Apéndice A. Arquitectura de Openh323

A.1 Vista lógica

Se definen dos categorías en la vista lógica de la librería *OpenH323* según el siguiente diagrama:

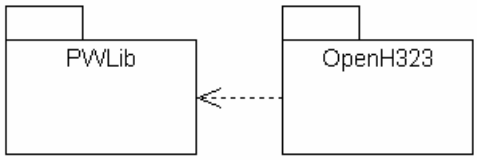

Figura A-1. Vista lógica OpenH323

A.1.1 Categoría PWLib

Se definen aquí todas las clases que forman parte de la librería PWLib. Es una librería de propósito general en la que se definen unas 300 clases que proporcionan las abstracciones del sistema operativo necesarias para que la aplicación sea portable.

A.1.2 Categoría OpenH323

En OpenH323 se incluyen clases que implementan los mensajes definidos en las recomendaciones H.225.0, H.245 y H.235, además de proporcionar el soporte para la transmisión de audio y vídeo.

Para tener una idea general de la librería *OpenH323* se puede simplificar la arquitectura centrándose en las clases más importantes, quedando un diagrama según muestra la figura A-2.

Dentro de la arquitectura H323, la clase fundamental de la jerarquía es *H323Endpoint*. Una aplicación basada en esta librería tendrá al menos una instancia de un descendiente de esta clase. El descendiente definido en la aplicación pondrá valores para algunos parámetros H323, el más importante será la tabla de capacidades que define los codec que la aplicación es capaz de manejar.

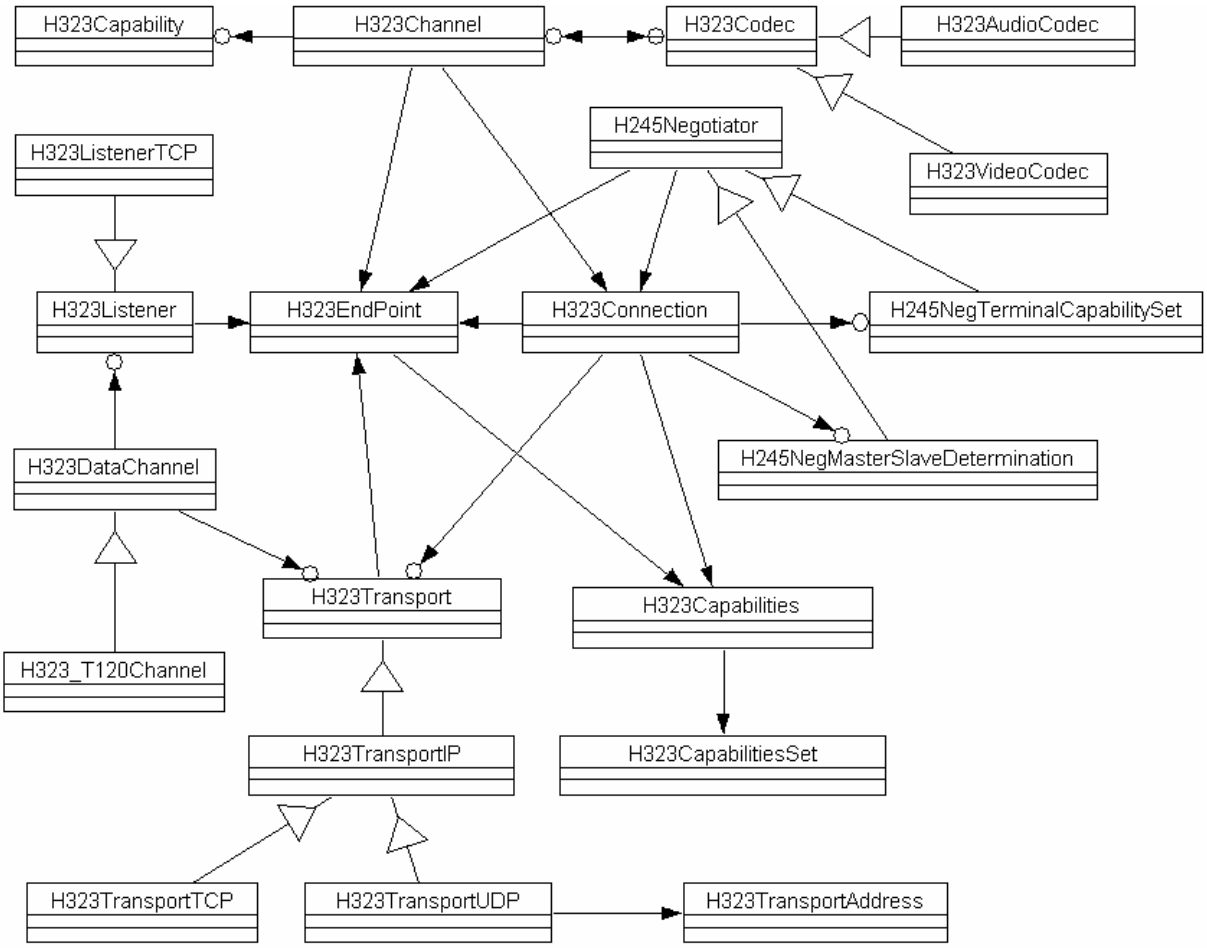

Figura A-2. Arquitectura de OpenH323

También se pueden crear en la aplicación instancias de uno o más descendientes de la clase *H323Listener*. Hay un descendiente de esta clase para cada protocolo que soporta. Por ejemplo, *H323ListenerIP* para uso en Internet. Cada *listener* genera una hebra (*thread*) que monitoriza su protocolo, y cuando detecta una nueva llamada crea una instancia del descendiente de la clase *H323Transport*, la cual, al igual que la clase H323Listener, tiene un descendiente para cada protocolo soportado, por ejemplo *H323TransportIP*.

Cuando llegan los primeros datos, esto es, la primera PDU (*Packet Data Unit*) a *H323Transport* usando los protocolos Q.931 y H.225, se crea una referencia, o *token*, de la llamada que identifica la conexión que se ha realizado. Esta conexión se representa por la clase *H323Connection*, que contiene toda la información para una conexión entre *endpoints* H.323. La instancia de *H323Endpoint* se mantiene al tanto de las conexiones activas. Si todavía no hay una conexión para el número de referencia de llamada, se creará una nueva y comienza la negociación de señales H.323.

A menudo una aplicación redefine la clase *H323Connection*, ya que se pueden sobrescribir un gran número de métodos virtuales. Estas funciones permiten a la aplicación tanto obtener información o modificar el comportamiento en varias fases de las negociaciones del protocolo. Por ejemplo, puede ser muy específico de la aplicación cuando una llamada entrante está en progreso y se debe avisar al usuario de la aplicación, lo que se puede hacer mediante un pitido leve hasta una ventana '*poppup*'.

La clase *H323Negotiator* se usa para mantener el estado y la funcionalidad de cada comando o variable definido por el protocolo H.245.

