TRABAJO FINAL DE CARRERA

TÍTULO: MCU para multiconferencias en alta definición

AUTOR: Javier López Rubio

DIRECTOR: Antoni Oller Arcas

FECHA: 1 de febrero de 2007

Titulo: MCU para multiconferencias en alta definición

Autor: Javier López Rubio

Director: Antoni Oller Arcas

Fecha: 1 de febrero de 2007

Resumen

En este trabajo se describen todos los pasos seguidos para el desarrollo de una MCU (Multipoint Control Unit), capaz de gestionar videoconferencias de alta definición. En el se incluyen las especificaciones y el diseño tanto a nivel de arquitectura como a nivel de aplicación.

La MCU resultante es el resultado de la investigación que la fundación i2Cat, encargó al grupo de investigación de banda ancha del departamento de ingeniería telemática. Este trabajo se encuentra incluido dentro del proyecto "Machine", proyecto en cual se han desarrollado diferentes clientes de videoconferencia, con los que la MCU deberá interactuar.

Los clientes desarrollados en el proyecto "Machine", utilizan el protocolo SIP para señalizar todas las comunicaciones que realicen, por lo tanto la MCU necesita una pila SIP para poder comunicarse con ellos.

Al tratarse de una aplicación independiente y gestionable remotamente, es necesario también el uso de un protocolo de control. Este protocolo de control se ha realizado a partir de la pila SIP ofrecida por la librería Jain Sip. Además del protocolo SIP estándar, se ha definido un nuevo tipo de mensaje, que nos permitirá ejecutar acciones de forma remota en la MCU.

Durante el desarrollo de éste trabajo se han desarrollado diferentes prototipos capaces de realizar el procesado multimedia de los flujos de alta definición. Finalmente se realizó un pequeño estudio en el que se muestran los resultados obtenidos con cada uno de ellos.

Title: MCU for high definition multiconferences

Author: Javier López Rubio

Director: Antoni Oller Arcas

Date: 1st February 2007

Overview

In this work are described all the steps followed in the development of an MCU (Multipoint Control Unit), that is able to manage high definition videconferences. Architecture and application design are also included in the documentation.

This MCU is the result of the investigation that the i2Cat foundation, ordered to the Broadband investigation group, which depends to the engineering Telematics department, this work is included within the project “Machine”. In the project many different videoconference clients have been developed, with which the MCU will have to interact.

The clients developed in the project “Machine”, use protocol SIP to signalize all the communications that they make, therefore the designed MCU needs a SIP stack to be able to communicate with them. When being an independent and remotely manageable application, the use of a control protocol is necessary.

This protocol of control has been developed based on the SIP stack offered by the Jain Sip API. In addition to standard protocol SIP, a new kind of message has been defined, that will allow us to execute remote actions in the MCU.

During the development of this work, different prototypes have been developed to process high resolution multimedia flows. Finally a comparison between the prototypes was made, and inside the document a report of the results is submitted.

ÍNDICE

INTRODUCCIÓN

Cada vez es más habitual el uso de aplicaciones que ofrecen al usuario la posibilidad de realizar videoconferencias en alta definición sobre Internet.

La fundación i2Cat, ha participado en el desarrollo de herramientas que permiten al usuario final establecer fácilmente, una videoconferencia entre sólo dos participantes.

Al realizar videoconferencias con más participantes, estas herramientas son insuficientes. En estos casos, se necesita utilizar un dispositivo intermedio que simplifique las comunicaciones entre todos los participantes, este dispositivo es conocido como MCU.

Una MCU es un elemento que se encarga de recibir el video de todos los participantes, y enviar hacia ellos el video del usuario que dispone del turno de palabra. La selección se realiza manualmente, por lo tanto, existe un participante encargado de seleccionar el participante que habla en cada momento.

El objetivo final de este proyecto es el de conseguir una MCU capaz de realizar multiconferencias en alta definición. Esta MCU debe tener un diseño modular que nos permita ofrecer escalabilidad, además de integrarse fácilmente en el escenario ya desarrollado por i2Cat.

Las herramientas ya desarrolladas utilizan SIP como protocolo de señalización, así que, nuestra MCU también deberá de incorporar una capa de señalización basada en este protocolo, con la que podrá interactuar con los demás participantes de la videoconferencia. Además dispondrá de una capa de control para que el usuario encargado de la realización pueda gestionar remotamente la MCU.

En el primer capítulo del documento se introducen los conceptos básicos y se enuncian los beneficios de usar este nuevo elemento en las multiconferencias de alta definición.

En el segundo capítulo se introduce la arquitectura de red que se ha utilizado y se definen todos los elementos que forman parte de ella.

El capítulo tercero es el dedicado al diseño de la aplicación, indicando los requisitos iniciales, las soluciones estudiadas, y finalmente el diseño interno de la aplicación.

En el cuarto capítulo se indican las tecnologías que se han utilizado, se definen las pruebas a realizar y finalmente los resultados obtenidos con las diferentes soluciones estudiadas.

Para finalizar se incluye un capítulo dedicado a la planificación del proyecto, y un capítulo final donde se describen todos los objetivos cumplidos, las conclusiones y los trabajos futuros a desarrollar sobre esta aplicación.

CAPÍTULO 1. CONCEPTOS BÁSICOS

Las videoconferencias en alta definición en la actualidad se reducen a experimentos realizados en universidades o empresas relacionadas con el sector, ya que los requerimientos tanto de los equipos, como de la red, son muy altos.

Las arquitecturas actuales no son útiles para realizar comunicaciones multipunto de alta definición, ya que el ancho de banda necesario es demasiado elevado. El objetivo de este proyecto es conseguir diseñar un elemento que nos permita realizar videoconferencias multipunto (inicialmente para tres participantes) que solucione los problemas de ineficiencia de las arquitecturas usadas hasta ahora.

Este proyecto propone una arquitectura que permite el desarrollo de sesiones de multiconferencia (en un entorno de alta definición) simplificando las exigencias al usuario final, tanto a nivel de conocimientos, como a nivel de red y dispositivos, los requisititos iniciales son definidos en el apartado 3.1.

En los siguientes apartados se indican unos conceptos básicos que son necesarios para la comprensión del proyecto.

1.1. MCU

Una MCU (Multipoint Control Unit) es un dispositivo, ya sea Hardware o Software, encargado de realizar conmutaciones de flujos de datos. Recibe la información de todos los extremos y selecciona el flujo que en ese momento deben recibir todos los extremos.

A continuación se mostraran las arquitecturas que habitualmente se usan para realizar multiconferencias y finalmente se indican los beneficios del uso de una *MCU*.

1.1.1. Arquitectura clásica de una multiconferencia

En la siguiente figura (Fig. 1.1) se muestran los participantes y los flujos de datos que se intercambian en una sesión de multiconferencia (formada por tres participantes) que usa una arquitectura clásica, es decir, sin utilizar ninguna técnica de replicación a nivel de red, y sin utilizar ningún punto neutro intermedio.

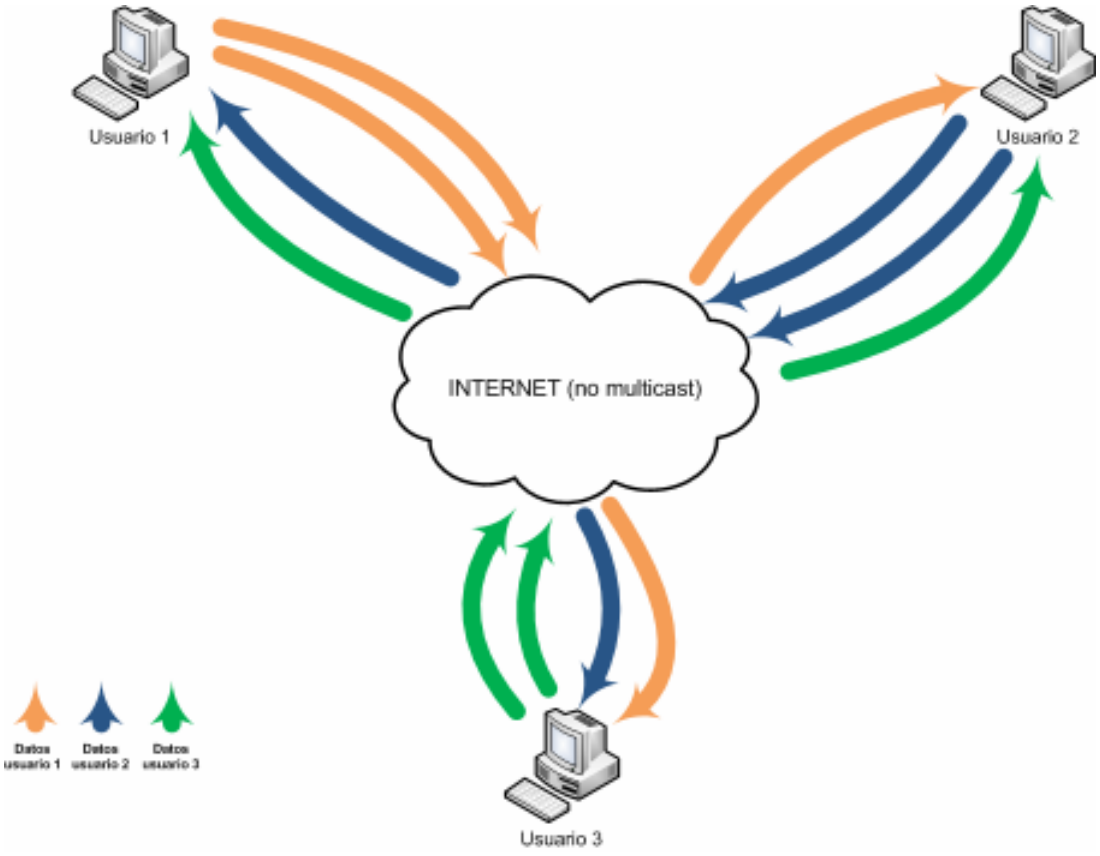


Fig. 1.1 Arquitectura clásica de una multiconferencia

En la imagen (Fig. 1.1) se observa, que cada usuario debe generar una copia del flujo de salida para cada participante de la multiconferencia (sin contarse el mismo), y debe ser capaz de recibir los flujos que provengan de los demás participantes.

Con este ejemplo podemos observar el principal problema de las multiconferencias sin puntos neutros, la gran cantidad de ancho de banda necesaria para su correcta realización. Si además de esto añadimos que la mayoría de conexiones que actualmente disponen los usuarios de Internet, son asimétricas, es decir, que la cantidad del ancho de banda no es igual en ambos canales (subida y bajada), llegamos a la conclusión de que esta arquitectura es muy ineficiente, ya que las calidades que se pueden ofrecer con ella son muy bajas.

Para solucionar este tipo de problemáticas (donde se envía un mismo flujo a varios participantes) se desarrollo el modo de multidifusión también conocido como “multicast”.

1.1.2. Arquitectura multiconferencia con soporte multicast

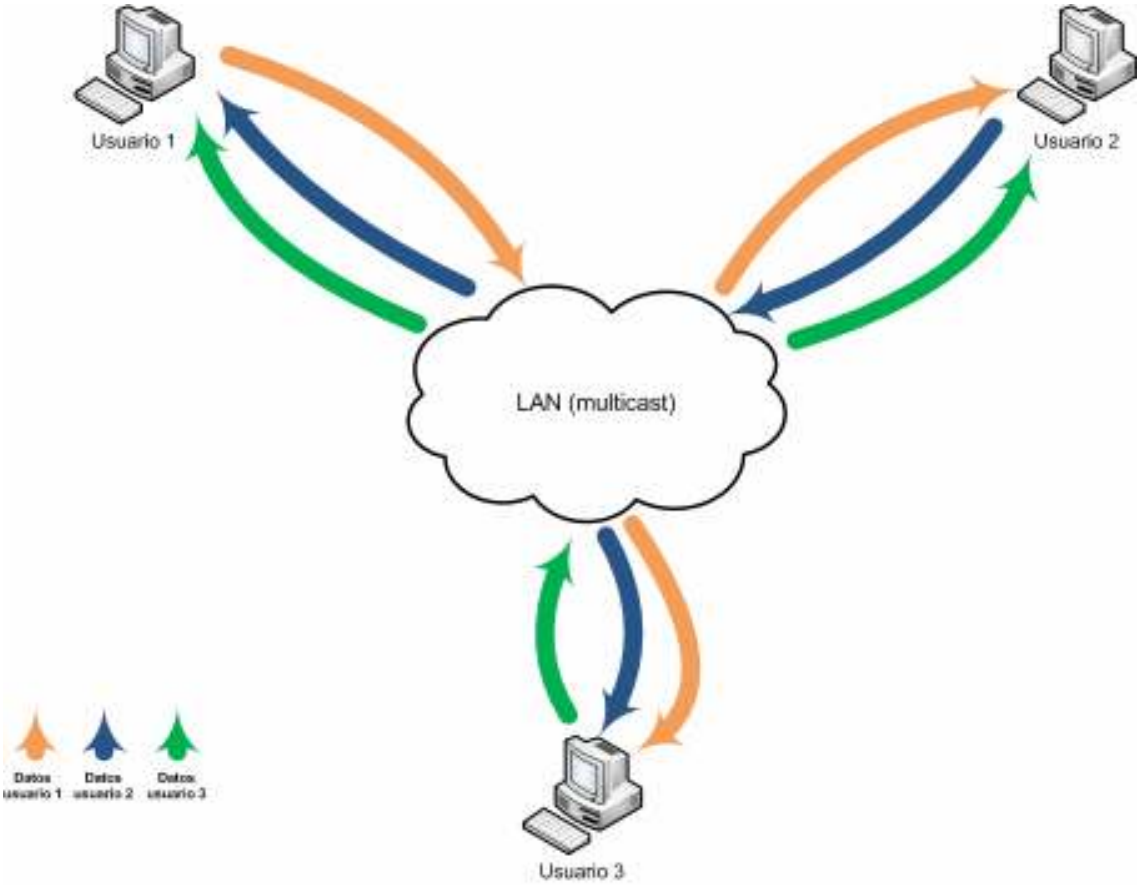


Fig. 1.2 Arquitectura de una multiconferencia con soporte multicast

Con este sistema se consigue que los participantes sólo deban enviar un único flujo, que recibirán todos los participantes de la multiconferencia. Esta seria una buena solución si realmente se pudiese llevar a la práctica. Ya que los proveedores de acceso a Internet (ISP), hoy por hoy no permiten que en sus redes se utilice el modo multicast.

