamada sea manipulada por el *gatekeeper* o no, es configurable.

El programa necesita cierta información para su funcionamiento, parte de esta información le puede ser suministrada en su invocación en la línea de comandos, esta misma información así como otros datos necesarios le pueden ser suministrados a través de un archivo de configuración de tipo INI según se explica en el punto A.1.2 , en caso de información presente en este archivo que también sea suministrada como parámetro en el inicio del programa se prioriza esta última.

Como se describe luego, una vez iniciada la ejecución del programa (ver A.1.1) éste abrirá una consola en la cual se puede visualizar cierta información así como ejecutar determinados comandos (ver A.1.3).

El programa cuenta con la posibilidad de registrar su actividad median-

te una traza, ésta puede ser de utilidad para la determinación de posibles problemas, esta traza se puede presentar en pantalla o escribirla en un archivo según se explicará posteriormente, se puede elegir el nivel de detalle de la información trazada, para ello se tiene una escala de niveles que va de 1 a 6 siendo este último el más específico.

Por otra parte el programa lleva un registro de las llamadas cursadas, para ello genera un archivo de log de llamadas, el cual es generado en el directorio donde se encuentra el ejecutable del programa y tiene por nombre el nombre del *gatekeeper* y extensión *.log*. También es posible desplegar este archivo en la consola del programa, por información del formato de la información registrada, etc (ver A.1.3).

A.1.1. Inicio del gatekeeper

En la inicialización del programa, como fue dicho anteriormente, éste carga en primera instancia los parámetros que le son suministrados en su invocación en la línea de comandos.

A continuación se detalla la puesta en funcionamiento del programa así como los parámetros que le pueden ser suministrados.

NOMBRE

gk - Gatekeeper h323

SYNOPSYS

gk [- h help] [- i dirección IP de la interfaz] [- n identificador del gatekeeper] [- p puerto del canal RAS] [- t nombre del archivo de traza] [- T nivel de la traza] [-f archivo de configuración] [- r modo ruteado por gatekeeper] [- q puerto del canal de señalización]

OPCIONES

Los parámetros requeridos por el programa le son proporcionados mediante línea de comandos con las siguientes opciones.

- i: Especifica la dirección IP de la interfaz en la que estará escuchando el gatekeeper. Si no se le proporciona una, por defecto ésta será la de loopback 127.0.0.1.

- n: Especifica el identificador que tendrá el gatekeeper, es un string de caracteres por el cual se lo reconocerá.

- p: Puerto TCP donde escuchará mensajes RAS, por defecto tomara el 1719.

- t: Especifica el nombre del archivo donde se escribirá la traza durante el funcionamiento.

- T: Determina el nivel de la traza, esto es el nivel de detalle, pudiendo tomar valores del 1 al 6, siendo el último el de mayor detalle. Por defecto tomará 1.

- f : Especifica el nombre del archivo de configuración que debe cargar. Por defecto tomará el archivo config.ini presente en el directorio donde se encuentra el ejecutable si no se especifica la ruta completa.

- r: Indica que las llamas serán ruteadas por el gatekeeper.

- q: Especifica el puerto TCP donde se escuchará los menajes de señalización de llamada. Por defecto tomará el 1720.

Una vez iniciado el *gatekeeper,* éste despliega en pantalla cierta información referente a algunos de los parámetros que está utilizando, según se describe a continuación, y habilita una consola la cual dispone de un conjunto de comandos que se explican en el punto A.1.3.

La información que se despliega en pantalla es: la versión del programa que está ejecutando, el nombre del archivo de configuración que ha sido cargado, el nombre del archivo de traza donde se está escribiendo ésta, el nivel de dicha traza, el puerto TCP que se está usando para el canal RAS y, por último, el puerto TCP que se está utilizando para la señalización de llamada. Esto se muestra en la figura A.1.

Figura A.1: Inicio del programa

A.1.2. Archivo de configuración

Como se mencionó anteriormente al iniciarse el *gatekeeper* éste carga un archivo de configuración donde se encuentran un conjunto de datos que determinarán su forma de funcionamiento.

Dicho archivo es del tipo INI y esta dividido en diferentes secciones. Puede contener comentarios, debiendo comenzar éstos con el caracter **";"** para indicarlo.