Durante algunas negociación H.245 se pueden crear canales lógicos, tanto para el *endpoint* remoto como para la aplicación local. La clase *H323Channel* representa esto. Un uso típico es abrir un *stream* de audio codificado. La clase *H323Channel* crearía un *H323Codec* usando la *H323Capability* que se acordó durante las negociaciones del protocolo.

A continuación se nombran algunas características de las clases más importantes.

A.2 H323EndPoint

Un *endpoint* puede tener cero o más instancias de la clase *H323listener* para crear conexiones. Cuando existe una conexión ésta se gestiona desde la instancia de la clase *endpoint*.

Lo principal que esta clase incluye son las capacidades de la aplicación, esto es, los codecs y protocolos que entenderá.

Los métodos principales de esta clase son:

- `H323EndPoint()`: Constructor de la clase.
- `void AddCapability(H323Capability * capability)`: Añade un codec a la tabla de capacidades. Asegura que el campo `assignedCapabilityNumber` en el codec sea único para todos los codecs que el *endpoint* tenga.
- `PINDEX SetCapability(PINDEX descriptorNum, PINDEX simultaneous, H323Capability * cap)`: Añade un nuevo codec a la lista de descriptores de

capacidades. La capacidad aquí especificada se añade automáticamente a la tabla de capacidades mediante la función `AddCapability`.

- `PINDEX AddAllCapabilities (PINDEX descriptorNum, PINDEX simultaneous, const PString & name)`: Añade todas las capacidades que se correspondan con la cadena `name`.

- `void RemoveCapabilities(const PStringArray & codecNames)`: Elimina capacidades de la tabla.

- `void ReorderCapabilities(const PStringArray & preferenceOrder)`: Reordena la tabla de capacidades colocando en el primer lugar los que se correspondan con la cadena `preferenceOrder`.

- `BOOL StartListener(H323Listener * listener)`: Añade un *listener* al *endpoint*, esto es, crea una nueva hebra para atender las llamadas.

- `BOOL RemoveListener(H323Listener * listener)`: Elimina un *listener* del *endpoint*.

- `virtual BOOL ClearCall (const PString & token, H323Connection::CallEndReason reason = H323Connection::EndedByLocalUser)`: Cierra una conexión actual.

- `virtual void ClearAllCalls (H323Connection::CallEndReason reason = H323Connection::EndedByLocalUser, BOOL wait = TRUE)`: Cierra todas las conexiones actuales.

- `H323Connection * FindConnectionWithLock(const PString & token)`: Devuelve la conexión identificada por `token`.

- `PStringList GetAllConnections()`: Devuelve la lista de las conexiones actuales del endpoint.

- `virtual BOOL OnIncomingCall(H323Connection & connection, const H323SignalPDU & setupPDU, H323SignalPDU & alertingPDU)`: función

que se ejecuta cuando llega una llamada para definir la respuesta antes de transmitir al otro *endpoint*. Por defecto sólo se devuelve TRUE.

- `virtual H323Connection::AnswerCallResponse OnAnswerCall (H323Connection & connection, const PString & callerName, const H323SignalPDU & setupPDU, H323SignalPDU & connectPDU)`: Función que define la respuesta a una llamada.

- `virtual void OnConnectionEstablished(H323Connection & connection,const PString & token)`: Función que se llama cuando se establece una conexión. Indica que se ha establecido una conexión a un *endpoint* remoto con un canal de control y cero o más canales lógicos.

- `virtual H323Connection * CreateConnection ( unsigned callReference, void * userData, H323Transport * transport, H323SignalPDU * setupPDU )`: Crea una nueva conexión.

A.3 H323Connection

Esta clase representa una conexión concreta entre dos *endpoints*. Se usan al menos dos hebras (*threads)*, una que vigila el canal de señalización, y otra el canal de control. Puede haber hebras adicionales para cada canal de datos que cree la hebra del protocolo del canal de control. Las funciones principales de esta clase son:

- `H323Connection(H323EndPoint & endpoint, unsigned callReference, unsigned options = 0)`: Constructor de la clase.

- `~H323Connection()`: Destructor de la clase.

- `BOOL Lock()`: Bloquea la conexión para uso exclusivo desde algún *thread* (gana el semáforo). Devuelve FALSE si no se puede bloquear debido a que la conexión ya se ha terminado.

- `void Unlock()`: Desbloquea la conexión.

- `virtual BOOL ClearCall (CallEndReason reason=EndedByLocalUser)`: termina la conexión con un *endpoint* remoto.

- `virtual AnswerCallResponse OnAnswerCall(const PString & callerName, const H323SignalPDU & setupPDU, H323SignalPDU & connectPDU)`: Una aplicación usa esta función para controlar la respuesta a las llamadas entrantes. Normalmente se usa para indicar la acción inmediata a tomar cuando se contesta una llamada. Los posibles valores que devuelve:

```
enum AnswerCallResponse {
  AnswerCallNow,
  AnswerCallDenied,
  AnswerCallPending,
  AnswerCallDeferred,
  AnswerCallAlertWithMedia,
  AnswerCallDeferredWithMedia,
  NumAnswerCallResponses
};
```

  Si se devuelve `AnswerCallNow` el protocolo H.323 continúa con la conexión. Si se devuelve `AnswerCallDenied` la conexión se aborta y se envía una PDU para indicar que se libere. Si se devuelve `AnswerCallPending` se envía una PDU de alerta y las negociaciones del protocolo se paran hasta que se llama a la función `AnsweringCall()`. Si se devuelve `AnswerCallDeferred` se procede igual que cuando se devuelve `AnswerCallPending` pero no se envía PDU. Si se devuelve `AnswerCallAlertWithMedia` es igual que en `AnswerCallPending` pero se inicializan los canales. Si se devuelve `AnswerCallDeferredWithMedia` es igual que en `AnswerCallDeferred` pero se inicializan los canales.

- `virtual BOOL OnStartLogicalChannel (H323Channel & channel)`: Función que se ejecuta cuando se inicia un canal lógico. Por defecto no hace nada y devuelve TRUE.

- `virtual BOOL OpenAudioChannel (BOOL isEncoding, unsigned bufferSize, H323AudioCodec & codec)`: Abre un canal para que lo use un

codec de audio. La clase `H323AudioCodec` usará esta función para abrir el canal y poder leer/escribir datos PCM. Por defecto llama a la función equivalente del *endpoint*.

- `virtual BOOL OpenVideoChannel(BOOL isEncoding, H323VideoCodec & codec)`: Abre una canal para que lo use un codec de vídeo. La clase `H323VideoCodec` usará esta función para abrir el canal y poder leer/escribir datos de imágenes.