Además, si deseamos optimizar el uso del ancho de banda al máximo, aún es necesario recibir en los extremos los flujos de todos los participantes, cosa que realmente es innecesaria, ya que nunca un usuario podrá mantener la atención en una conversación con más de una persona simultáneamente.

El diseño de una MCU partió de esta simple idea, permitiéndonos solucionar los problemas derivados de no poder utilizar el modo multicast y reduciendo al máximo el ancho de banda necesario.

Al utilizar una MCU existirá la limitación de visualizar a un único participante de forma simultánea; permitiendo de éste modo un fácil control y moderación de la sesión de multiconferencia.

1.1.3. Arquitectura multiconferencia con MCU

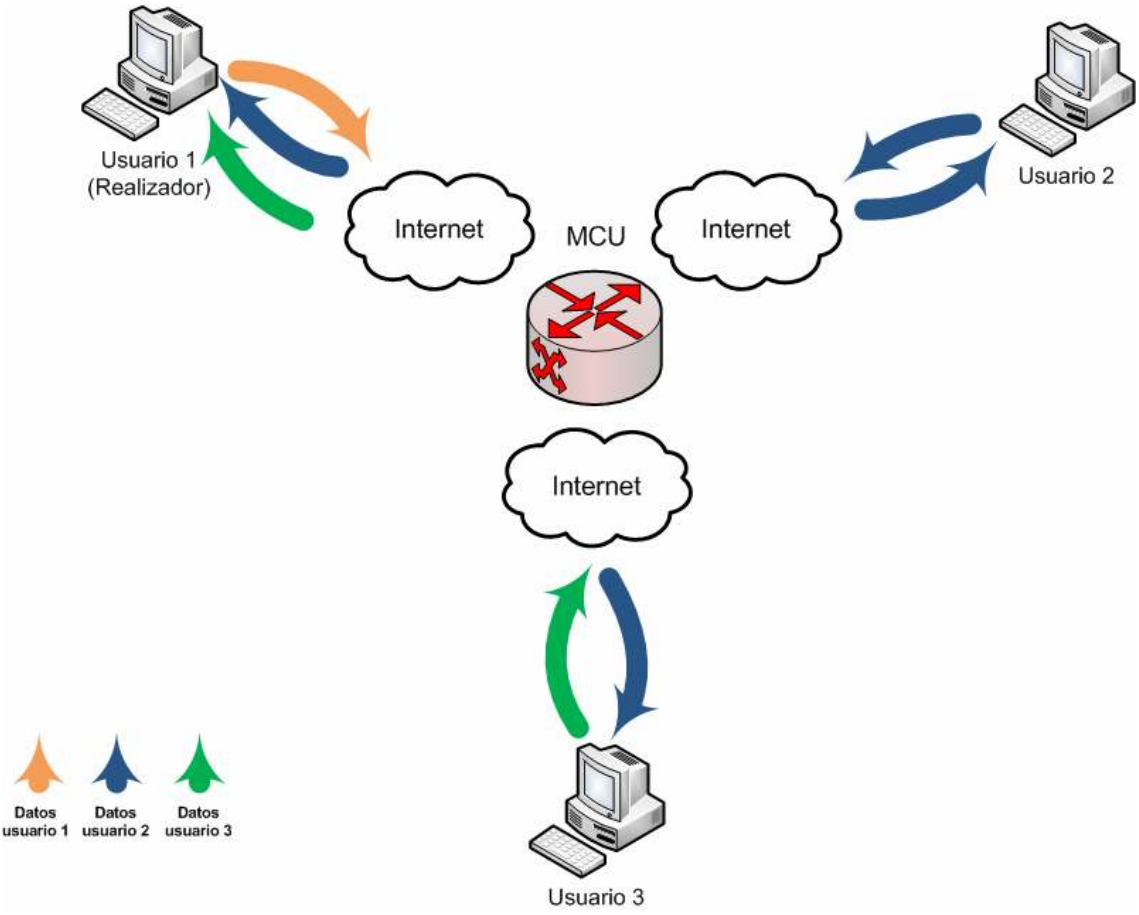


Fig. 1.3 Arquitectura de una multiconferencia con MCU

En este diagrama podemos observar como el ancho de banda necesario en los participantes de la conferencia se ha reducido a un único flujo tanto en recepción como en emisión. Como puntos débiles de esta arquitectura de comunicaciones podemos ver que tanto la *MCU*, como el cliente encargado de la realización sufrirán una mayor carga que los demás.

En el caso de una multiconferencia para tres participantes, el realizador recibirá tres flujos multimedia, y enviara un flujo, la *MCU* deberá soportar un flujo de entrada para cada participante (tres en total), y en salida deberá soportar cuatro flujos (dos enviados hacia el realizador, y dos flujos más con la señal ya conmutada.

1.2. Introducción a la alta definición

Cuando se habla de alta definición en video, nos referimos a un sistema capaz de dar mayor resolución (mayor definición) que los sistemas analógicos actuales, en proceso de extinción, por ejemplo PAL o NTSC.

Haciendo un poco de historia podremos observar que la primera aparición de un sistema de Alta Definición tal y como hoy lo comprendemos, fue a principios de los años 80. Japón desarrollo un sistema capaz de reproducir 1125 líneas con un refresco de 30 imágenes por segundos, éste sistema fue bautizado como NHK en Japón y SMPTE en Estados Unidos.

En la actualidad y con la aparición de la Televisión Digital, se ha apostado fuerte por la implantación de tecnologías de Alta Definición sobre soportes digitales. Han aparecido diferentes estándares, que en la actualidad emiten sobre diferentes medios de difusión, satélite mayoritariamente, y en un futuro cercano cable y televisión digital terrestre. Además también cabe la posibilidad de realizar la difusión a través de Internet (IPTV).

Estos estándares actuales de alta definición comenzaron a tratarse en el año 1987, cuando la FCC decidió iniciar un proceso para mejorar los sistemas de recepción de la señal de televisión, durante éste proceso se definieron diferentes estándares que abarcaban diferentes calidades y finalidades.

1.2.1. Estándares de alta definición digital

En la actualidad básicamente se usan tres denominaciones para los estándares digitales de alta definición [1] (720p, 1080i, 1080p).

Todos estos estándares asumen una relación de aspecto de las imágenes 16:9, y dependiendo de la situación geográfica se permite el uso de diferentes velocidades de refresco de imagen. Estas diferencias se mantienen respecto a los estándares analógicos PAL, NTSC, SECAM, en general se utilizan 24, 25, 30,50 y los 60 Hz.

- 720p, el numero 720 representa las 720 líneas de resolución vertical, y la letra “p”, indica que se realiza escaneo progresivo, de forma que la imagen se refresca al doble de velocidad que lo haría en el modo entrelazado. El escaneo progresivo mejora la resolución temporal al máximo posible. La resolución total de este estándar de 1280×720 píxeles.

- 1080i, es un estándar de resolución superior (1080 líneas de resolución vertical) por lo tanto puede mostrar mayor detalle y definición en la imagen, pero al no utilizar la técnica del escaneo progresivo, en momentos de mayor movimiento de imágenes puede dar peores resultados que el estándar 720p. Su resolución total es de 1920x1080 píxeles.

- 1080p, es el estándar más potente de video en alta definición, combina la alta resolución (1080 líneas de resolución vertical), con la técnica del escaneo progresivo (duplicando el refresco de las imágenes), consiguiendo la mayor calidad posible, dentro de los 3 estándares. La resolución total es de 1920x1080 píxeles.

Tabla 1.1 Resumen de características estándares HD

Estándar	Resolución Horizontal	Resolución Vertical	Refresco de imagen	Refresco de campo	Píxeles
720p	1280	720	24, 30, 60; 25, 50	---	0,92 Mpix
1080i	1920	1080	---	50, 60	1.04 Mpix
1080p	1920	1080	24, 30; 25	---	2,07 Mpix

1.2.2. Ventajas e inconvenientes de la HD en videoconferencias

El uso de estas tecnologías en videoconferencias basadas en soportes IP, nos lleva a valorar sus ventajas e inconvenientes.

Como ventajas, básicamente la mayor nitidez en la imagen recibida y/o emitida. Estas imágenes pueden ser utilizadas para diferentes finalidades; telemedicina, operaciones a distancia, tele asistencia, y cualquier tipo de actividad que requiera alta precisión/resolución.

Como inconvenientes a su implantación, aparecen varios que se pueden resumir en motivos económicos.

- Necesidad de renovar el equipamiento, desde cámaras hasta editores de video, ya que necesitaran mayor capacidad de proceso, finalizando por los proyectores o pantallas donde se visualizaran las imágenes transmitidas.
- Se necesita un gran ancho de banda disponible para poder transmitir y recibir la información que genera una transmisión de video en alta definición
- Necesidad de aumentar la capacidad de los dispositivos para poder procesar altas cantidades de información.
- Alto coste de todos los dispositivos que intervienen en una emisión de alta definición (codificadores del video).

1.2.3. Software HD para videoconferencias

En la fundación i2Cat básicamente se usan tres aplicaciones para la realización de videoconferencias en alta definición, DVTS conjuntamente con XDVShow para video conferencias de aproximadamente unos 20 o 30 Mbps y Ultragrid para videoconferencias entorno al Gbps.

1.2.3.1. DVTS

El software DVTS [2], nos permite capturar los paquetes DV generados por una videocámara convencional a través del puerto Firewire, añadirles cabeceras IP y retransmitirlos a través de la red. Con éste sistema se pueden realizar transmisiones de hasta 30 Mbps ofreciendo un sistema de videoconferencia de alta calidad destinado a redes de alta velocidad. Esta utilidad se usa conjuntamente con XDVShow para realizar sesiones de multiconferencia.

1.2.3.2. XDVShow

XDVShow [3] es una utilidad encargada de manejar diferentes tipos de flujos de video de tipo DV. Puede obtenerlos de varias fuentes, como son a través de la interfaz de red del equipo, o directamente de una interfaz IEEE 1394 (Firewire). Esta utilidad es capaz de reproducir estos flujos de video, independientemente de que interfaz por la que se capture el flujo, ya sea de la interfaz de red o directamente del puerto Firewire.

1.2.3.3. Ultragrid

El proyecto Ultragrid [4] es una colaboración entre varias universidades (Information Sciences Institute at the University of Southern California y Department of Computing Science at the University of Glasgow). Este software ha sido adoptado por varios otros organismos para realizar sus estudios sobre HDTV.

El software de Ultragrid se encarga de convertir señales de alta definición extraídas de las videocámaras, y convertirlas en paquetes RTP/UDP/IP y así poder ser distribuidas en redes de alta capacidad. La arquitectura de éste software ha sido pensada para reducir al máximo el retraso en las imágenes.

Básicamente se usan dos modos de funcionamiento, en ambos no existe compresión alguna del video, con lo cual los anchos de banda necesarios para su uso son muy elevados. El primer modo trabaja en el límite de 1 Gbps (para facilitar el uso en entornos donde no sea posible conseguir velocidades superiores), y el segundo modo ya supera los 1,2 Gbps, en éste modo Ultragrid ofrece todas sus posibilidades sin ningún tipo de limitación en la calidad.

1.3. Introducción al protocolo SIP

SIP (Session Initiation Protocol) se encuentra definido en el RFC[1] 3261 [5] y es un protocolo que proporciona herramientas para la creación, modificación y finalización de sesiones. Normalmente las sesiones son llamadas entre dos puntos y éstas se identifican por un call-ID. El Call-ID es un identificador de

[1] Request For Comments (Véase el anexo de acrónimos)

sesión que se crea mediante la dirección de origen, la de destino y otros parámetros de la sesión.

SIP proporciona el establecimiento de una sesión entre un terminal origen y un terminal destino. También permite poder localizar el destino, incluyendo mapeos de nombres, resolución de direcciones y redirección de destinatarios. Otra utilidad es la de determinar las capacidades del terminal de destino; para éste fin se suele utilizar otro protocolo, el protocolo SDP[2].