Una vez en funcionamiento el programa se puede modificar el archivo de configuración y volver a cargarlo, para eso se utiliza el comando *rcg* de la interfaz de comandos (ver A.1.3), una vez recargado éste los cambios tendrán efecto en las nuevas llamadas, no en las llamadas que permanecieron establecidas durante la recarga.

Las secciones que componen el archivo son: General, Zona, Clase, Usuario, las dos últimas pueden repetirse, se diferenciarán entre si por un número al final de su nombre, esto se explica en las respectivas secciones.

Para los alias de tipo numérico según E.164 se tiene la posibilidad de definir rangos de números, es decir definir un conjunto de alias numéricos definido por una regla, no por extensión. Se pueden tener rangos de diferentes longitudes. Cada rango estará definido por una parte fija y una variable, la primera serán dígitos específicos y la parte variable estará determinada por caracteres punto "." que indican, como una expresión regular, un dígito cualquiera cada punto, de modo que cada rango será de longitud fija (igual a la parte fija más la variable) y los distintos alias pertenecientes a él se diferenciaran entre sí por los dígitos de la parte variable. Por ejemplo: el rango "15.." estará formado por todos los números que van desde el

1500 al 1599.

Estos rangos serán útiles en las secciones clase y usuario.

A continuación se detallarán las distintas secciones que componen el archivo de configuración.

A.1.2.1. Sección General

En esta sección se encuentran los parámetros generales del *gatekeeper*, éstos son los que en se le pueden suministrar al *gatekeeper* desde la línea de comando en su inicialización y que en caso de no ser especificados allí son tomados de este archivo.

Este grupo de parámetros no será recargado en caso de ser recargado el archivo de configuración durante la ejecución programa mediante el comando *RCG*, si se desea modificar alguno de ellos es necesario finalizar la ejecución programa y volver a iniciarla con el archivo modificado.

Los campos presentes en esta sección son:

- **traza** : nombre del archivo de traza, si no esta presente la traza será escrita en un archivo de nombre ptraceGk-fecha-hora.txt (en formato yy-mm-dd_hh-mm) siendo ésta la fecha y hora de comienzo de la ejecución del programa.
- **nivelTraza** : nivel de la traza, entero entre 1 y 6 creciente para mayor nivel de detalle.
- **dirección IP** : dirección IP de la interfaz en la que estará escuchando el *gatekeeper* los mensajes RAS y señalización si se esta en modo *ruteadoPorGK*.
- **puertoRAS** : puerto TCP donde escuchará mensajes RAS. Típicamente el 1719.
- **nombreGK** : es un nombre que identifica al *gatekeeper*.
- **ruteadoPorGK** : valor booleano (“si” o “no”) que determina si el modo de funcionamiento será *gatekeeper routed call* o *direct call*. En el primer caso la señalización de llamada H.225/Q.931 será ruteada por el , mientras que en el segundo caso la señalización será directamente intercambiada entre los terminales. Debe hacerse notar que en caso de que la señalización de llamada no sea ruteada por el *gatekeeper* no se tendrá la posibilidad de brindar los servicios suplementarios.
- **puertoQ931** : puerto TCP que el *gatekeeper* utilizará para la señalización Q.931. Típicamente será el 1720.

A continuación se da un ejemplo de la “sección general” de un archivo de configuración.

```
[general]
traza=ejemplo_traza.txt
nivelTraza=4
direccionIP=192.168.100.54
puertoRAS=1719
nombreGK=gatekeeperH323
ruteadoPorGK=si
puertoQ931=11720
```

Cuadro A.1: Ejemplo de Sección General

A.1.2.2. Sección Zona

Esta sección contiene los datos necesarios para la determinación de la zona a ser atendida por el *gatekeeper*. Esto es que terminales tienen permitido registrarse con él y utilizarlo.

Existen dos criterios posibles en base a los cuales determinar qué terminales pueden registrarse, en primer lugar de acuerdo al alias del terminal y en segundo lugar tomar como criterio además de controlar el alias controlar también la dirección IP desde la que está intentando registrarse el terminal. Esto se elige en el campo "modo" de esta clase.

Como se verá posteriormente existe una sección en este archivo para cada usuario (ver A.1.2.4) perteneciente a la zona a ser atendida por el *gatekeeper*, una vez leído este archivo se tendrán los datos necesarios para efectuar estos controles dado que en ellas se tienen el o los alias posibles y la dirección IP, los que se compararan con los presentes en las solicitudes entrantes al *gatekeeper*.