A.4 H323Channel

Dentro del fichero *channels.cxx* se definen las siguientes clases:

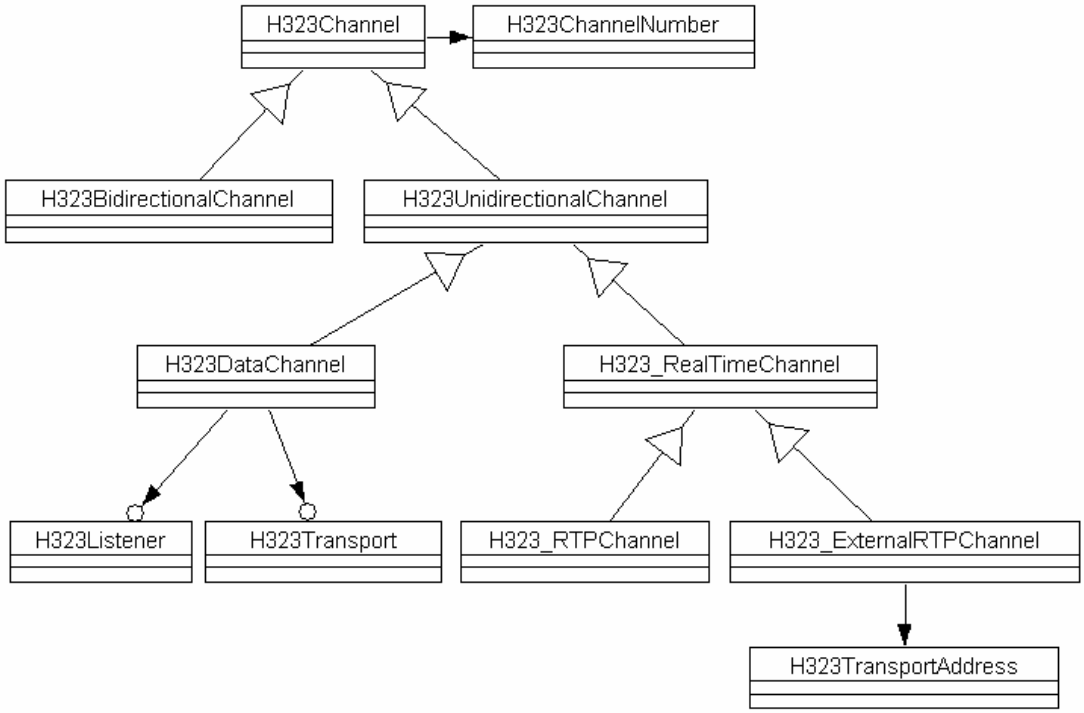

Figura A-3. Herencia de la clase H323Channel

La clase *H323Channel* describe un canal lógico entre dos *endpoints*. Entre sus funciones principales destacamos:

- `virtual BOOL Start() = 0`: función llamada cuando el canal puede transferir datos.

- `virtual void Receive() = 0`: función que gestiona la recepción de datos. Se llama por la hebra iniciada por la función `Start()` y normalmente consiste en un bucle: escribe al codec y lee desde el canal de transporte.

- `virtual void Transmit() = 0`: función que gestiona la transmisión de datos. Se llama desde la hebra iniciada por la función `Start()` y típicamente consiste en un bucle: lee desde el codec y escribe al canal de transporte.

Por otro lado, la clase `H323_RTPChannel` se define para encapsular el protocolo en tiempo real (RTP, *Real Time Protocol*) de IETF . Se modifican aquí las siguientes funciones:

- `virtual void Receive() = 0`: Si corresponde a una conexión en modo punto a punto envía directamente el audio y el vídeo al otro extremo de la conexión. Si no funciona normalmente

- `virtual void Transmit() = 0`: Sólo se llega a ejecutar cuando corresponde a modo multipunto.

A.5 Otras clases:

- **H323Capability:** Describe el interfaz para una capacidad del endpoint, normalmente un codec, usada para transferir datos vía los canales lógicos abiertos y gestionados por el canal de control H.323. Nótese que no se trata de una instancia del codec mismo, sino de una descripción de dicho codec.

- **H323Codec:** Implementación de una instancia de codec concreta usada para transferir datos vía los canales lógicos, abiertos y gestionados por el canal de control H.323.

- **H245Negotiator:** clase base para realizar negociaciones H.245.

- **H245NegTerminalCapabilitySet:** clase base para el intercambio de capacidades de una conexión H.245.

- **H245NegMasterSlaveDetermination:** determina el maestro y el esclavo en una conexión H.245.

- **H323Listener:** Describe el *listener* en el protocolo de transporte. Se ejecuta en una hebra separada.

- **H323ListenerTCP:** clase `H323Listener` para TCP/IP.

- **H323Transport:** esta clase describe un protocolo de transporte I/O.

- **H323TransportIP:** representa el caso particular de *H323Transport* para IP.

Apéndice B. Arquitectura MCU

B.1 Vista lógica

Se pueden definir las siguientes categorias:

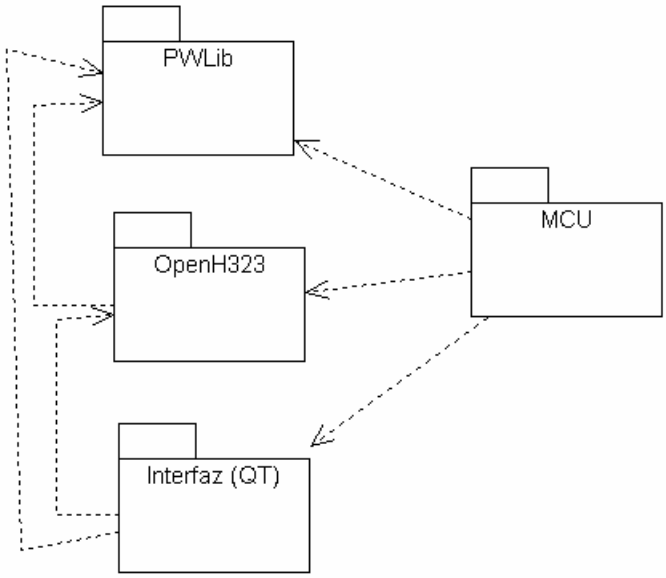

Figura B-1 Vista lógica MCU

La categoría principal es la correspondiente a la MCU, y es la que se comenta a continuación.

B.2 Categoría MCU

El modelo de objetos de esta categoría está representado en la figura 5-5.

En cualquier momento habrá N nodos conectados, consecuentemente habrá N copias de la clase *MyH323Connection*, que etiquetaremos como connA, connB, ..., connN.

Habrá N*(N-1) instancias de *AudioBuffers*.

Cada conexión tiene un diccionario que contiene (N-1) instancias de *AudioBuffers*. La conexión connI tendrá los *AudioBuffers* etiquetados abA, abB, ..(no abI) .., abM, abN.