1.3.1. Mensajes SIP

En esta sección comentaremos las cabeceras comunes genéricas de un mensaje SIP y los tipos de mensaje SIP más habituales en una sesión.

Aquí tenemos un ejemplo de mensaje SIP:

```
INVITE sip:user1@i2cat.upc.net SIP/2.0
Via: SIP/2.0/UDP ser.i2cat.upc.net;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: User 1 <sip:user1@i2cat.upc.net>
From: User 2 <sip:user2@i2cat.upc.net>;tag=1928301774
Call-ID: a84b4c76e66710@ser.i2cat.upc.net
CSeq: 314159 INVITE
Contact: <sip:user1@i2cat.upc.net>
Content-Type: application/sdp
Content-Length: 142
```

Fig. 1.4 Mensaje SIP

En la primera línea mensaje se observa el tipo de mensaje, quien ha generado el mensaje, y la versión del protocolo SIP que utiliza ese cliente, en éste caso se trata de un mensaje **INVITE** (Inicio de una sesión).

La cabecera **Via**, indica por todos los enrutadores SIP que ha pasado el mensaje, en éste caso por un único enrutador.

Max-Forwards, indica el numero de saltos que la petición puede hacer en su camino hacia el destino, cada vez que se realiza un salto esta cabecera se decrementa en una unidad.

To, indica el nombre del cliente hacia el que se envía la petición y la URI asociada a ese nombre.

From, es similar a la cabecera To, pero además añaden un campo “tag” que es usado para discriminar al cliente que realiza la llamada.

Call-ID, es uno de los campos más importantes del mensaje ya que es el identificador único de esa sesión.

[2] Session Description Protocol (Véase el anexo de acrónimos)

CSeq, Indica el numero de intercambio de ese dialogo, cada vez que se realiza un nuevo intercambio de mensajes se actualiza.

Contact, indica la URI directa para contactar con el cliente que género el mensaje.

Content-Type, indica el tipo de contenido del BODY del mensaje.

Content-Length, indica la longitud del BODY del mensaje.

Dentro del protocolo SIP se definen gran cantidad de mensajes, en esta introducción solo comentaremos los de uso más habitual.

REGISTER, es un mensaje que se envía hacia el SIP Proxy indicándole que nos queremos registrar en el.

INVITE, mensaje de inicio de sesion, que el usuario que desea iniciar la session envía hacia el destinatario éste mensaje.

100 TRYING, mensaje de retorno de estado, que indica que se esta intentando procesar la petición.

180 RINGING, al igual que el anterior se trata de un mensaje de retorno de estado, y en éste caso indica que el mensaje ha sido recibido por el usuario y se esta a la espera de que realice alguna acción.

200 OK, también es otro mensaje de retorno del estado, que indica la aceptación del mensaje.

Existen más mensajes, la totalidad de ellos se encuentran definidos en el RFC 3261 del protocolo SIP.

1.3.2. Dialogo SIP

Un concepto importante dentro del protocolo SIP, es el concepto de Dialogo entre participantes. Utilizaremos un ejemplo para mostrarlo.

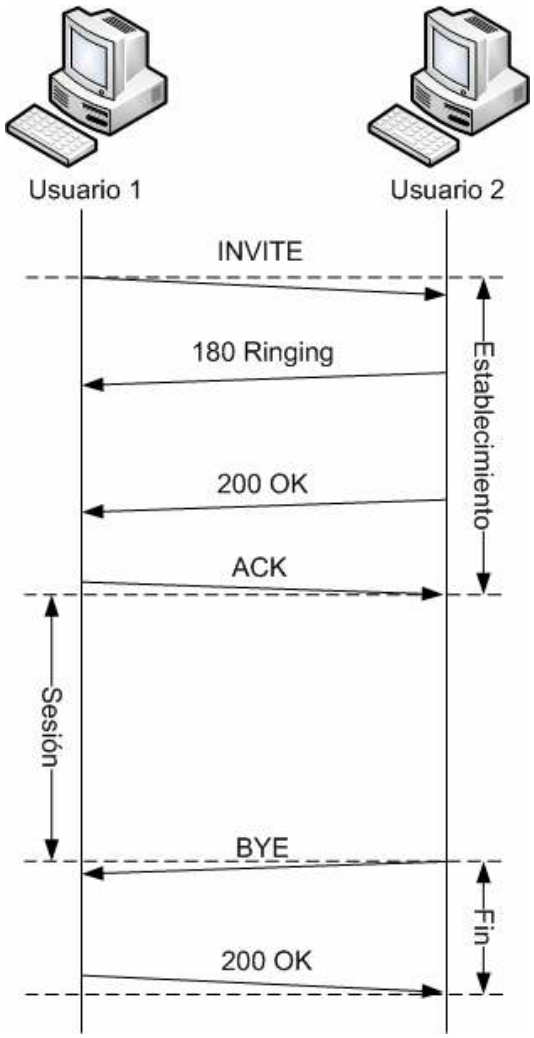


Fig. 1.5 Intercambio de mensajes SIP

- **INVITE:** El usuario uno (a partir de ahora USER1) realiza una petición de inicio de sesión al usuario dos (a partir de ahora USER2).
- **TRYING – 100 / RINGING – 180:** En cuanto el usuario recibe la petición (INVITE), éste crea una respuesta a este mensaje del tipo TRYING, indicando que va intentar comunicar al cliente que tiene una llamada entrante. En cuanto el cliente es notificado (y por lo tanto el usuario recibe un aviso ya sea sonoro o visual) el USER2 envía una respuesta del tipo RINGING hacia el USER1.
- **200 OK:** En el momento que el USER2 decide aceptar la llamada, el cliente enviara un mensaje del tipo 200 OK hacia el USER1, para indicar que la sesión puede ser establecida.
- **ACK:** Una vez recibido el 200 OK por parte del USER2, el USER1 debe confirmar el dialogo respondiendo con un mensaje de confirmación ACK.
- **BYE:** El USER2 decide finalizar la comunicación, para ello envía una petición de tipo BYE, hacia el USER1.
- **200 OK:** El USER1, confirma la finalización de esta sesión y ambos usuarios liberan los recursos ocupados.

1.3.3. SIP como protocolo de señalización versátil

En su RFC, SIP es definido como un protocolo de capa de aplicación, capaz de crear, modificar y finalizar sesiones de uno o más participantes, y estas sesiones las limita a llamadas telefónicas, distribución de multimedia, y conferencias.

Pero el tiempo ha demostrado que con pequeñas variaciones éste protocolo puede ser útil para más finalidades.

- **HomeSip** [6], es un proyecto desarrollado por un conjunto de empresas de electrónica y universidades, para utilizar SIP como protocolo de control de una vivienda domótica. Se han presentado diferentes drafts para la estandarización de sus variantes SIP, pero nunca se han llegado a aceptar.

- **SIMPLE** (Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions) [7], se trata de una extensión para permitir el uso del protocolo SIP como protocolo de mensajería instantánea. Se está desarrollando desde un grupo de trabajo de la IETF. Al igual que HomeSip han presentado diferentes drafts (algunos ya aprobados), y se esta a la espera de aceptación del ultimo draft presentado.

CAPÍTULO 2. ARQUITECTURA DE RED

Tomando de referencia la arquitectura ya utilizada por i2Cat, se incorporo a esta la nueva MCU.

La única modificación que se realizó fue la de añadir una capa de control para permitir la gestión remota de la MCU.

2.1. Participantes en una multiconferencia

Los elementos básicos que participaran en una multiconferencia con nuestra MCU son:

- Usuarios (Fig 2.1): cada uno con su cámara, y software necesario, para captura/recepción de video, más un pequeño cliente SIP capaz de señalizar todas las comunicaciones que se deseen realizar. Dentro de los usuarios, existirá un usuario encargado de la realización del evento. Dicho usuario dispondrá de unos privilegios para poder controlar el desarrollo de la multiconferencia en todo momento.

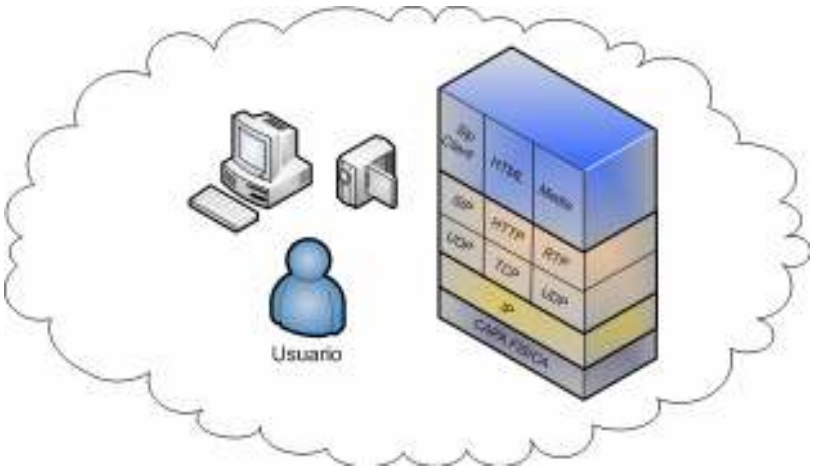


Fig. 2.1 Usuario

- La MCU (Fig 2.2), que será la encargada de conmutar los flujos enviados hacia ella por los participantes, al igual que un usuario, también dispone de una pila SIP, para comunicarse con los demás participantes.

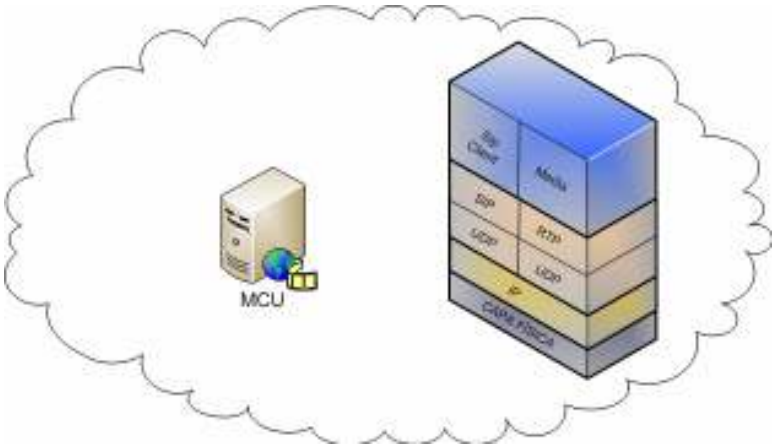


Fig. 2.2 MCU

- Un SIP Proxy, donde a través del protocolo SIP se registraran, toda la señalización SIP es enviada a través de el y se encargara de reenviar estos mensajes a su destinatario. Además de éste SIP Proxy es necesario un *agente de presencia*, que será el encargado de informar sobre estado y las capacidades de los demás clientes, por ejemplo, si un usuario está conectado o desconectado y las capacidades multimedia de dicho usuario.

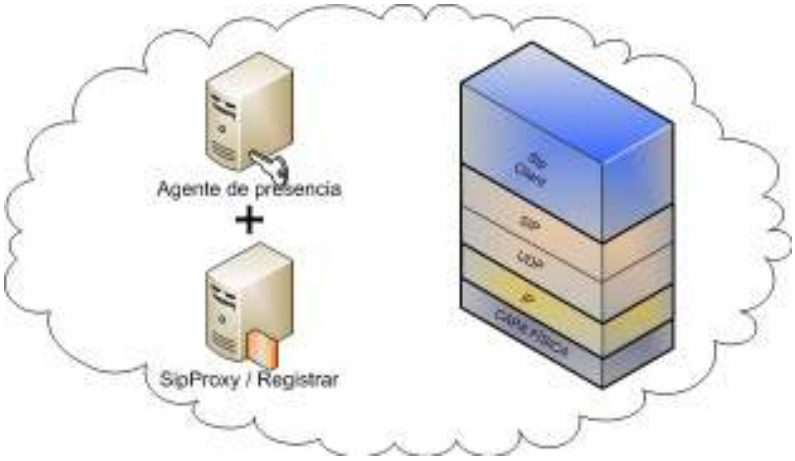


Fig. 2.3 Agente de presencia y SIP Proxy

Después de enunciar todos los participantes, la arquitectura resultante para la realización de un evento de multiconferencia, sería la siguiente:

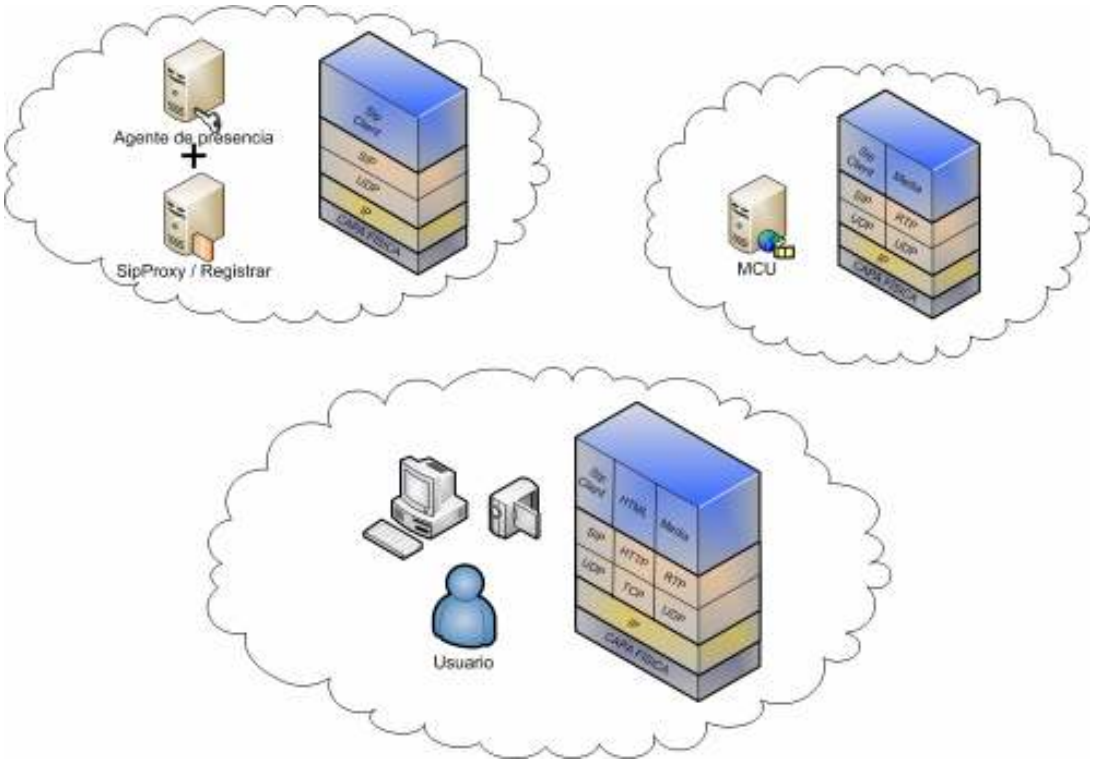


Fig. 2.4 Arquitectura completa

2.2. Relaciones entre participantes

En esta sección, explicaremos a un nivel básico las interacciones que existen entre los diferentes participantes que intervienen en una multiconferencia donde forma parte nuestra MCU.

Todos los clientes SIP (incluida la propia *MCU*) interactúan a través de mensajes SIP con el *SIP Proxy*, éste es el encargado de reenviar el mensaje al destinatario final. Todos los clientes al iniciarse deben enviar un mensaje SIP del tipo REGISTER hacia el *SIP Proxy*.

Una vez registrados todos los participantes, realizan una suscripción con el agente de presencia, el cual les notificara la lista de usuarios subscritos, y cualquier modificación del estado de los usuarios.

En este momento todos los participantes podrían realizar comunicaciones “peer to peer”[3] entre ellos sin necesidad de participar la *MCU*. En caso de que se hubiese programado una sesión de multiconferencia, el siguiente paso a realizar sería, que todos los participantes en el evento se subscribiesen a la *MCU*, de forma similar a la suscripción realizada con el agente de presencia. En el momento que todos los usuarios estén subscritos con la *MCU*, ésta realizará las acciones necesarias para iniciar la multiconferencia.

[3] Comunicaciones punto a punto entre participantes.

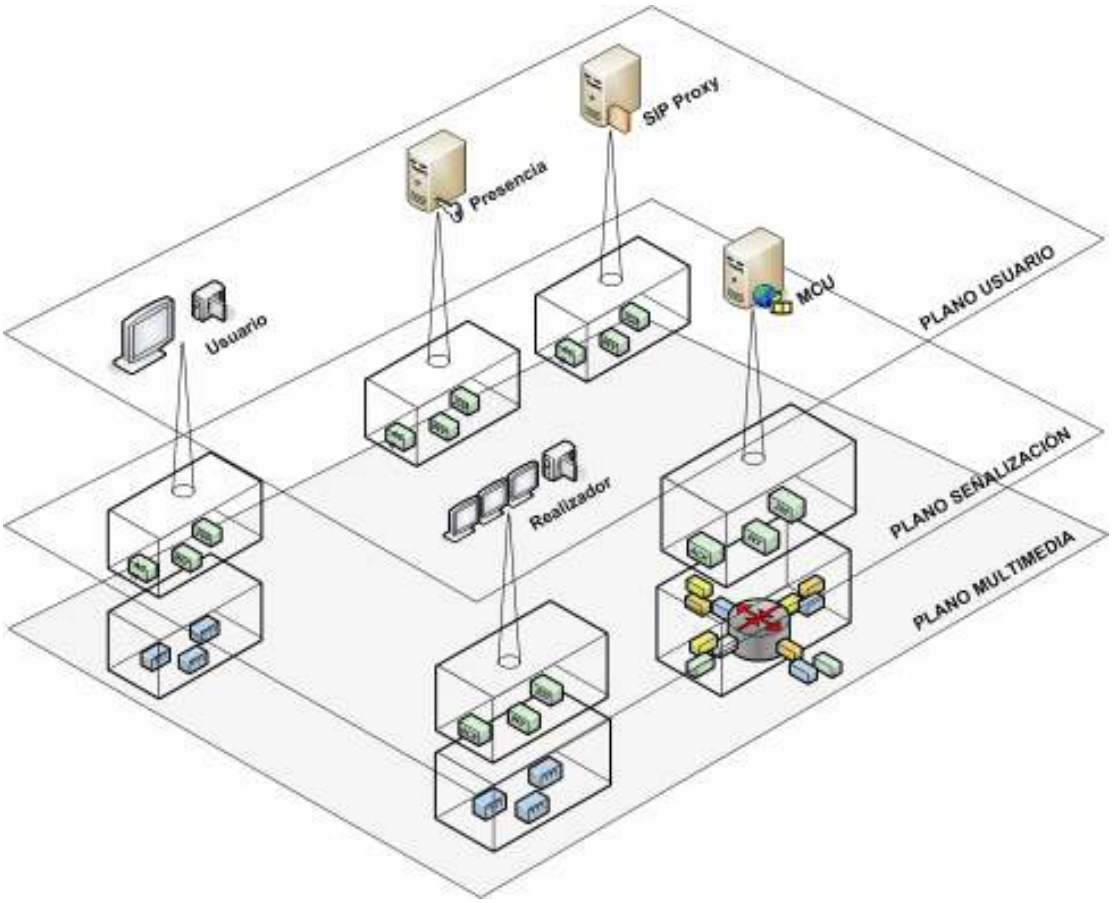


Fig. 2.5 Participantes y planos

Al iniciarse la multiconferencia, los participantes recibirán un mensaje SIP, generado por la *MCU*, invitándoles a iniciar la sesión, en estos momentos, iniciarán la aplicación encargada de la capa multimedia, y comenzaran a enviar los flujos de video/audio hacia la *MCU*, la cual procesará y realizará las conmutaciones necesarias durante el desarrollo de multiconferencia.

Antes de seguir avanzando en aspectos más técnicos de la *MCU*, se presenta la necesidad de diferenciar las comunicaciones, desde un punto de vista basado en la división en planos.

2.2.1. División en planos

Para facilitar la comprensión de las interacciones entre los participantes, es importante diferenciar las interacciones en el plano de señalización (básicamente los mensajes SIP que se intercambian), las interacciones a nivel multimedia (Flujos RTP[4] / UDP[5] que se envían, reenvían y conmutan) y finalmente el plano de control de la aplicación.

[4] Real Time Protocol (Véase el anexo de acrónimos)
[5] User Datagram Protocol (Véase el anexo de acrónimos)

Plano multimedia

Como plano multimedia se entiende, a toda la información que se genera, al realizar algún tipo de comunicación, siempre y cuando esta información transmitida sea de tipo multimedia.

En nuestro caso, el plano multimedia se reduce a todos los flujos que son enviados entre los participantes de una multiconferencia.

Plano de señalización

El plano de señalización se define como la información que se intercambiada entre órganos de la misma red, con la finalidad de controlar y gestionar los servicios ofrecidos. Esta información es generada y administrada por la propia red.

Para nosotros el plano de señalización se limita a los mensajes de tipo SIP, que se intercambian entre los diferentes clientes y servidores, para la creación de una sesión de multiconferencia.

Plano de control

El plano de control, es toda la información que se intercambia entre los equipos, para poder realizar gestiones de forma remota. En el caso de la *MCU*, el control estará formado por los mensajes que el realizador envíe hacia la *MCU* para su administración remota.

En la imagen (página siguiente, Fig. 2.6) se observan las interacciones entre los participantes en todos los planos:

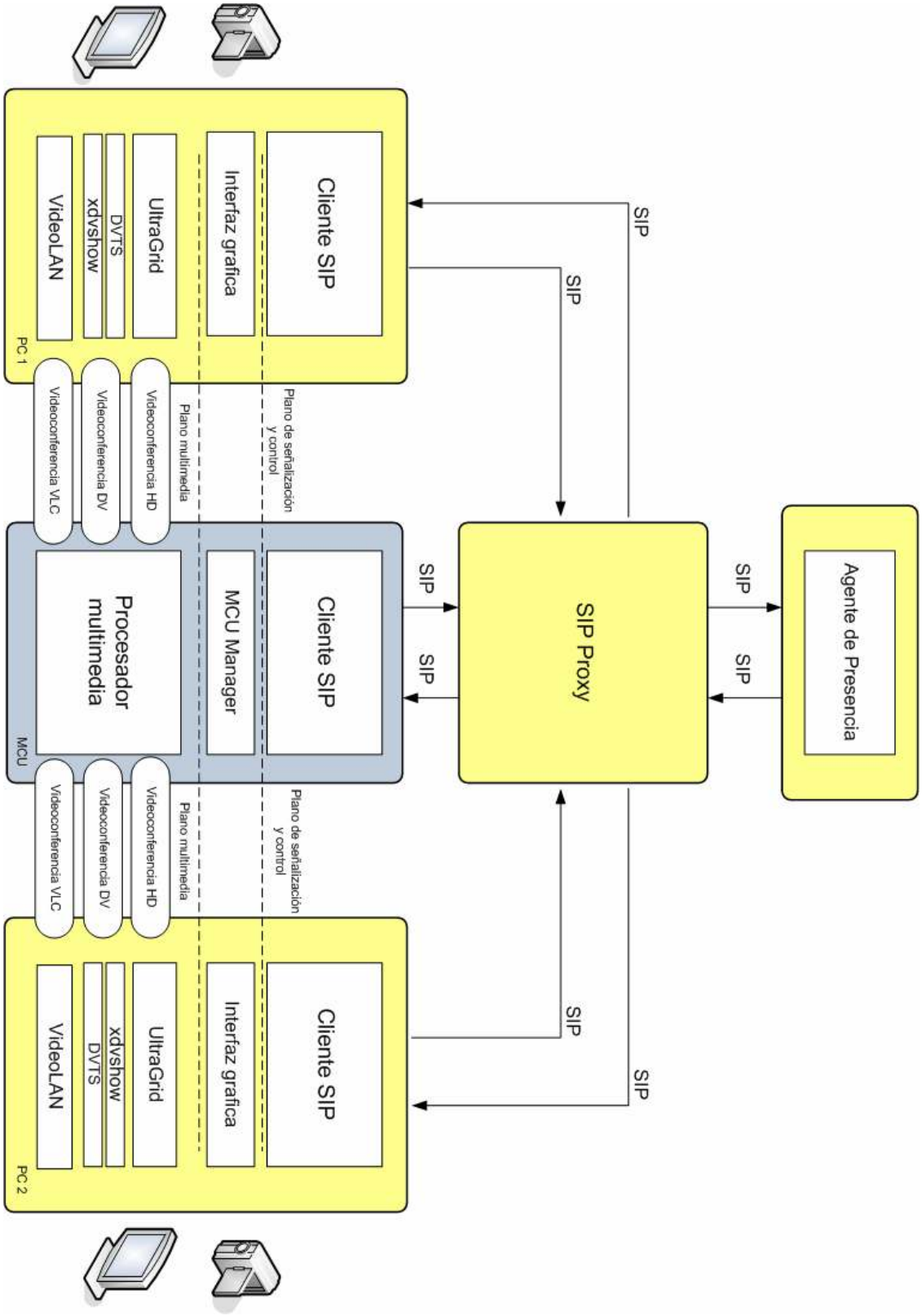


Fig. 2.6 División en planos

CAPÍTULO 3. DISEÑO DE LA MCU

En esta capitulo, se indican los requisitos iniciales de la aplicación, las soluciones estudiadas, y el diseño final realizado.

3.1. Requisitos iniciales del diseño

Al iniciarse el proyecto, se definieron unos requisitos mínimos, que debían ser completados al finalizar el proyecto:

- Utilización de señalización SIP estándar.
- Capacidad de control de una multiconferencia simultanea.
- Implementación de un sencillo procesador multimedia basado en Java.
- Diseño y desarrollo de un gestor de salas capaz de controlar más de una sesión de multiconferencia de forma simultánea.
- Compatibilidad e integración con el cliente SIP en desarrollo por el equipo de i2Cat.
- Mínimo de tres usuarios por multiconferencia.