Esta sección contiene los siguientes campos:

- **modo** : determina el modo de definición de la zona a ser atendida por el *gatekeeper*, pudiendo tomar los valores "0" y "1". El valor "modo = 0" refiere a que se tomará como modo de control el de alias, mientras que el valor "modo=1" hará que se emplee como criterio de control el alias y la dirección IP conjuntamente.
- **redIP** : Es la dirección de la subred a la cual deben pertenecer los terminales.
- **redmascara** : Mascara utilizada para saber si una dirección IP determinada pertenece o no a la subred indicada en el campo anterior.

Estos dos últimos campos pueden repetirse varias veces y se diferenciaran agregando números crecientes y consecutivos al final del nombre del campo, por ejemplo "redIP1", "redmascara1", "redIP2", "redmascara2", etc.

En el cuadro siguiente se da un ejemplo de la "sección zona" de un archivo de configuración.

```
[zona]
; modo=0 soloAlias
;modo=1 aliasEIP
modo=1
redIP1=192.168.100.0
redMascara1=255.255.255.0
redIP2=127.0.0.0
redMascara2=255.255.255.0
```

Cuadro A.2: Ejemplo de Sección Zona

A.1.2.3. Sección Clase

Se tienen varias secciones denominadas *claseN* donde cada una define una clase de servicio a la que pertenecerá un cierto grupo de usuarios, cada usuario deberá pertenecer a una y sólo una clase de servicio. Estas se diferencian entre sí agregando un número consecutivo creciente al final del nombre, por ejemplo clase1, clase2, etc.

Básicamente las diferencias entre las distintas clases de servicio residen en las posibilidades de efectuar y recibir llamadas que tienen los terminales pertenecientes a ellas y hacia que terminales se le permite llamar, pudiendo hacerse combinaciones de estas posibilidades para generar diferentes clases mediante combinaciones en los campos presentes en ellas. Los servicios suplementarios no forman parte de la clase de servicio sino que directamente son configurados para cada usuario según se describe en A.1.2.4.

Los campos presentes en cada una de las clases de servicio son:

- **tipo**: Identifica el tipo de clase de servicio, este tipo será fijo y tendrá el valor “servicio_basico”, dejándose la posibilidad de que posteriormente se implementen otro tipo de clases de servicio.
- **entrantes**: Booleano que indica si los usuarios pertenecientes a esta clase de servicios podrán recibir llamadas o no, los valores posibles para este campo son “si” y “no”.
- **salientes**: De forma similar al anterior, éste indica si los usuarios podrán efectuar llamadas, los valores posibles son “si” y “no”.
- **lista_de_restricciones**: Los usuarios pueden tener uno, o un grupo de destinos restrictos, es decir usuarios a los que no les es permitido llamar, si no existen restricciones para los usuarios de esta clase de servicio, éste campo contendrá el valor “no”, en caso contrario este campo contendrá los usuarios restrictos, separados por coma, ya sean alias o identificadores E.164, en este último caso si se trata de todo un rango de números no es necesario especificarlos por extensión sino que se puede especificar por los caracteres fijos seguidos por los puntos necesarios que determinan el rango según se explicó al comienzo del presente manual en el punto A.1.2.

Ejemplo de una “sección clase” de un archivo de configuración.

```
[clase1]
;llamadas entrantes y salientes con restricciones
tipo=servicio_basico
entrantes=si
salientes=si
lista_de_restricciones=3401,usuario3
```

Cuadro A.3: Ejemplo de Sección Clase

A.1.2.4. Sección de Usuario

Se tiene una sección usuario por cada usuario perteneciente a la zona administrada por el *gatekeeper*, las distintas secciones usuario se diferenciarán entre sí de la misma forma que las secciones clase, es decir sus nombres serán la palabra "usuario" seguida de un número consecutivo creciente a partir de 1, por ejemplo "usuario1", "usuario2", etc.

En estas secciones se tendrá información de cada usuario particular, por ejemplo alias, dirección IP, clase de servicio a la que pertenece y la configuración de sus servicios suplementarios.

Los servicios suplementarios sólo estarán disponibles si se está en el modo de funcionamiento llamada ruteada por *gatekeeper*.