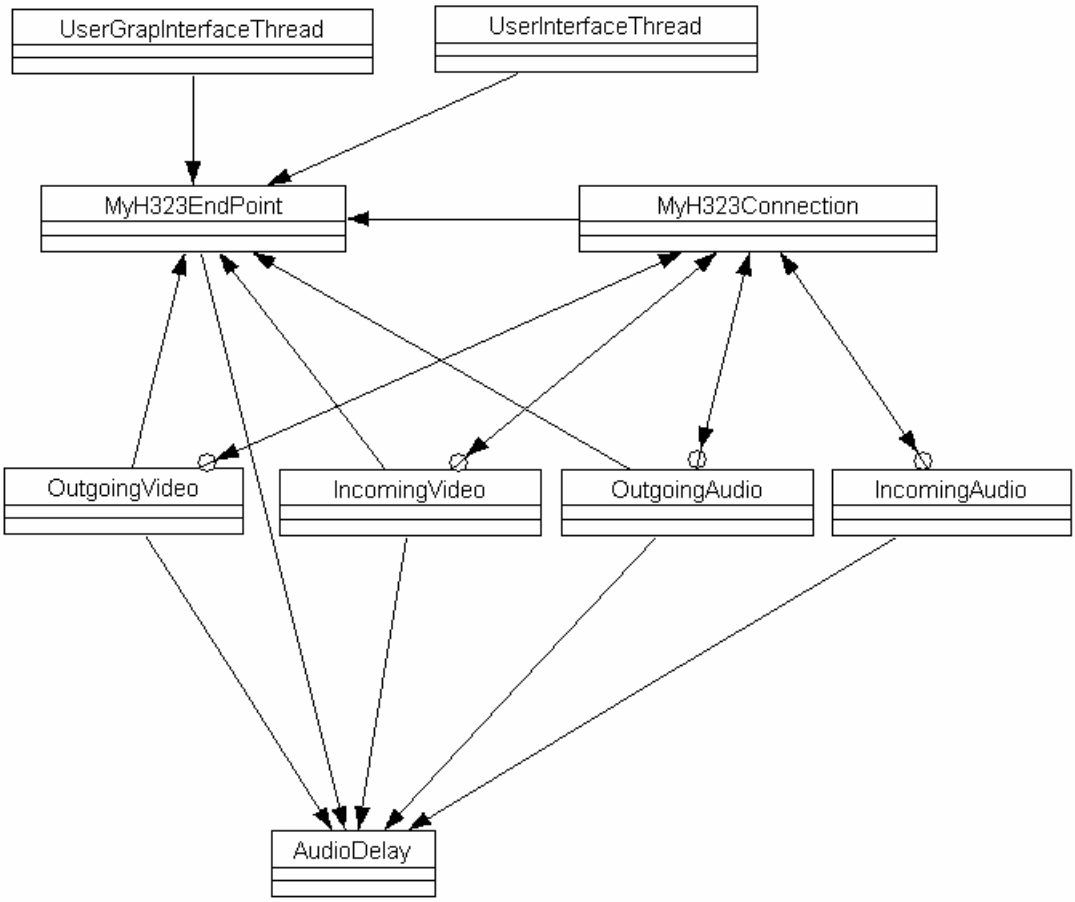

Figura B-2 Arquitectura de la MCU

Cuando llegan datos de audio a la MCU se siguen los siguientes pasos:

a) Los codecs de audio escriben en el canal *IncomingAudio*.

b) *IncomingAudio* envía datos a connI

c) ConnI escribe datos al *endpoint*

d) El *endpoint* copia los datos para connA, connB, ..(no para connI) .. connM, connN

Las conexiones listadas en el punto d) copian los datos al *AudioBuffer* correspondiente, así los datos de audio de la conexión connI se copian en abI para connA, en abI para connB, etc. De esta forma los datos de audio para connI se copian (N-1) veces.

Cuando el codec de audio solicita datos de audio para enviar, se dan los siguientes pasos:

a) El codec de audio solicita datos del canal *OutgoingAudio.*

b) El canal *OutgoingAudio* solicita datos de connI.

c) connI solicita datos del *endpoint.*

d) el método del *endpoint* (`MyH323EndPoint::ReadAudio`) encuentra entonces la conexión asociada con el codec de audio que ha solicitado los datos, en este caso connI.

e) El método `MyH323Connection::ReadAudio` es llamado por connI

f) `MyH323Connection::ReadAudio` combina los datos en cada uno de sus audiobuffers que son abA, abB, ..(no abI).., abM, abN.

Nótese que *OutgoingAudio* tiene un trabajo adicional, en el cual connI (en el paso c) podría evitar el *endpoint* e ir directamente a sus propios *AudioBuffers* y leer los datos. Este código no es seguro, ya que entonces el semáforo *memberMutex* no hace el acceso de protección (mediante el código de *outgoing*) a las conexiones.

En cuanto al vídeo, hay un *buffer* de vídeo en la clase *EndPoint*. Cuando llega un paquete de audio se mueve el marcador para esa conexión a lo alto de la lista. Si una conexión concreta está entre los 4 primeros (se habló recientemente), entonces cuando llega una trama de datos (*frame*) de vídeo la conexión escribe el *frame* a la sección correspondiente del *buffer* de vídeo (por ejemplo, esquina superior izquierda).

Cuando se solicita un vídeo se copia, en el *buffer* de vídeo, el *frame* de datos completo y se devuelve a la conexión.

B.2.1 MyH323EndPoint

Hereda de la clase `H323EndPoint` de la librería *OpenH323*

Propiedades:

- `StringListDict memberListDict`: lista de los nodos conectados.
- `StringListDict roomTiempos`: lista de la información de tiempos de las habitaciones creadas.

- `PMutex memberMutex`: previene el acceso múltiple a la lista de los nodos conectados.

- `BOOL hasMenu`: indica si se activa o no el menú.

- `BOOL hasInterfaz`: indica si se muestra o no la interfaz gráfica.

- `H323Capabilities capabilitiesPTOaPTO`: tabla de codecs punto a punto.

- `H323Capabilities capabilitiesMULTIPUNTO`: tabla de codecs multipunto.

- `StringListDict spokenListDict`: array que almacena los últimos que han hablado para decidir que cuatro imágenes se transmiten. Una lista por habitación.

- `VideoBufferDict videoBufferDict`: array de *videoBuffer*, uno por habitación.

- `StringListDict videoPosnDict`: array que indica dónde se muestra el vídeo.

Funciones:

- `MyH323EndPoint()`: constructor de la clase

- `virtual H323Connection * CreateConnection ( unsigned callReference,PString)`: crea una nueva connexion.

- `virtual void ListenForIncomingCalls()`: muestra un mensaje en pantalla: “Esperando llamadas …”

- `virtual void AwaitTermination()`: Función que comprueba los cambios registrados para saber si se termina la ejecución de la MCU, además de realizar otras comprobaciones relativas a los ficheros de eliminación de usuarios, y al número de usuarios que quedan en cada sala.

- `void AddMember (MyH323Connection * conn)`: añade una nueva conexión a una sala, si es la primera conexión para esa habitación, la crea.

- `void RemoveMember(MyH323Connection * conn)`: elimina una conexión de una habitación.

- `BOOL ReadAudio(const PString & token, void * buffer, PINDEX amount, PString roomID)`: Lee el audio correspondiente.

- `BOOL WriteAudio(const PString & token, const void * buffer, PINDEX amount, PString roomID)`: Escribe el audio correspondiente.