Opcionalmente y para mejorar la eficiencia de la aplicación se propuso la creación de un segundo procesador multimedia, basado en C.

El diseño de nuestra *MCU* partió de la base de ser un sistema modular e independiente para su futura evolución y mejora. Para ello se realizaron búsquedas de nuevas tecnologías apropiadas para su fin o el uso de algunas ya conocidas por el desarrollador.

Durante el diseño y las primeras implementaciones se observaron carencias en la propuesta inicial y se fueron desarrollando soluciones que más adelante se comentaran.

3.2. Estudio de las necesidades y búsqueda de soluciones

Una vez comunicados los requisitos de nuestra aplicación, se realizó una primera búsqueda soluciones que nos ayudasen en el desarrollo de la aplicación.

Estas búsquedas se centraron básicamente, en tres diferentes campos:

- Protocolo de señalización, ya que uno de los requisitos básicos era la utilización del protocolo SIP como protocolo de señalización, Se propuso la utilización de una pila SIP sobre la cual, se programaría, la lógica necesaria de nuestra aplicación.

- Protocolo de control, nuestra aplicación iba a estar aislada de todos los demás elementos de la arquitectura. Por lo tanto, esta debía ser controlable externamente, se necesitaba definir un protocolo de control

(se comentará con mayor profundidad en el apartado 3.3.2) para poder gestionar la *MCU* a distancia.

- Procesado multimedia, se necesitaba buscar una herramienta ya creada, o diseñar una nueva herramienta propia, que fuese capaz de tratar flujos RTP o en el caso de trabajar sobre la capa de transporte, debía procesar datagramas UDP. El tratamiento básicamente se reduce a ser capaz de replicar la información, y de conmutar entre diferentes flujos de información.

3.3. Soluciones estudiadas

En este apartado se enuncian todas las soluciones propuestas durante la realización del proyecto. Durante el desarrollo de la aplicación se tuvieron en cuenta todas ellas, aunque finamente en el diseño final, sólo se seleccionaron las más eficientes.

3.3.1. Protocolo de señalización

SIP fue el protocolo de señalización seleccionado, y desde un principio se tuvo claro que la pila SIP usada seria la proporcionada por la API de Jain Sip. Un factor decisivo fue que el equipo de desarrollo de i2Cat ya había trabajado con esta librería y por lo tanto el desarrollo debía ser más rápido.

3.3.1.1. Jain Sip

Jain Sip, es la especificación propia del lenguaje Java para el uso de la señalización SIP. De esta especificación se han realizado varias implementaciones, durante este proyecto se estudio la implementación de referencia creada por NIST[6], ya que es una implantación libre y abierta, que nos permitía realizar modificaciones en caso necesario.

Jain Sip proporciona al desarrollador una pila SIP con la que es posible desarrollar cualquier tipo de aplicación con éste estándar de señalización.

3.3.2. Protocolo de control

Al no existir ningún tipo de exigencia o requisito mínimo para el diseño e implementación del protocolo de control del sistema, se estudiaron más de una única solución. A continuación se enuncian las diferentes opciones y se incluye una pequeña descripción de cada una de ellas.

[6] National Institute of Standards and Technology (Véase el anexo de acrónimos)

3.3.2.1. Protocolo de control propietario

La primera opción que se valoro, fue la de diseñar un protocolo de control propietario, basado en comunicaciones por sockets ya fuesen TCP[7] o UDP. La simplicidad del protocolo necesario, hacia factible éste diseño. Como puntos negativos cabe comentar que al realizar soluciones propietarias, y por lo tanto usar puertos no estandarizados, podríamos llegar a tener problemas en el futuro.

3.3.2.2. SIP + DO (XML)

La segunda solución propuesta fue la de reutilización del protocolo SIP como protocolo de control, añadiendo un mensaje SIP de tipo "DO", que contuviese un documento en XML[8], en éste documento se encontraría la acción a realizar. Éste tipo de mensaje fue definido en el draft [9] propuesto por la empresa Telcordia Technologies, que se incluye dentro del proyecto HomeSip, para la utilización del protocolo SIP como protocolo de control de una casa domótica.

Esta reutilización, por una parte simplificaba el desarrollo de un módulo de control, pero provocaba ciertas dudas. Posiblemente se debería reimplementar varios métodos de las librerías que controlan el protocolo de señalización y además cabía el riesgo de que ciertos SIP Proxy no aceptaran éste tipo de mensajes no estandarizados.

3.3.2.3. SIP + DO (SOAP)

La tercera solución propuesta fue una evolución de la anterior, en ella se contempla la incorporación del mensaje DO, pero en éste caso en vez de incorporar en su cuerpo, un documento XML, se debería incorporar un objeto serializado con SOAP[9], que generaría el realizador, y una vez recibido por la MCU, esta extraería el objeto y lo ejecutaría. Esta solución fue desarrollada por la empresa Ubiquity Software, y propuesta en un draft [10], la diferencia entre nuestra solución y la que ellos propusieron es que en nuestro caso el mensaje es del tipo "DO" y en su draft proponían un mensaje propio llamado "SERVICE".

3.3.3. Procesado multimedia

El procesado de la información recibida es uno de los puntos críticos de la MCU, debido a la cantidad de información que se debe tratar. Aparecieron diferentes soluciones a medida que se iba adelantando en el proyecto.

[7] Transmission Control Protocol (Véase el anexo de acrónimos)
[8] eXtensible Markup Language (Véase el anexo de acrónimos)
[9] Simple Object Access Protocol (Véase el anexo de acrónimos)

3.3.3.1. RTP Proxy

RTP Proxy [11] es una aplicación diseñada para procesar flujos multimedia, a través de Internet. Fue diseñada para facilitar el uso de clientes SIP (telefónicos), que estuviesen situados dentro de un NAT[10], éste software permite realizar las acciones que son necesarias para nuestra MCU, como son conmutar y replicar flujos. Su diseño fue realizado pensando en especificaciones de flujos de audio, así que se debería modificar su código interno, para poder permitirle trabajar con flujos de video. Fue desarrollado en C y esta optimizado para ofrecer el máximo rendimiento.

3.3.3.2. Media Proxy

Al igual que el RTP Proxy, Media Proxy [12], es otra aplicación diseñada para procesar flujos multimedia. Está basado en Phyton [13], y su código es de acceso libre.

3.3.3.3. Java.NIO y JAVA.IO

El desarrollo de un replicador y duplicador propio basado en Java [14], fue otra opción a considerar. Java ofrecía diferentes opciones para el tratamiento de sockets, Java.NIO [15] y Java.IO [16], cada una de ellas ofreciendo unas ventajas a cambio de unos inconvenientes asociados. Más adelante se comentaran estas pruebas y los resultados obtenidos.

3.3.3.4. Packet Reflector

La ultima, opción referente al procesado multimedia fue un software llamado Packet Reflector [17]. Realmente éste software fue desarrollado para replicar la información recibida por todos los participantes en un punto neutro, y así simular una red con soporte multicast (Comentado en el punto 1.3.1.2).

Éste software se esta desarrollando por parte de un equipo del CESNET en la república checa, la adaptación de su implementación de packet reflector seria posible ya que el código es de acceso libre, además la fundación i2Cat, mantiene contacto directo con el equipo de CESNET, lo que facilitaría el intercambio de conocimientos entre ambas entidades.

[10] Network Address Translator (Véase el anexo de acrónimos)

3.4. Arquitectura software MCU

Finalmente, se definió la siguiente estructura del software:

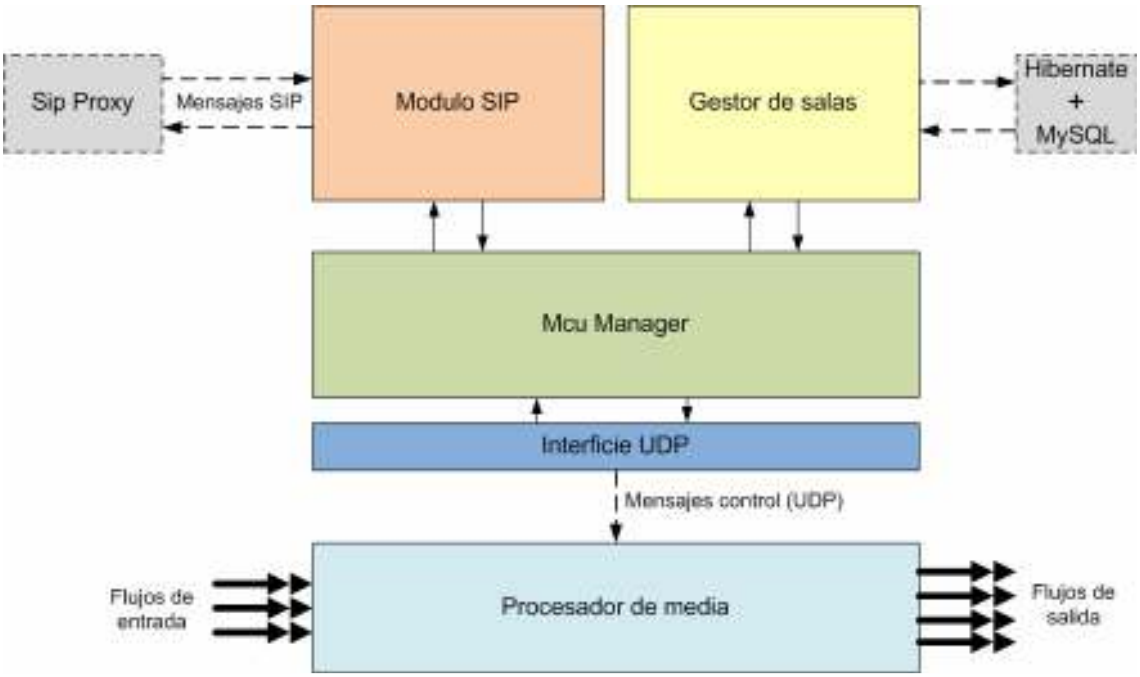


Fig. 3.1 Arquitectura de software

En la parte superior se observa, el módulo encargado de la señalización SIP, que se encargara de toda la comunicación entre todos los participantes SIP que existan y la propia *MCU*.

A su derecha se sitúa, el módulo que realizara la *gestión de salas*, en él se guarda toda la información referente a las salas de multiconferencia (Nombre de sala, usuarios participantes, realizador, etc.), además dispone de una serie de métodos para mantener actualizada esta información de las salas.

En el centro del diagrama se encuentra el *McuManager*, es el módulo central, encargado de gestionar las relaciones entre los diferentes módulos, dispone de referencias a todos de ellos, y tiene implementados diferentes métodos para realizar acciones sobre ellos. Al centralizar el control en único elemento, es más fácil de mantener y de saber el estado en el que se encuentran todos los módulos que forman parte de la aplicación.

En la parte inferior del diagrama se observa el módulo multimedia (procesador de media), que es el encargado de conmutar y redireccionar todos los flujos multimedia recibidos por la MCU. A nivel computacional es el más exigente, ya que se encarga de manejar grandes cantidades de información. La comunicación entre el módulo de control multimedia y el McuManager, se realiza a través de un socket UDP que utiliza un protocolo propietario.

En los siguientes apartados de la memoria ampliaremos la información de cada uno de estos módulos.

3.4.1. Gestor de salas

Gracias al uso de las técnicas de programación orientada a objetos se pudo modelar un sistema de gestión de salas con gran facilidad. Utilizando clases e interfaces como representación de los objetos que realmente representaría.

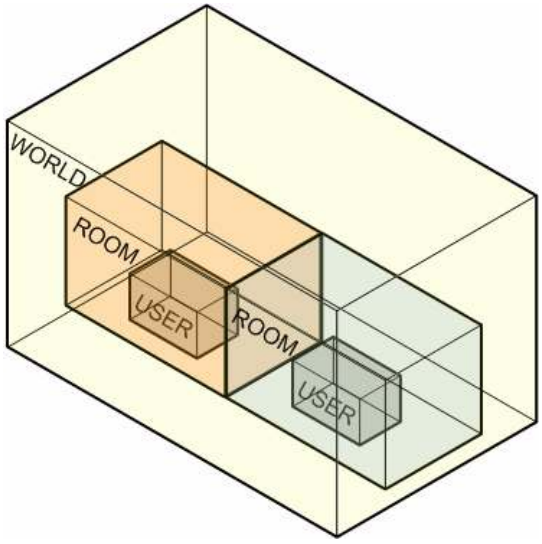


Fig. 3.2 Estructura del gestor de salas

Tal y como se observa en el gráfico anterior (Fig. 3., el sistema de *gestión de salas* se ha implementado en tres componentes World, Room y User donde cada una de ellas forma parte de la inmediatamente superior.