Estos servicios suplementarios son tres diferentes tipos de desvíos configurables en forma independiente a través de los parámetros "forwardUnconditional", **"**forwardBusy**",** "forwardNoReply**",** "NoReplyTimer". Estos desvíos deben ser configurados para cada usuario en particular.

Todos ellos funcionan de la siguiente forma: si no se los desea habilitar los respectivos parámetros deben contener el valor *no*, y en caso de que se los desee habilitar estos parámetros contendrán el alias del usuario hacia el que se debe desviar la llamada.

El primer desvío en orden de prioridad es el llamado "Desvío Incondicional", como su nombre lo indica si éste esta habilitado se desviarán incondicionalmente todas las llamadas entrantes a este usuario hacia otro, el cual es especificado en el campo "forwardUnconditional" a través de su alias, mientras que si el parámetro antes mencionado tiene el valor *no* este desvío estará inhabilitado. En caso de estar configurado éste será prioritario a los otros desvíos no teniéndoselos en cuenta. Por ejemplo si un usuario A llama a un usuario B el cual tiene configurado el desvío incondicional hacia un usuario C, nunca llegará señalización de llamada a el usuario B, sino que será reencaminada en el *gatekeeper* hacia el usuario C.

El siguiente desvío es el denominado "Desvío en caso de Ocupado", en este caso las llamadas entrantes a este usuario serán desviadas en el caso de que se encuentre ocupado, de forma análoga al anterior el destino último de la llamada será el indicado en el parámetro "fordwardOnBusy" por el alias o en caso de estar inhabilitado este parámetro contendrá el valor *no*. En este caso la señalización de llamada será dirigida hacia el terminal B (el que tiene configurado el desvío) y en el caso de que la respuesta sea que está ocupado se reencaminará la señalización de llamada hacia el usuario C.

Por último se tiene el desvío llamado "Desvío no Contesta", en este caso las llamadas serán desviadas hacia el destino indicado en el parámetro "forwardNoReply" si el terminal destino original no contesta antes de que transcurra el tiempo indicado en el parámetro "NoReplyTimer". En este caso también la señalización de llamada irá desde el terminal llamante (denominémoslo A) a el *gatekeeper*, éste la encaminará hacia el usuario B, se esperará la respuesta del usuario B durante el tiempo especificado y si no contestó, la señalización se reencaminará hacia el usuario C.

La implementación realizada admite solo un nivel de desvío, es decir que si el usuario **C** de los ejemplos también tiene configurado un desvío este no es tenido en cuenta.

La secciones usuario contendrán los siguientes campos:

- **nombre**: Es un *string* que representa el nombre del usuario. Es opcional y no es usado para control alguno.
- **alias**: Contendrá el alias con el que se registrará el usuario, este campo es necesario y será el utilizado para permitirle o no la registración y posteriormente para identificarlo y determinar sus propiedades. Si se trata de alias numéricos es válido lo explicado en el punto A.1.2 respecto de los rangos numéricos.
- **direccionIP**: Contendrá la dirección IP del usuario. Ésta será utilizada para determinar si el terminal pertenece o no a la zona administrada por el *gatekeeper* el caso de funcionamiento en modo alias e IP, así como para localizarlo en el caso de llamadas dirigidas hacia él.
- **clase**: Contendrá la clase de servicio a la que pertenecerá el usuario, este campo solo podrá tener un valor del tipo "claseN", por ejemplo "clase2".

El siguiente grupo de parámetros corresponde a los servicios suplementarios según se explicó al comienzo de esta sección.

- **forwardBusy**: Desvío en caso que el usuario llamado esté ocupado. En caso de existir este desvío este campo contendrá el *alias* del terminal al que se deben desviar las llamadas entrantes hacia este terminal en caso de que él se encuentre ocupado. Si no desea programar este desvío para el usuario en cuestión este campo contendrá el valor "no".
- **forwardUnconditional**: Desvío incondicional, si está programado se desviarán todas las llamadas entrantes a este usuario hacia el terminal especificado en este campo, análogamente al anterior este campo contendrá el valor "no" en caso de que no se desee tener este desvío.

Los dos parámetros siguientes corresponden al caso de "Desvío no Contesta", como ya fue explicado, a diferencia de los anteriores en este caso se necesita un parámetro adicional que es el tiempo que se debe esperar para proceder a desviar la llamada .