- `PINDEX FindTokensVideoPosn(const PString & thisToken, PString roomID)`: Devuelve la posición de vídeo que ocupa en la habitación `roomID` el usuario identificado por `thisToken`.

- `BOOL AddVideoPosnToken(const PString & thisToken, PString roomID)`: coloca al usuario representado por `thisToken` en la posición de vídeo que pueda.

- `void HandleUserInterface()`: menú de usuario.

- `void HandleUserGrapInterface()`: lanza la interfaz gráfica.

- `const H323Capabilities & GetCapabilitiesPTOaPTO()`: Devuelve la tabla de codecs punto a punto.

- `const H323Capabilities & GetCapabilitiesMULTIPUNTO()`: Devuelve la tabla de codecs multipunto.

- `void SetCapabilitiesPTOaPTO (H323Capabilities cap)`: Inicializa la tabla de codecs punto a punto con las capacidades `cap`.

- `void SetCapabilitiesMULTIPUNTO (H323Capabilities cap)`: Inicializa la tabla de codecs multipunto con las capacidades `cap`.

- `int NumUsuEnHabitacion(PString nombreHabitacion)`: Devuelve el número de usuarios en la habitación `nombreHabitacion`.

B.2.2 MyH323Connection

Hereda de la clase `H323Connection` de la librería *OpenH323*.

Propiedades:

- `MyH323EndPoint &ep`: *endpoint* al que pertenece.
- `PString roomID`: identificador de la habitación a la que pertenece.
- `PString audioTransmitCodecName, audioReceiveCodecName`: nombres de los codecs de audio usados por la conexión.
- PMutex audioMutex: semáforo para el acceso al audio.
- `PMutex videoMutex`: semáforo para el acceso al vídeo.
- PString videoTransmitCodecName, videoReceiveCodecName: nombre de los codecs de vídeo usados.
- `int modoConexion`: 0 = punto a punto, 1 = multipunto.
- `BOOL esAsesor`: indica si es un asesor.
- `IncomingAudio * incomingAudio`: canal para audio entrante.
- `OutgoingAudio * outgoingAudio`: canal para audio saliente.
- `IncomingVideo * incomingVideo`: canal para vídeo entrante.
- `OutgoingVideo * outgoingVideo`: canal para vídeo saliente.

Funciones:

- `MyH323Connection ( MyH323EndPoint &, unsigned, BOOL=TRUE)`: constructor de la clase.
- `BOOL OpenAudioChannel(BOOL, unsigned, H323AudioCodec & codec)`: abre un canal de audio.
- `void CleanUpOnCallEnd()`: cierra las instancias de `incomingAudio`, `outgoingAudio` y de `incomingVideo` y `outgoingVideo` si existen.
- `BOOL OpenVideoChannel( BOOL isEncoding, H323VideoCodec & codec)`: abre un canal de vídeo.

- `AnswerCallResponse OnAnswerCall(const PString &, const H323SignalPDU &, H323SignalPDU &)`: Define la respuesta a las llamadas entrantes.

- `BOOL OnStartLogicalChannel(H323Channel & channel)`: función que se ejecuta cuando se inicia un canal lógico. Muestra información que indica si es recibe o envía.

- `void AddMember (const PString & token)`: crea un nuevo *buffer* de audio para mantener audio entre la conexión identificada por `token` y la conexión actual.

- `void RemoveMember(const PString & token)`: elimina el *buffer* de audio de la conexión identificada por `token` en la conexión actual.

- `BOOL ReadAudio(const PString & token, void * buffer, PINDEX amount)`: lee audio.

- `BOOL WriteAudio(const PString & token, const void * buffer, PINDEX amount)`: escribe audio.

- `void EnableVideoReception(BOOL isOK)`: define si tendrá o no vídeo.

- `PString GetVideoTransmitCodecName()`: devuelve el nombre del codec de vídeo usado para transmitir.

- `PString GetVideoReceiveCodecName()`: devuelve el nombre del codec de vídeo usado para recibir.

- `PString GetRoomID()`: devuelve el identificador de la habitación a la que está conectado.

- `PString GetAudioTransmitCodecName()`: devuelve el nombre del codec de audio usado para transmitir.

- `PString GetAudioReceiveCodecName()`: devuelve el nombre del codec de audio usado para recibir.

- `int GetModoConexion()`: devuelve el modo de conexión usado.

- `void SetModoConexion(int modo)`: actualiza el modo de conexión a `modo`.
- `BOOL GetEsAsesor()`: devuelve si es o no un asesor.
- `void SetEsAsesor(BOOL a)`: actualiza el valor de `esAsesor`.

B.2.3 IncomingAudio

Hereda de la clase `PChannel` de la librería *OpenH323*.

Funciones

- `IncomingAudio(MyH323EndPoint & ep, MyH323Connection & conn)`: Constructor de la clase.
- `BOOL Write(const void * buffer, PINDEX amount)`: escribe audio en el canal.
- `BOOL Close()`: cierra el canal.

B.2.4 OutgoingAudio

Hereda de la clase `PChannel` de la librería *OpenH323*.

Funciones:

- `OutgoingAudio ( MyH323EndPoint & ep, MyH323Connection & conn)`: constructor de la clase.
- `BOOL Read(void * buffer, PINDEX amount)`: lee audio del canal.
- `BOOL Close()`: cierra el canal.

B.2.5 IncomingVideo

Hereda de la clase `PvideoChannel` de la librería *OpenH323*.

Funciones:

- `IncomingVideo(MyH323EndPoint & ep, MyH323Connection & conn)`: constructor de la clase.
- `~IncomingVideo()`: destructor de la clase.
- `BOOL Write(const void * buffer, PINDEX amount)`: escribe vídeo en el canal.
- `BOOL Close()`: cierra el canal.
- `void SetRenderFrameSize(int _width, int _height)`: define el tamaño para el vídeo.
- `BOOL IsOpen()`: Devuelve si está o no abierto el canal.
- `PINDEX GetGrabWidth()`: Devuelve el ancho del vídeo.
- `PINDEX GetGrabHeight()`: devuelve la altura actual seleccionada

B.2.6 OutgoingVideo

Hereda de la clase `PvideoChannel` de la librería *OpenH323*.

Funciones:

- `OutgoingVideo(MyH323EndPoint & ep, MyH323Connection & conn, int framesPerSec, BOOL videoLarge)`: constructor de la clase.
- `~OutgoingVideo()`: Destructor de la clase.
- `BOOL Close()`: cierra el canal.
- `BOOL Read(void * buffer, PINDEX amount)`: lee los datos de vídeo del canal.
- `void SetRenderFrameSize(int /*_width*/, int /*_height*/)`: Define el tamaño del vídeo.

- `BOOL IsOpen()`: Devuelve si está o no abierto.
- `BOOL IsGrabberOpen()`: Devuelve TRUE.
- `PINDEX  GetGrabWidth()`: Devuelve el ancho del vídeo.
- `PINDEX  GetGrabHeight()`: Devuelve la altura del vídeo.

B.2.7 AudioBuffer

Hereda de la clase `PObject`.