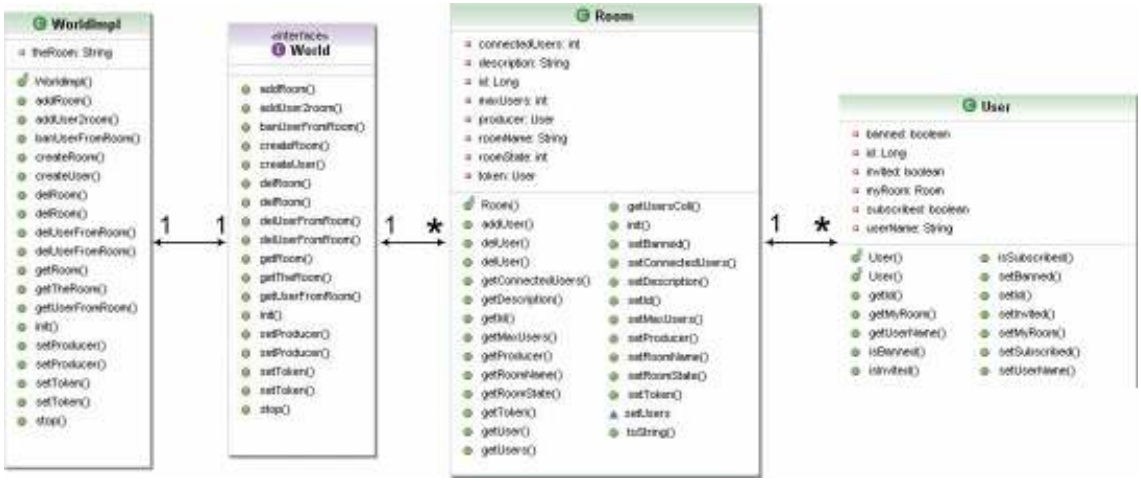


Fig. 3.3 Relación entre clases del gestor de salas

En este diagrama UML[11] podemos observar las relaciones entre los diferentes objetos.

[11] Unified Modelling Language (Véase el anexo de acrónimos)

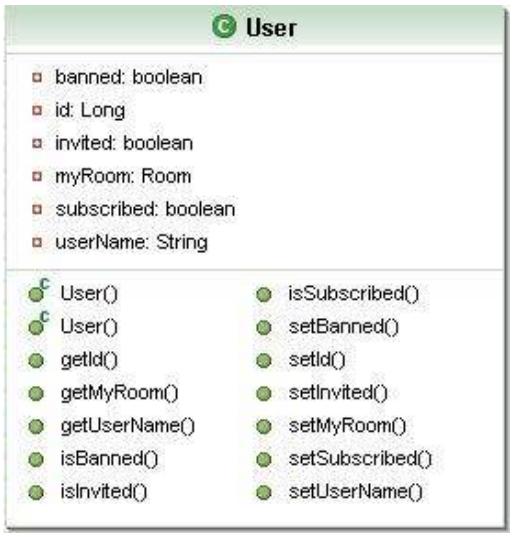


Fig. 3.4 UML objeto User

La clase **User** es la representación en un objeto de un participante en la multiconferencia, en este objeto se guarda como atributos el identificador de usuario, el nombre de usuario, y tres variables boléanas usadas para identificar el estado en el que se encuentra el usuario, así como una referencia a la sala en la que participa. Además de los atributos este objeto dispone de varios métodos para modificar estados, cambiar nombres, etc.…

Fig. 3.5 UML objeto Room

La clase **Room** representa a una sala, o lo que es lo mismo una sesión de multiconferencia. Como atributos contiene un identificador de sala, el nombre de la sala, una pequeña descripción de la actividad que se realizara en esa

sesión de multiconferencia, el número de usuarios máximos permitidos, usuarios conectados en ese momento. Respecto a los usuarios se almacenan en una estructura de datos todos los usuarios participantes, y se mantienen dos referencias, una para el usuario que tiene el turno de palabra, y otra para el usuario con privilegios de realizador. Al igual que la clase *Room*, dispone de varios métodos de gestión internos del objeto, cambiar el token[12], definir realizador, etc.…

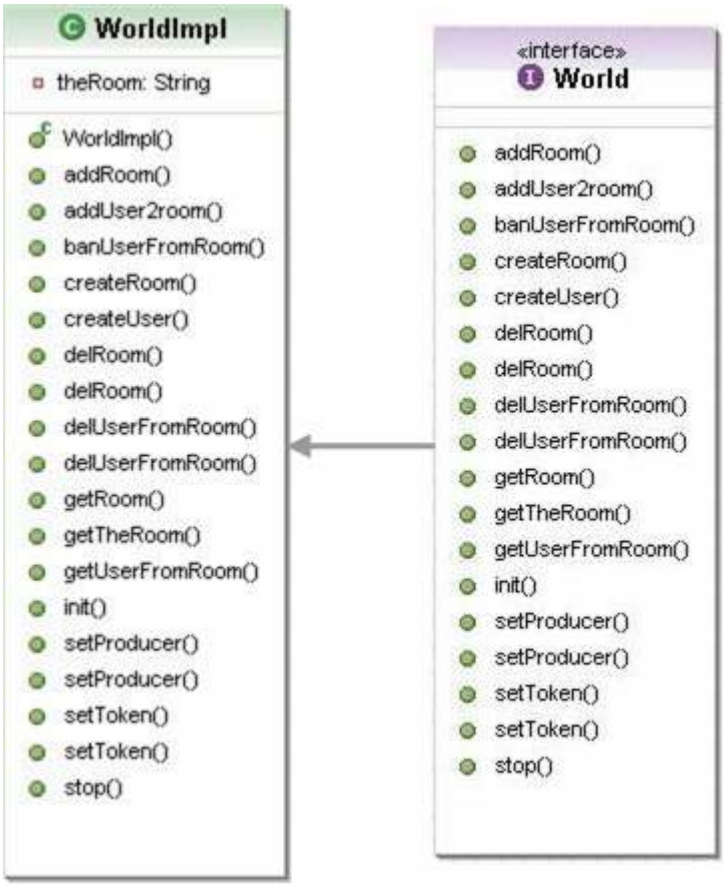


Fig. 3.6 UML World

Debido a la mayor complejidad del Objeto **World**, se decidió crear una interfaz intermedia para definir de forma clara cuales eran las acciones básicas que una implementación de esta interfaz debían tener. Esta interfaz se diseño pensando que las acciones y controles debían ser realizados a nivel de la clase World, no a niveles inferiores, así cualquier acción sobre el gestor de salas se deberá realizar desde la instancia del objeto World que se comportara como una caja negra y no permitirá acceder a capas inferiores.

La interfaz World, contempla las siguientes acciones o métodos:

- Inicializar, finalizar el gestor de salas.
- Añadir, editar, eliminar salas.
- Seleccionar participantes, asignar realizador y turno del usuario.
- Añadir, eliminar, o bloquear a un usuario que participe en una sala.

[12] Usuario que dispone del turno de palabra.

Una vez definida esta interfaz con todos los métodos necesarios para la administración de un conjunto de salas, se realizo la implementación de esta interfaz, con el nombre de WorldImpl. En esta clase además de lo anteriormente comentado, se encuentra una estructura de datos con todos los objetos Room creados, y otra estructura de datos donde se indexan todos los usuarios y el Room en el que se encuentran. Esta segunda estructura de datos no es necesaria para el funcionamiento del gestor de salas, pero se decidió incorporarla para casos en que fuese necesario disponer de una búsqueda global de un usuario (o modificación de información de un usuario), con éste sistema el resultado es mucho más eficiente y rápido, de lo que seria una búsqueda sala a sala.

Además el gestor de salas se ha diseñado de tal forma que, usando la librería Hibernate, se puedan realizar mapeos entre objetos y una base de datos para así poder almacenar de forma más fiable la información necesaria.

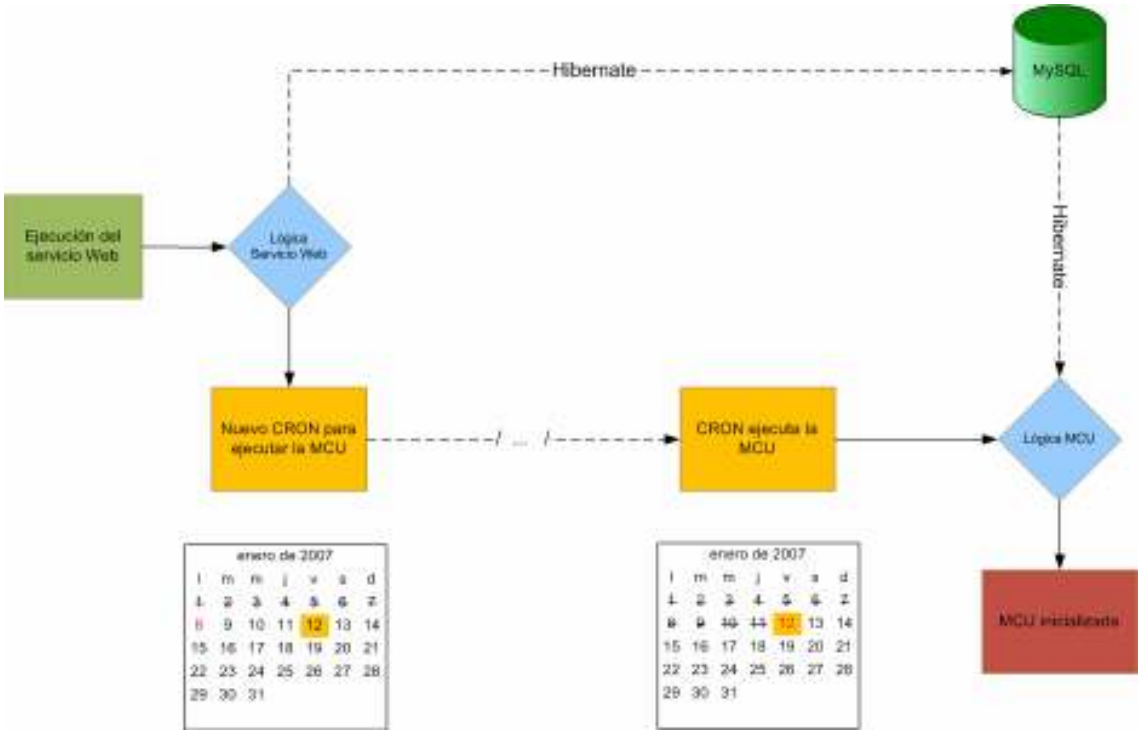


Fig. 3.7 Hibernate con MCU

En el diagrama anterior se muestra el uso de esta tecnología. Esta funcionalidad se utilizará para planear las sesiones de multiconferencia a través de un servicio Web.

3.4.2. Módulo SIP

Éste es uno de los módulos más complejos de todo el sistema, es el encargado de generar y procesar todos los mensajes de señalización de la MCU, además de los mensajes de control que enviara el realizador y afectara a toda la MCU.

3.4.2.1. Estructura del módulo

El módulo SIP, esta formado por las siguientes clases y packages:

SipClient, es una interfaz donde se definen los métodos básicos que va a necesitar nuestro módulo SIP.

SipClientImpl, es la implementación de la anterior Interfaz. En ella se define una maquina de estados con toda la lógica de señalización, en cada estado se realizan las acciones pertinentes, y se sigue un orden preestablecido para controlar errores.

En el caso de la MCU la maquina de estados que se ha diseñado responde al siguiente modelo:

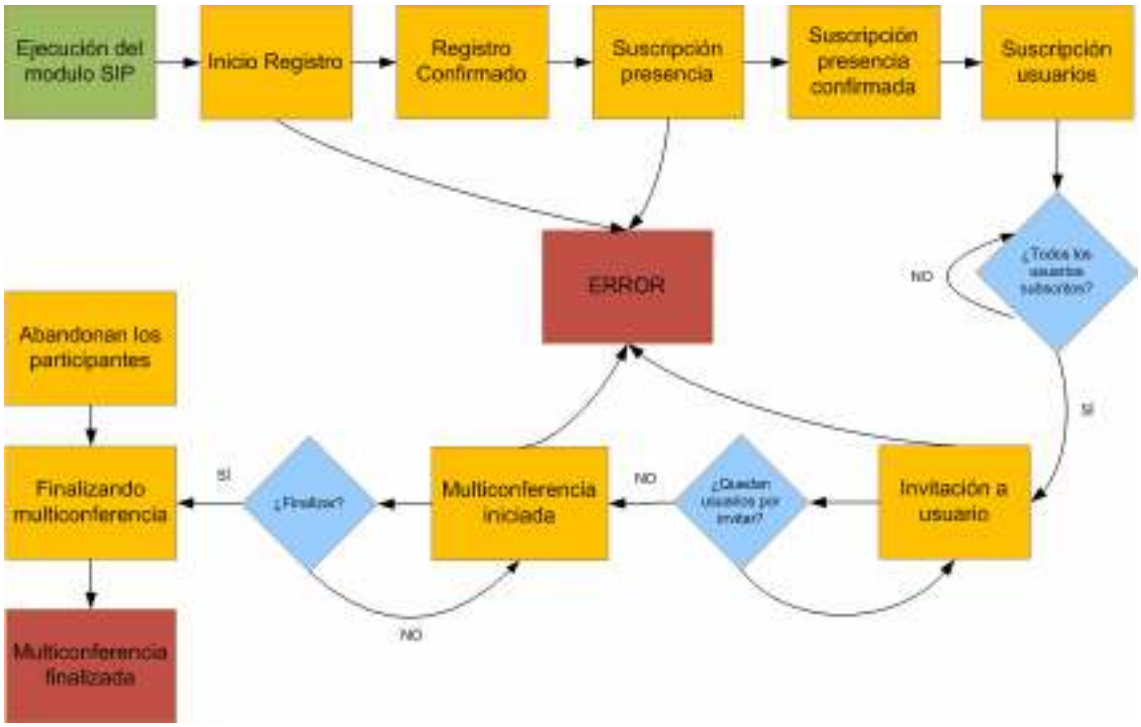


Fig. 3.8 Maquina de estados SipClientImpl

Al iniciarse el módulo SIP, el primer estado consiste en Registrar la MCU contra el SIP Proxy, y así poder enviar/recibir mensajes a través de el. Si el registro no fuese posible, el cliente pasaría al estado de “error” con un código asociado a éste error para identificarlo, si el registro se realiza correctamente pasaríamos al estado de registro confirmado.