- **forwardNoReply**: Desvío en caso de no contesta. Análogamente a los desvíos anteriores este campo contendrá el alias del terminal al que se debe desviar la llamada, o en caso de no desear programar este desvío contendrá el valor "no".

- **NoReplyTimer**: Valor del tiempo que se debe esperar a que el destino atienda la llamada antes de desviarla. Este valor debe ser un número entero positivo expresado en segundos. Por ejemplo 10 ó 15. En caso de que el campo fordwardNoReplay tenga el valor "no" el timer no será leído por el programa por lo que puede estar vacío. Se debe tener en cuenta que un valor negativo o cero hará que se esté esperando indefinidamente por la respuesta.

Ejemplo de una "sección usuario" de un archivo de configuración.

```
[usuario1]
nombre= Alberto Rodriguez
alias=arodriguez
direccionIP=192.168.100.57
clase=clase1
forwardBusy=1030
forwardUnconditional=no
forwardNoReply=no
NoReplyTimer=
```

Cuadro A.4: Ejemplo de Sección Usuario

A.1.3. Interfaz de Comandos

Como se dijo al comienzo del presente manual una vez iniciada la ejecución del programa éste presenta un interfaz de comandos para el ingreso de algunos comandos y la visualización en pantalla de cierta información en tiempo de ejecución.

El conjunto de comandos disponibles en esta consola es:

RCG [-a archivo_de_configuración] Este comando efectúa la recarga del archivo de configuración, por defecto recargará el de nombre "config.ini" que se encuentre en el mismo directorio que el ejecutable, es posible recargar otro archivo de configuración mediante el ingreso del parámetro "-a" y la ruta completa hacia el nuevo archivo.

Esto da por ejemplo la posibilidad de habilitar nuevos usuarios o deshabilitar algunos existentes, cambiar la configuración de los servicios suplementarios de los usuarios, así como las clases de servicio a la que pertenecen, etc. Luego para activar estos cambios en el archivo de configuración se recarga del mismo. Los cambios en el archivo de configuración recargado tendrán efecto sobre las nuevas llamadas, no sobre las llamadas en curso durante la recarga.

En el archivo de configuración existe cierta información que no puede ser modificada con el programa en ejecución como por ejemplo la dirección IP de la interfaz de red, esta información es la presente en la sección general de dicho archivo, por lo que esta sección no es recargada por este comando.

Si se quiere modificar estos parámetros se debe finalizar la ejecución del programa y volver a iniciarla con el archivo de configuración modificado

o suministrarlos a través de los parámetros en la linea de comando en la inicialización del programa (ver A.1.1).

LOG Este comando habilita o deshabilita el despliegue del Log de llamadas por consola, este se seguirá registrando en el archivo de log aun cuando se lo despliegue en pantalla.

Este comando no tiene parámetros, su funcionamiento es el de conmutar entre los dos estados posibles: log por pantalla habilitado, o inhabilitado.

Una vez ingresado el comando se muestra en pantalla uno de los siguientes mensajes: "Log por pantalla habilitado" o "Log por pantalla inhibido" de forma de saber en qué estado se está.

Si el log por pantalla está habilitado luego de finalizada cada conexión se desplegará la información referente a ésta en la consola.

Contendrá información sobre el origen y el destino de la llamada, así como la fecha y hora de comienzo y fin, además del ancho de banda utilizado por la conexión.

A continuación se detalla el formato de la información presentada en el log por pantalla, los datos tendrán idéntico formato en el archivo de log pero allí solamente estarán los datos mismos, no la descripción que se muestra en pantalla y se tendrá una línea por conexión donde los datos están separados por el caracter "#".

- **Origen** (Alias:IP): contiene el alias y la dirección IP del usuario originante de la llamada.

Tendrá el formato AAAAA:IIIIII, donde: AAAAA es el alias del terminal, el cual tiene número de caracteres variable. Si hay varios alias estarán separados por espacios blancos. La otra parte del campo origen (representado por IIIIII) es la dirección IP del terminal, serán 4 grupos de 3 caracteres como máximo separados por punto, por ejemplo 192.168.1.3.

- **Destino** (Alias:IP): idéntico al anterior salvo que contendrá la información del usuario destino de la llamada.