Propiedades:

- `BYTE *buffer`
- `PINDEX bufferLen`: Número de bytes sin leer en el *buffer*.
- `PINDEX bufferStart`: posición actual en el *buffer*.
- `PINDEX bufferSize`: Número total de bytes en el *buffer*.
- `PMutex audioBufferMutex`: Semáforo.

Funciones:

- `AudioBuffer()`: constructor de la clase.
- `~AudioBuffer()`: Destructor de la clase.
- `void Write(const BYTE * ptr, PINDEX amount)`: escribe en el *buffer*.
- `void Read(BYTE * ptr, PINDEX amount)`: lee del *buffer*.
- `void ReadAndMix(BYTE * ptr, PINDEX amount, PINDEX channels)`: realiza la lectura y mezcla del audio.
- `void Mix(BYTE * dst, const BYTE * src, PINDEX count, PINDEX channels)`: realiza la mezcla de audio.

B.2.8 VideoBuffer

Hereda de la clase `PObject`.

Propiedades:

- `BYTE * buffer`
- `PINDEX videoBufferSize`: Número total de bytes en el *buffer*, nunca cambia.
- `PMutex videoBufferMutex`: Semáforo.
- `int bufferFrameSize`: tamaño del frame.

Funciones:

- `VideoBuffer()`: Constructor de la clase.
- `~VideoBuffer()`: Destructor de la clase.
- `void Write( BYTE * ptr, PINDEX amount, PINDEX posn)`: escribe el vídeo en el *buffer*.
- `void Read(BYTE * ptr, PINDEX amount)`: lee el vídeo del *buffer*.
- `void Clear(PINDEX posn)`: reinicia el *buffer* en la posición `posn`.
- `void SetSize(int x, int y)`: Establece el tamaño del *buffer* de vídeo.

B.2.9 UserInterfaceThread

Hereda de la clase `PThread` de la librería *OpenH323*. Gestiona el menú de la MCU.

B.2.10 UserGrapInterfaceThread

Hereda de la clase `PThread` de la librería *OpenH323*. Lanza la interfaz gráfica de la MCU.

B.3 Categoría interfaz

Define las clases que implementan el interfaz gráfico de la MCU. La arquitectura se muestra en la figura

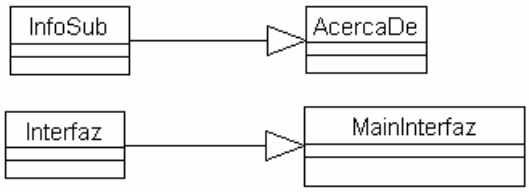

Figura B-3. Arquitectura Interfaz

Las clases `InfoSub` e `Interfaz` se crean a partir de las pantallas desarrolladas con la herramienta QTDesigner, de ellas heredan las clases `AcercaDe` y `MainInterfaz`, en las que se escribe el código asociado a cada evento de la pantalla.

B.3.1 AcercaDe

Sólo se define aquí la función `void InfoSub::bot_cerrar_pressed()` que cerrará la pantalla cuando se pulse el botón cerrar, además del constructor de la clase.

B.3.2 MainInterfaz

Se definen las siguientes funciones:

- `Interfaz::Interfaz() : MainInterfaz ( 0, "MainInterfaz")`: constructor de la clase.
- `void Interfaz::bot_AcercaDe_pressed()`: función que se ejecuta cuando se pulsa el botón “Acerca de”. Muestra la ventana de información.
- `void Interfaz::bot_Salir_pressed()`: función que se ejecuta cuado se pulsa el botón salir. Sale del interfaz gráfico.
- `void Interfaz::bot_Estadisticas_pressed()`: se ejecuta cuando se pulsa el botón de refresco de datos. Actualiza los datos que se muestran en pantalla.

- `void Interfaz::bot_cambiarNombreMCU_pressed()`: cambia el nombre con el que la MCU es identificada en los clientes.

- `void Interfaz::tabla_Habitaciones_currentChanged(int row, int col)`: función que se ejecuta cuando se cambia la fila seleccionada en la tabla de las habitaciones. Actualiza los datos de la tabla en la que se muestra la información de los usuarios.

- `void Interfaz::refrescoPantalla(int fila)`: función que refresca los datos que se muestran en pantalla.

Apéndice C. Recursos utilizados

C.1 Introducción

Para el desarrollo del proyecto se ha contado hasta con 6 ordenadores conectados en red, uno en el que está instalada la MCU y el resto se usaba con los programas para videoconferencia correspondientes, funcionando como asesores o clientes. En este capítulo se describen los recursos con los que se ha contado.

C.2 Recursos Hardware

El ordenador principal, en el que se ha desarrollado el trabajo, y, por tanto, el que tiene la MCU instalada, es un Pentium III a 733MHz, con tarjeta de red ethernet.

Todos los ordenadores forman parte de una red de área local a 100Mbps.

El resto de los ordenadores que se han usado para las pruebas se corresponden con las siguientes descripciones:

- Procesador Intel Pentium III, 731 MHz, 256MB de RAM.
- Procesador AMD Athlon (TM)XP1700+, 1’48GHz, 512MB de RAM.
- Procesador AMD Athlon (TM)XP1700+, 1’11GHz, 512MB de RAM.
- Procesador Intel Pentium III, 733MHz, 256 RAM.
- Procesador Intel Pentium III, 550MHz.

Todos con tarjeta de red, además todos los que no tienen el sistema operativo Linux están equipados con:

- Cámaras web.
- Tarjeta de sonido.
- Micrófono y altavoces.

C.3 Recursos Software

El ordenador de la MCU trabaja con el sistema operativo Linux RedHat 8. Además, este ordenador ha necesitado:

- Compilador g++
- *QTDesigner* y las librerías QT, para el desarrollo de la interfaz.

Por otro lado, los ordenadores que se usan con los programas clientes para las videoconferencias trabajan con los sistemas operativos Windows XP, Windows 98 y Linux RedHat 8. Como programas clientes se han utilizado:

- *NetMeeting*, principalmente, ya que así lo requería el proyecto CIMA.
- *OhPhone* (bajo Linux) y *OpenPhone* (bajo Windows), con las librerías *PWLib* y *OpenH323* correspondientes. Ambos de *OpenH323 Project*.

Por otro lado, para la documentación del proyecto se ha usado la herramienta CASE *Rational Rose C++ 4.0*.

Apéndice D. Instalación

D.1 Introducción

Se detallan aquí los pasos a seguir para instalar la MCU en Linux RedHat 8 o RedHat 9, ya que son en estos sistemas operativos en los que se llegó a probar.

D.2 Archivos

La instalación es muy simple, sólo se tendrán que copiar en el ordenador tanto las librerías *PWLib*, y *OpenH323*, así como los ejecutables de la MCU y del *Watchdog*, y definir las variables de entorno correspondientes.

Sólo será necesario copiar el árbol de directorios que se encuentra en el cd que acompaña a este trabajo, dentro de la carpeta `"/Instalacion",` a cualquier directorio del ordenador destino. Dentro hay cuatro carpetas: `pwlib`, `openh323`, `openmcu` y `watchdog`. No es necesario copiarlas dentro del mismo directorio, en función de dónde se copien se definirán las variables de entorno.