Una vez registrada la MCU, existen dos posibilidades, utilizar el sistema de presencia y esperar a que los demás usuarios, se subscriban a nuestra sesión de multiconferencia, o pasar directamente a invitar a los usuarios participantes, en éste caso comentaremos el modo de funcionamiento más completo.

Al igual que el registro necesitamos avisar a todos nuestros clientes que la sala de multiconferencia ya esta disponible para que los participantes puedan acceder. Para ello utilizaremos el servidor de presencia, donde al subscribirnos como MCU, se enviaran unas notificaciones a los demás usuarios subscritos, informando de nuestra presencia. Si todo transcurre correctamente pasaríamos al estado de suscripción confirmada, si no al igual que en el caso del registro se pasaría al estado de error con un código asociado.

Después de subscribirnos al servidor de presencia, el siguiente paso es que los participantes de nuestra sala, se subscriban a nuestra MCU. Una vez todos se hayan subscrito, pasaríamos al siguiente estado.

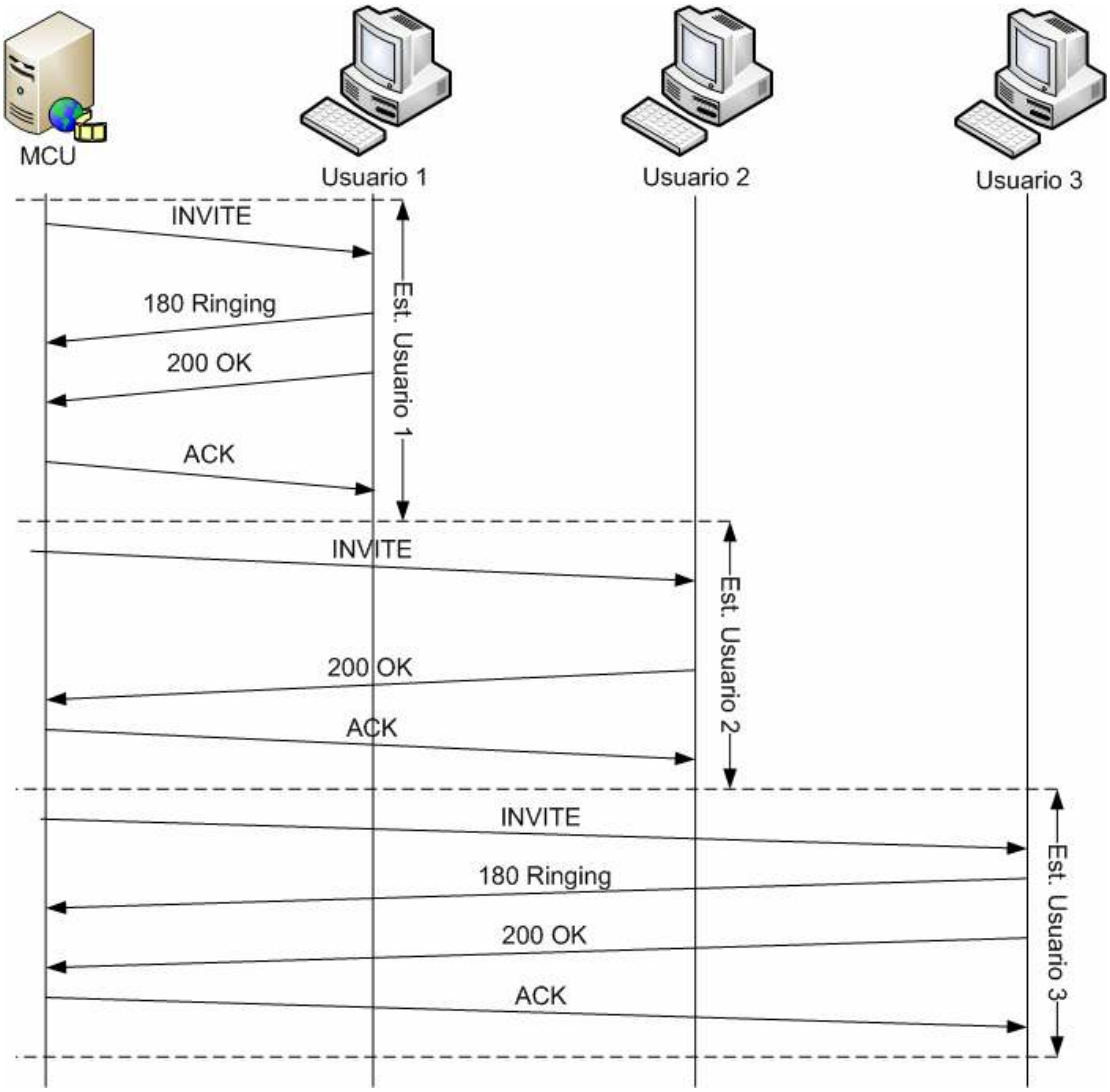


Fig. 3.9 Señalización inicio multiconferencia

En el estado de “invitación a usuarios” se realiza una a una, las invitaciones de todos los participantes de la multiconferencia (Fig 3.9). Cada vez que es aceptada la invitación por un participante, se comprueba si falta algún usuario por invitar, si es así, se vuelve al estado anterior y se realiza la invitación, una vez todos los usuarios han sido invitados, la sesión de multiconferencia se da por establecida.

Una vez iniciada la multiconferencia sólo hay dos formas de finalizarla, la primera es que el realizador decida finalizar la conferencia, con lo cual se enviaran los mensajes de señalización oportunos y se finalizara. La segunda opción sería que los participantes de la multiconferencia abandonasen la sala, por lo tanto, no tendría ninguna lógica seguir ocupando recursos que no son utilizados. Al igual que en el primer caso, se enviarían los mensajes necesarios y la sesión de multiconferencia se daría por acabada.

SipActor, en esta clase se almacenan todos los atributos y los métodos genéricos que son necesarios para poder crear mensajes SIP por la Jain Sip. Por cada tipo de mensaje que se necesita crear se ha implementado un método, en el cual se comprueba la existencia de todos los atributos necesarios, y una vez agrupados, se crean las cabeceras que forman el mensaje.

MySip, es una extensión de SIP Actor, donde además de lo anteriormente comentado, se añade la lógica de intercambio de mensajes SIP. Es decir, las respuestas que se van a generar dependiendo del mensaje recibido a través de la pila SIP.

Una vez notificado un nuevo mensaje, es procesado y se ejecuta el método necesario, en la mayoría de casos se crea una respuesta y si es necesario un cambio de estado, se notifica a SipClient, para que esta actualice el estado y ejecute la lógica del nuevo estado.

Existe un caso especial, que es el referente al protocolo de control, ofrecido a través de la pila SIP. Al recibir un mensaje de Control a través de la pila SIP (Mensaje del tipo DO), lo primero es comprobar a través del McuManager si el usuario tiene permisos de realizador, si es afirmativa la respuesta, el siguiente paso es extraer el XML del cuerpo del mensaje, y pasarlo como referencia al McuManager, que se encargara de su procesado, éste caso se explicara de forma más amplia en la sección 3.4.2.2.

Clase **MessageDigestAlgorithm** y package **AKA**, son un conjunto de clases para poder realizar autenticación en servidores donde sea necesario autenticarse de forma segura. Para la autenticación en servidores seguros, al registrarse se debe realizar un intercambio de mensajes especial (“handshake”). Durante éste “handshake” se intercambian diferentes informaciones entre cliente y el servidor, y en estas clases están implementados los cálculos necesarios sobre la información intercambiada.

3.4.2.2. Modificaciones JainSip para implementar el mensaje DO

Al utilizarse SIP como protocolo de señalización, la opción del protocolo propietario de control finalmente se descarto, ya que reutilizando la pila ofrecida por SIP, se eliminaban los problemas derivados de implementar una nueva pila de protocolos no estandarizada, por ejemplo, el posible bloqueo de

comunicaciones al pasar por algún NAT, o el uso de un puerto ya usado por otra aplicación.

Para desarrollar el nuevo mensaje del tipo DO, se modifico la implementación ofrecida por NIST de Jain Sip, en esta nueva implementación se añadió un nuevo tipo de mensaje, y se genero una lógica de respuestas a éste mensaje.

```
DO sip:Prueba@192.168.48.209:5080 SIP/2.0
Call-ID: 4f242473e77d1f9f6b4705cafad46c6d@192.168.48.209
CSeq: 5 DO
To: <sip:Prueba@192.168.48.209>;tag=12345
From: <sip:3@192.168.48.124:5060>;tag=4321
Max-Forwards: 68
Via: SIP/2.0/UDP 192.168.48.124:5060;
Record-Route: <sip:192.168.48.124:5060;lr>
Content-Type: text/xml
Content-Length: 54

<action type="SELECT"><user value="2"/></action>
```

Fig. 3.10 Mensaje DO

Tal y como se observa en la figura, el mensaje DO, contiene las cabeceras genéricas de un mensaje SIP, el “content type” indica el tipo MIME del contenido del mensaje, y dentro del cuerpo del mensaje se puede observar un pequeño documento en XML que indica la acción que se debe ejecutar en nuestra MCU.

La implementación realizada de este mensaje nos permite, tanto el uso como contenedor de lenguaje XML, como el uso de contenedor de objetos serializados con SOAP[13]. Finalmente se selecciono el contenido XML, para facilitar el desarrollo de la aplicación, aunque durante el desarrollo de esta siempre se tuvieron en cuenta las necesidades de la serialización con SOAP, para que en un futuro sea posible evolucionar hacia esa tecnología sin problemas.

Al recibir mensaje del tipo DO, la *MCU* comprueba sus cabeceras y su contenido, una vez realizadas todas las acciones necesarias, contesta al cliente que genero el mensaje con una respuesta del tipo 200 OK, esta respuesta no debe ser confirmada por el otro extremo, en caso de perdida de alguno de los dos mensajes (petición o respuesta), el cliente que genera la acción, es el encargado de reenviar el mensaje del tipo DO de nuevo.

[13] Simple Object Access Protocol (Véase el anexo de acrónimos)

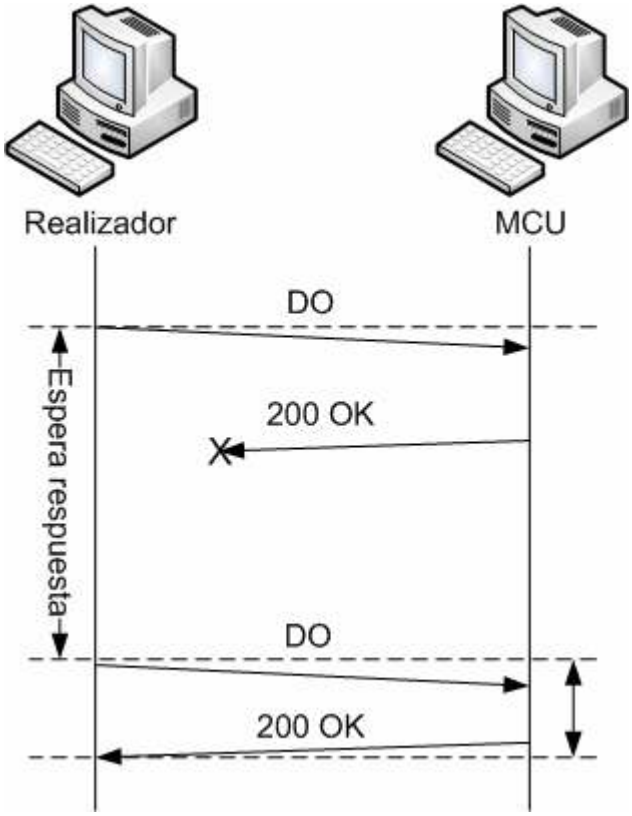


Fig. 3.11 Intercambio de mensajes DO

El mensaje XML generado por el realizador representa el tipo de acciones que el realizador puede realizar sobre la MCU.

```
<action type="BAN"><user value="1"/></action>
```

Fig. 3.12 XML

El atributo “type”, representa el tipo de acción, y “value” representa el usuario al que va a afectar la acción. El atributo type, puede tomar como valor cualquiera de las tres acciones posibles (“SELECT”,”BAN”,”BYE”) y en atributo “value”, una cadena caracteres, que representa el nombre del usuario. Por ejemplo en la figura anterior la acción que se demandaba era la de vetar el acceso a la MCU, para el usuario identificado como 1.