- **Fecha_hora_de_inicio**: contendrá un total de 15 caracteres que indican fecha y hora de inicio de la conexión, con el formato:

AAAAMMDD HHMMSSPP, donde:
AAAA : Año expresado con 4 caracteres.
MM : Mes, 2 caracteres.
DD : Día, 2 caracteres:
La fecha y hora estarán separados por un único espacio en blanco.
HH : hora, 2 caracteres.
MM : minutos, 2 caracteres.
SS : segundos, 2 caracteres.
PP : dos caracteres para identificar a.m. o p.m.

- **Fecha_hora_de_fin**: idéntico al anterior salvo que contendrá la información referente al fin de la llamada.

- **Ancho_de_Banda** (x100 b/s): contendrá un total de 5 caracteres para mostrar el ancho de banda bidireccional solicitado para la conexión, en unidades de 100 bits por segundo.

En la figura se muestra un ejemplo de log de llamada.

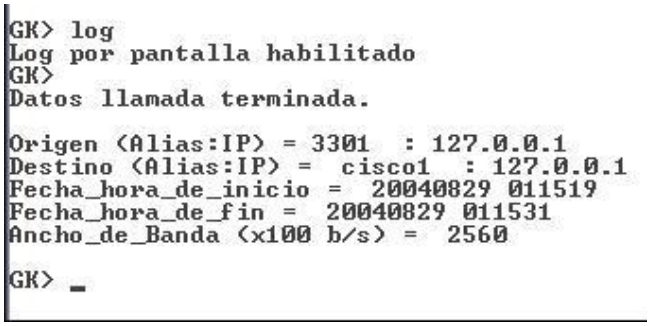

Figura A.2: Ejemplo de log por consola

HLP Muestra una breve ayuda sobre los comandos disponibles y su formato correcto.

TRZ [-n nuevo nivel traza, -c traza por consola, -a traza por archivo] Con este comando se pueden realizar dos acciones, por separado o conjuntamente, lo que es especificado por los parámetros con los que se ejecuta el comando.

Una de ellas es cambiar el nivel de traza al nuevo nivel indicado por el parámetro "-n", el cual es opcional, de modo que si no está presente se lleva el nivel de traza al nivel mínimo, o sea 1. De lo contrario seguido del parámetro "-n" debe ir el valor al que se desea cambiar el nivel de la traza. Esto es válido tanto para traza por consola como por archivo.

Se recuerda que el nivel es un número entero entre 1 y 6, indicando el nivel de detalle de la traza siendo el nivel 6 el de mayor detalle.

La otra acción que se puede tomar con este comando es cambiar la salida de la traza de archivo a consola, o viceversa. Si la traza se esta escribiendo en un archivo al ejecutar este comando con el parámetro "-c" se pasará a desplegar la traza en consola, a diferencia de log de llamadas la traza no es desplegada en pantalla y a su vez escrita en el archivo, solo se tiene una salida por vez.

De la misma forma si la traza se esta desplegando en consola mediante este comando con el parámetro "-a" se pasa a escribirla en un archivo.

FIN Este comando finaliza la ejecución del programa.

REG Este comando es utilizado para visualizar en pantalla los usuarios que se tienen registrados en ese instante. Hace que se desplieguen los alias o identificadores E.164 de los usuarios registrados, en caso de usuarios con más de un alias se mostrará sólo el primero.

LLA Este comando es utilizado para visualizar en pantalla las llamadas en curso actualmente. Se muestra en pantalla una línea por llamada actualmente en curso en donde se muestra el origen, el destino y el *callidentifier* (valor usado en la recomendación H.323 para identificar cada llamada) de la llamada.

Apéndice B

Compilación del proyecto

Este proyecto como ya fue explicado en secciones anteriores cuenta con una portabilidad entre los sistemas operativos Microsoft Windows y Linux, la cual se debe a las bibliotecas utilizadas, las que brindan esta posibilidad al programador cliente. En virtud de esto el código obtenido puede ser compilado para ser ejecutado en dichos sistemas operativos, esto fue realiado para las versiones de windows 98/ME/XP/NT/2000 y para las distribuciones de linux Slackware 10.0, Debian 3.0.

Seguidamente se describirá brevemente como se realizó esto en los diferentes entornos de desarrollo utilizados.

B.1. Compilación en Visual C++ 6.0

Las versiones de las bibliotecas utilizadas en este proyecto para windows son: OpenH323 1.9.5, Pwlib 1.3.5.