D.3 Variables de entorno

Habrá que definir las siguientes variables de entorno según los directorios en los que se hayan copiado los archivos:

```
PWLIBDIR=$HOME/…/pwlib
export PWLIBDIR
```

es la ruta dónde se ha copiado la carpeta `pwlib`.

```
OPENH323DIR=$HOME/…/openh323
export OPENH323DIR
```

indica el directorio en el cual se ha copiado la carpeta `openh323`.

```
LD_LIBRARY_PATH=$PWLIBDIR/lib:$OPENH323DIR/lib
export LD_LIBRARY_PATH
```

además, si se instala bajo Linux RedHat 9 también habrá que definir la variable LD_ASSUME_KERNEL según:

```
LD_ASSUME_KERNEL = 2.4.1
```

De esta forma, si se han copiado las carpetas del cd en el directorio `$HOME/MCU/`, se definen las variables de entorno según:

```
PWLIBDIR=$HOME/MCU/pwlib
export PWLIBDIR
OPENH323DIR=$HOME/MCU/openh323
export OPENH323DIR
LD_LIBRARY_PATH=$PWLIBDIR/lib:$OPENH323DIR/lib
export LD_LIBRARY_PATH
```

Apéndice E. Manual de Usuario

E.1 Introducción

El funcionamiento de la MCU es sencillo, una vez arrancada, según las opciones de configuración deseadas, consta de un simple menú mediante el que realizar algunas consultas, tal y como se verá a continuación. Como ya se ha visto, también dispone de un interfaz gráfico que facilita las consultas. Nos centramos aquí en el funcionamiento en modo texto.

E.2 Arranque de la MCU

Una vez instalada la MCU y las librerías necesarias tal y como se vio en el apéndice C, habrá dos formas de arrancar la MCU, desde el *WathcDog*, o directamente.

Si se desea arrancarla desde el *WatchDog* para tener un control del arranque y finalización de la MCU simplemente se va al directorio en el que se encuentra su ejecutable y le pasaremos dos parámetros, por un lado la ruta del ejecutable de la MCU, y en segundo lugar la ruta de los ficheros de acceso y eliminación:

```
Unix> ./wd /ruta_ejecutable_mcu/iopenmcu /ruta_ficheros
```

Si se desea arrancar la MCU directamente hay que ir al directorio en el que se encuentra su ejecutable y pasarle la ruta de los ficheros:

```
Unix> ./iopenmcu /ruta_ficheros
```

E.3 Opciones de configuración

La MCU se puede configurar según las siguientes opciones, las cuales las leerá del fichero denominado *opciones* que deberá encontrarse en el directorio desde el que se ejecute, aunque tiene fijadas por defecto algunas por si no se encuentra el fichero. Las opciones son las siguientes:

`-u –username str`	Define el nombre local de la MCU a *str*. Será el nombre que les aparezca a los usuarios en el *NetMeeting*.

`-g –gatekeeper` *`host`*	Especifica la situación del *Gatekeeper*.
`-n –no-gatekeeper`	Indica que no habrá *Gatekeeper*.
`--require-gatekeeper`	Termina si falla el *Gatekeeper*.
`-i ––interface ip`	Enlaza con un determinado interfaz.
`-j –jitter [min-]max`	Establece el *buffer* del *jitter* mínimo (opcional) y máximo (en milisegundos).
`--g711frames count`	Fija el número de *frames* en capacidades G.711 (por defecto 30).
`--gsmframes count`	Fija el número de *frames* en capacidades GSM (por defecto 4).
`-t ––trace`	Habilita los mensajes de traza. En función de las veces que se use dará más o menos detalle en estos mensajes. Esto es , `-t` dará menos información que `–ttt`.
`-o --output`	Fichero en el que se guardará la traza de salida, por defecto es *stderr*.
`--save`	Guarda las opciones en el fichero de configuración para las futuras ejecuciones de la MCU. Este fichero está en el directorio oculto `/.pwlib_config` dentro del directorio `$HOME`, se denomina `openmcu.ini` y al usar esta opción se añade la sección `[options]`.
`-v ––video`	Habilita el envío/recepción de vídeo para todas los usuarios.
`--videolarge`	Aumenta el tamaño del vídeo de normal (176x144) a grande (352x288).
`--videotxquality n`	Fija la calidad del vídeo que se envía. El valor por defecto 9. El rango de posibles valores varía entre 1 (buena calidad) a 31 (baja calidad).
`--videofill n`	Número de bloques actualizados por *frame*. Rango entre 1 y 99.

`--videotxfps n`	Máximo número de *frames* de vídeo transmitidos por segundo. Por defecto 10. Varía entre 1 y 30.
`--disable-menu`	Deshabilita el menú de la MCU.
`--disable-interfaz`	Deshabilita el interfaz gráfico.
`--audio-loopback name`	Los usuarios de la habitación *name* podrán escuchar su propia voz.
`-h –help`	Muestra esta lista de opciones.

Opciones por defecto : -n –v

E.4 Menú de la MCU

Una vez ejecutada la MCU se dispone de un menú mediante el que realizar algunas consultas, a menos que se haya especificado la opción `--disable-menu` en el fichero de opciones:

?	Imprime en pantalla la lista de opciones posibles del menú.
v	Indica, para cada habitación, la posición que ocupa cada conexión en el vídeo que se transmite.
s	Muestra un informe de las conexiones que existe en el momento de la consulta.
z	Escribe un mensaje en el fichero log (para depuración).
t	Informa de los tiempos para cada habitación que existe en el momento de la consulta.
i	Sólo devuelve una lista de las conexiones en cada habitación y el modo de conexión de cada una (punto a punto o multipunto).
p	Lista los codec remotos para cada conexión.

q, z Salir de la MCU.

Apéndice F. Acrónimos y definiciones

ACD: *Automatic Call Distributor*

***Buffer*:** Dispositivo de almacenamiento usado corrientemente para compensar diferencias en la velocidad de transmisión de datos, o temporización de eventos, cuando se transmite de un dispositivo a otro. Se usa también para eliminar el *jitter*.

CCITT: (*Consultative Committee for International Telegraph and Telephone*) Comité Consultivo Internacional de Telefonía y Telegrafía.

CIF: (*Common intermediate format*). Estándar de vídeo con 352 píxeles por línea y 288 líneas por imagen.

CIMA: Centro de Información Multimedia Avanzado.

***codec*:** Algoritmo software usado para comprimir/descomprimir señales de voz o audio. Se caracterizan por varios parámetros como la cantidad de bits, el tamaño de la trama *(fram*e), los retardos de proceso, etc. Algunos ejemplos de *codecs* típicos son G.711, G.723.1, G.729 o G.726.

gatekeeper (portero): Entidad de red H.323 que proporciona traducción de direcciones y controla el acceso a la red de los terminales, pasarelas y MCUs H.323. Puede proporcionar otros servicios como la localización de *gateways*.

gateway (pasarela): Dispositivo empleado para conectar redes que usan diferentes protocolos de comunicación de forma que la información puede pasar de una a otra. En VoIP existen dos tipos principales de pasarelas: la Pasarela de Medios *(Media Gateway*s), para la conversión de datos (voz), y la Pasarela de Señalización *(Signalling Gateway*), para convertir información de señalización.