En primer lugar se deben obtener los códigos fuentes de las bibliotecas OpenH323 y Pwlib, estos se encuentran en el sitio *http:\\sourceforge.net*.

Se debe tener en cuenta que las versiones antes mencionadas de las bibliotecas ya contienen el archivo de proyecto de tipo *dsw* para el ambiente de desarrollo Visual C++ versiones V5, V6 y V7.

Posteriormente se deberán compilar estas siguiendo las indicaciones dadas en el *readme* que se encuentra en *http:\\www.openh323.org\ build.html*.

Luego de compiladas las bibliotecas se debe descomprimir el archivo "gatekeeper.zip" el cual contiene el archivo de proyecto "gatekeeper.dsw", abrir éste con el Visual C++ y compilar alguna de las opciones siguientes.

- release: genera el ejecutable con la opción de traza [1] habilitada.
- debug: genera un ejecutable con información de debug para ser utilizado en el Visual Studio.
- no trace: genera el ejecutable con la opción de traza deshabilitada.

[1] Habilita un grupo de macros insertados en el código que genrará un archivo de traza durante la ejecución del mismo. Como ya se dijo, esta traza podrá tener diferentes niveles.

Una vez obtenido el ejecutable se deberá copiar el archivo "config.ini", al mismo directorio donde se encuentre el primero. Este archivo de configuración es indispensable, deberá contener los parámetros necesarios según se detallan en el manual del administrador.

B.2. Compilación en Gnu/Linux

Las versiones de las bibliotecas utilizadas para Gnu/Linux son: 1.11.7 de la OpenH323 y 1.4.11 de la Pwlib. Las mismas puden obtenerse en el sitio mencionado en la sección anterior. También se disponen de instrucciones detalladas para su complicación en *http:\\www.openh323.org\build.html*.

Las herramientas utilizadas para la compilación son las estándar de GNU/Linux: gcc, make, asignación de variables de ambiente, etc.

Una vez compiladas éstas, se deberá descomprimir el archivo "gatekeeper.tgz" en el directorio /openh323/samples/, cambiar al directorio generado por el tgz y compilar usando las siguintes opciones.

- make opt - genera el ejecutable con traza.
- make debug - genera el ejecutable con información de debug para las herramientas de GNU/Linux como gdb.
- make both - genera los dos ejecutables anteriores.
- make notrace - genera el ejecutable sin opción a salida de archivo de traza.

El ejecutable es generado en el directorio /obj_x86_r para la opción opt y en el directorio /obj_x86_d para la opción debug.

Luego de la obtención del ejecutable, se deberá copiar éste junto al archivo de configuración a un mismo directorio , y modificarlo con las opciones necesarias según se detalla en el manual del administrador.

Índice de figuras

Índice de cuadros

Bibliografía

[H323] ITU-T Recommendation H.323. Packet-based multimedia communication systems.

[Q931] ITU-T Recommendation Q.931. ISDN user-network interface layer 3 specification for basic call control.

[H225] ITU-T Recommendation H.225.0. Call signalling protocols and media stream packetization for packet-based multimedia communication systems.

[ASN1] ITU-T Recommendation H.225.0. Annex H - H.225.0 Message Syntax (ASN.1)

[H235] ITU-T Recommendation H.235.

[H245] ITU-T Recommendation H.245. Control protocol for multimedia communication.

[CTSS] ITU-T Recommendation H.450-2. Call transfer supplementary service for H.323.

[CDSS] ITU-T Recommendation H.450-3. Call diversion supplementary service for H.323.

[MPL] http://www.mozilla.org/NPL/MPL-1.0.html

[VoxG] http://www.voxgratia.org/

[O323] http://www.openh323.org/

[T323] http://toncar.cz/openh323/tut/

[DOCP] http://docpp.sourceforge.net/

[DGEN] http://www.doxygen.org/

[I210] ITU-T Recommendation I.210. Principios de los servicios de telecomunicación soportados por una red digital de servicios integrados y medios para describirlos

[S323] Service architectures in H.323 and SIP - A comparison – Josef Glasmann, Wolfgang Kellerer, Harald Müller.

[SIP] RFC3261 - SIP: Session Initiation Protocol

[ETH] http://www.ethereal.com/