GSM: (*Global System for Mobile Communications*) Es la tecnología telefónica móvil digital basada en TDMA predominante en Europa, aunque se usa en otras zonas del mundo. Se desarrolló en los años 80 y se desplegó en siete países europeos en 1992. La codificación de

audio del estándar GSM se utiliza en Telefonía IP y en la codificación de audio en ficheros WAV y AIFF.

IETF: (*Internet Engineering Task Force*) Grupo de trabajo de Ingeniería de Internet. Se reúne tres veces al año para fijar estándares técnicos sobre temas relacionados con Internet

IP: (*Internet Protocol*) Protocolo Internet. Es un protocolo de la capa de red que permite a las aplicaciones ejecutarse transparentemente sobre redes interconectadas.

ISDN: (*Integrated Services Data Network*) Red Digital de Servicios Integrados, RDSI.

ITU-T: (*International Telecommunications Union–Telecommunications*) Unión Internacional de Telecomunicaciones Telecomunicaciones.

***Jitter*:** Variación de retardo. Es un término que se refiere al nivel de variación de retado que introduce una red. Una red con variación 0 tarda exactamente lo mismo en transferir cada paquete de información, mientras que una red con variación de retardo alta tarda mucho más tiempo en entregar algunos paquetes que en entregar otros. La variación de retardo es importante cuando se envía audio o video, que deben llegar a intervalos regulares si se quieren evitar desajustes o sonidos ininteligibles.

MCU: (*Multipoint Control Unit*) Unidad de Control Multipunto.

Modem: (*Modulator-DEModulator*) Este término proviene de las palabras Modulador y Demodulador. Equipo que convierte señales digitales en analógicas y viceversa. Los modems se utilizan para enviar datos digitales a través de la red telefónica (PSTN), que normalmente es analógica. Un módem realiza una modulación del mensaje digital, convirtiéndolo en tonos que pueden ser enviados a través de la red telefónica. Al otro extremo, el demodulador del módem vuelve a convertir los tonos en una secuencia binaria (mensaje digital).

MPEG: (*Motion Picture Expert Group)* Grupo de expertos en imágenes en movimiento.

PCM: (*Pulse Code Modulation*) Convierte una señal analógica (sonido, voz normalmente) en digital para que pueda ser procesada por un dispositivo digital, normalmente un ordenador. Si, como ocurre en Telefonía IP, nos interesa comprimir el resultado para transmitirlo ocupando el menor ancho de banda posible, necesitaremos usar además un codec.

PDU: (*Packet Data Unit*). Unidad de datos del paquete.

PSTN: (*Public Switched Telephone Network*) Red telefónica convencional.

QCIF: (*Quarter common intermediate format*). Estándar de vídeo con 176 píxeles por línea, y 144 líneas por imagen.

QoS: (*Quality of Service*) Calidad de Servicio.

RAS: (*Registration, Authentication and Status*) Registro, Autentificación y Estado.

***Router*:** Encaminador, enrutador. Dispositivo que distribuye tráfico entre redes. La decisión sobre a dónde enviar los datos se realiza en base a información de nivel de red y tablas de direccionamiento.

RTP: (*Real Time Protocol*) El protocolo estándar en Internet para el transporte de datos en tiempo real, incluyendo audio y vídeo. Se utiliza prácticamente en todas las arquitecturas que hacen uso de VoIP, videoconferencia, multimedia bajo demanda y otras aplicaciones similares. Se trata de un protocolo que soporta identificación del contenido, reconstrucción temporal de los datos enviados y también detecta la pérdida de paquetes de datos.

TCP: (*Transmission Control Protocol*) Protocolo de la capa de transporte, asegura que los datos sean entregados, que lo que se recibe sea lo que se pretendía enviar y que los paquetes sean recibidos en el orden en el que fueron enviados. Se trata de un protocolo orientado a conexión

Telefonía IP: (*IP Telephony*) Tecnología para la transmisión de llamadas telefónicas ordinarias sobre Internet u otras redes de paquetes utilizando un PC, gateways y teléfonos estándar. En general, servicios de comunicación (voz, fax, aplicaciones de mensajes de voz) que son transportadas vía redes IP, Internet normalmente, en lugar de ser transportados vía la red telefónica convencional. Los pasos básicos que tienen lugar en una llamada a través de Internet son: conversión de la señal de voz analógica a formato digital y compresión de la señal a protocolo de Internet (IP) para su transmisión. En recepción se realiza el proceso inverso para poder recuperar de nuevo la señal de voz analógica.

UDP: (*User Datagram Protocol*) Protocolo de Datagramas de Usuario. Protocolo de la capa de transporte no orientado a conexión. No garantiza que los paquetes lleguen a su destino.

***VoIP, Voice over IP*:** (Voz sobre IP) Método de envío de voz por redes de conmutación de paquetes utilizando TCP/IP, tales como Internet.

Apéndice G. Bibliografía

[1]. Página de OpenH323: http://www.openh323.org

[2]. ITU-T Recommendation H.323: Packet-based Multimedia Communications Systems, Draft v4(11/2000).

[3]. Página de NetMeeting: http://www.microsoft.com/windows/netmeeting

[4]. "Thinking in C++": Bruce Eckel, Prentice Hall

[5]. H.323 Estándar de sistemas de comunicaciones multimedia sobre redes basadas en paquetes. Marcos Faúndez Zanuy, Escuela Universitaria Politécnica de Mataró.

[6]. Protocolos de señalización para el transporte de Voz sobre Redes IP. José Ignacio Moreno, Ignacio Soto, David Labarreiti, Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid.

[7]. Implementación de un sistema de teleasistencia con interfaz Web para redes IP: Proyecto CIMA. G.Gómez Paredes, F.J. González Cañete, E. Casilari, F.Sandoval, Departamento Tecnología Electrónica, E.T.S.I. Telecomunicación, Universidad de Málaga.

[8]. Desarrollo de un call center multiconferencia sobre IP. A.Triviño Cabrera, F.J. González Cañete, E. Casilari, F. Sandoval. Departamento Tecnología Electrónica, E.T.S.I. Telecomunicación, Universidad de Málaga.

[9]. "Unix Sistema V Versión 4": Kenneth H.Rosen, Richard R.Rosinski, James M.Farber, Douglas A.Host.

[10]. Página de *Equivalence Pty Itd:* http://www.equival.com

[11]. Página de Trolltech: http://www.trolltech.com

[12]. Documentación QT: http://doc.trolltech.com/3.2

[13]. MPL: http://www.mozilla.org/MPL/MPL-1.0.html

[14]. Página de *Quicknet*: http://www.quicknet.net/