3.4.3. MCU Manager

Se encarga de interconectar todos los módulos y realiza las llamadas entre módulos necesarias, dependiendo del evento que se éste ejecutando.

Al ser un módulo central y único dentro de nuestra MCU se decidió implementar usando el patrón de diseño de un Singleton [26]. Singleton es un tipo de patrón de diseño unitario, diseñado para restringir a una única instancia de una clase, y que facilita en gran medida el hecho de coordinar las acciones.

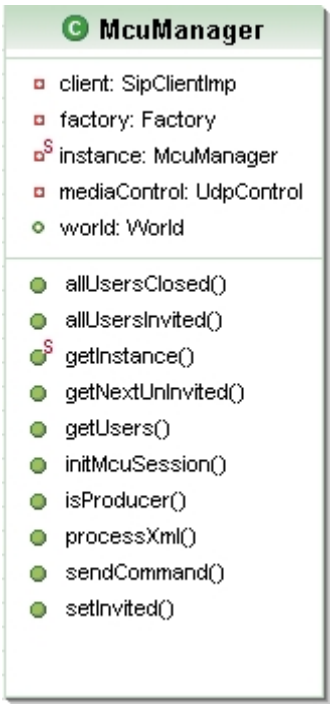


Fig. 3.13 UML McuManager

Dentro del *McuManager*, tenemos referencias a diferentes módulos de nuestra MCU:

- Referencia al *Gestor de Salas*.
- Referencia al *Cliente SIP*.
- Referencia a la interfaz que comunica con el procesador multimedia.
- Métodos necesarios, para interconectar los diferentes módulos, ya que no son visibles entre ellos, por esa razón cualquier acción que necesite ser notificada a más de un módulo, debe intervenir el *McuManager*.

Para hacer el *McuManager* lo más escalable posible y así evitar tener que realizar grandes modificaciones sobre él, se realizo una Interfaz que representa las acciones que puede realizar el manager.

Esta interfaz no presenta gran complejidad ya que las acciones que la MCU ejecutara son muy sencillas (seleccionar flujo de video, bloquear flujo, finalizar multiconferencia), así que en el peor de los casos, se necesitarían dos atributos para identificar una acción, que son el tipo de la acción y el destinatario de ella. La generación de estos objetos será realizada por una factoría de objetos, esta factoría se ejecuta cada vez que se recibe un mensaje un documento XML extraído del mensaje de control, este documento es tratado por el procesador de XML. Dicho procesador se ha programado utilizando las librerías SAX [19] de Java.

El tipo de acción será representada por el nombre de la clase que implementa el interfaz, y si es necesario dentro de cada clase existirá un atributo que identifique el destinatario de tal acción.

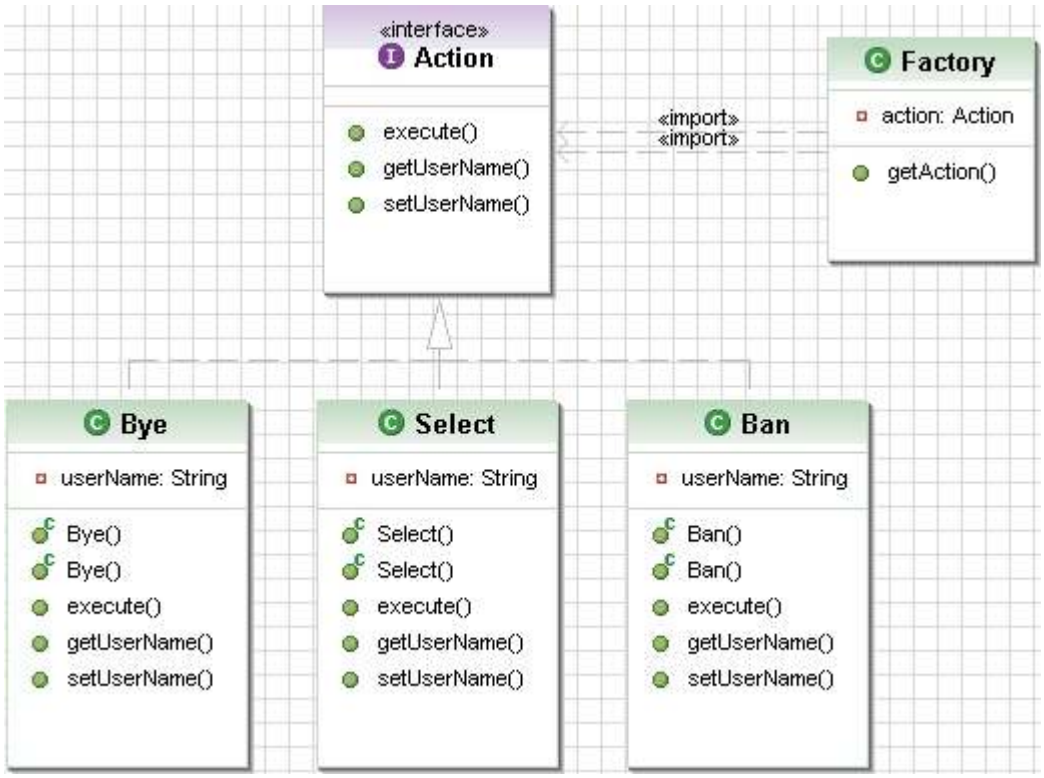


Fig. 3.14 UML package Action

A partir de esta interfaz básica, se implementaron los 3 tipos de acciones que debía soportar nuestra MCU (“Ban”, “Select”, “Bye”). Tal y como se comento al principio, el uso de una interfaz, facilita en gran medida, la adopción de nuevos tipos de acción, tan solo se debería crear una nueva clase que implementase la interfaz, y así no se deberían realizar ningún tipo de modificaciones sobre el McuManager.

Estas acciones al ser ejecutadas, realizan toda la lógica que sea necesaria, habitualmente afectara tanto al gestor de salas, como al procesador multimedia. Al implementarse el módulo encargado del procesado multimedia en otro lenguaje de programación (para aumentar su eficiencia), se decidió realizar las comunicaciones entre las dos aplicaciones a través de un socket UDP.

3.4.3.1. Protocolo de control procesador multimedia

Para poder acceder al procesador multimedia era necesario implementar un sencillo cliente, que fuese capaz de enviar datagramas (UDP) hacia el módulo de procesado multimedia.

Se decidió desarrollar un sencillo protocolo de mensajes entre el McuManager, y el procesador multimedia, ya que ambos módulos en principio estarán en la misma maquina, no se decidieron implementar mecanismos de control (confirmación de recepción), ni se considero la posibilidad de pérdidas de paquetes, ya que el entorno de uso las estas serian nulas.

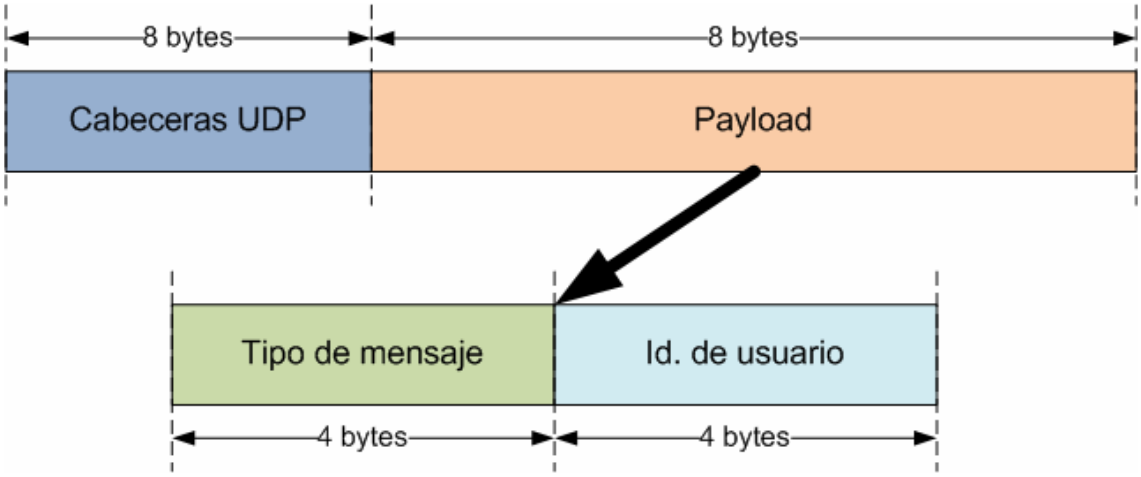


Fig. 3.15 Protocolo de control procesador multimedia

El protocolo diseñado, esta formado simplemente por dos campos con estructura de tipo entero (un entero ocupa 4 bytes).

Estos dos campos pueden tomar diferentes valores, en la actualidad el protocolo tiene asignados los siguientes valores:

Campo tipo de mensaje:

- Valor “0” conmutación de flujo hacia un usuario.
- Valor “1” bloqueo de un usuario (imposibilidad de participar).
- Valor “2” fin de la multiconferencia.

Campo Identificador de usuario:

Este campo es referido al identificador dentro de la sala de multiconferencia, en la actualidad sólo se soportan multiconferencias de hasta 3 usuarios (1,2 y 3), valores superiores a estos son ignorados por el procesador.

3.4.4. Procesador multimedia

El desarrollo del procesador multimedia se realizó tomando como partida el código del rum [20] (Una sencilla aplicación que replicaba datagramas UDP), y que es el embrión del actual “packet reflector” desarrollado por CESNET. Este módulo es capaz de funcionar en diferentes sistemas operativos, desde equipos UNIX o Linux, hasta equipos Windows siempre que se instale el software Cygwin [21], para permitir la instalación y ejecución de programas creados para entornos Unix.

A nivel de diseño, el módulo desarrollado utiliza una arquitectura de un único hilo y dos sockets UDP: un socket es el destinado a recepción y envío de información, y el otro socket es utilizado para recibir mensajes de control.

Se descartó la utilización de buffers internos para la recepción de paquetes, al ser un módulo que procesa gran cantidad de información y de una forma muy rápida. El buffer nos permite compensar las pequeñas variaciones de tasa que siempre existen en flujos de tasa variable, pero en nuestro caso la problemática es diferente.

Al trabajarse al límite de procesado del equipo, las pérdidas se producen al no poderse atender a todos los datagramas enviados por los participantes hacia nuestro socket. Implementándose un buffer se consigue controlar estas pérdidas durante un corto periodo de tiempo, pero finalmente llegamos al mismo estado, y como puntos negativos añadimos un retardo y una de sincronización entre los flujos recibidos.

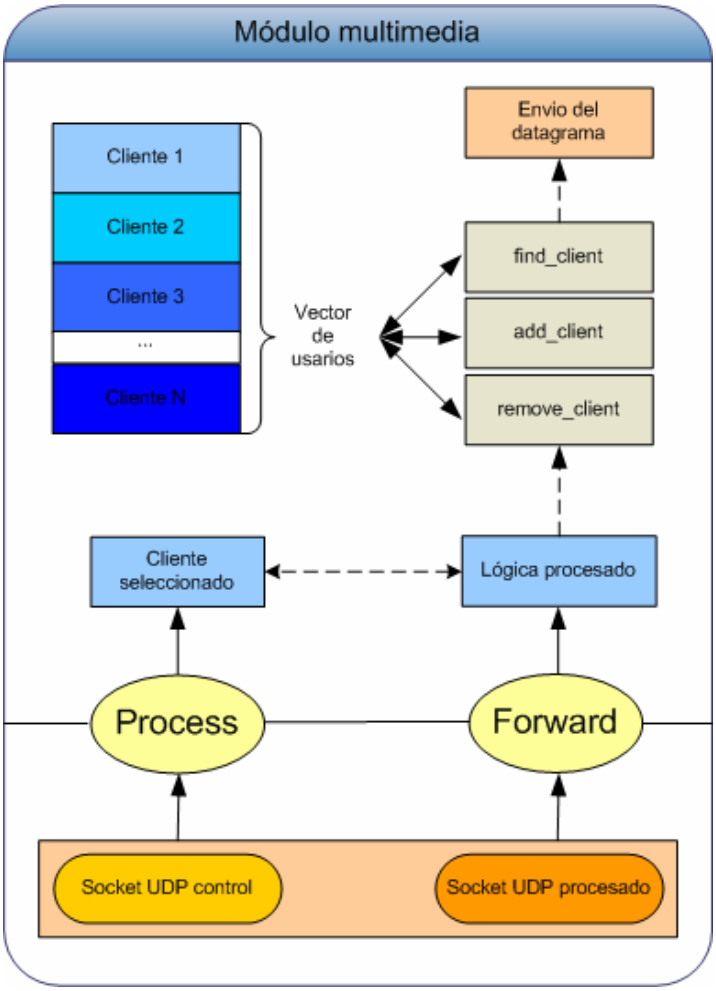


Fig. 3.16 Diagrama módulo multimedia

Existe una estructura de datos donde se guarda información de los participantes que están enviando datos hacia la MCU, básicamente los sockets a los que se les debe de enviar información.

Para administrar estas estructuras se creo un vector, y tres funciones que permiten gestionar éste vector. Una función es la encargada de añadir nuevos usuarios al enviar por primera vez información hacia la MCU (estos usuarios

son indexados en el vector por el numero de usuarios conectados), otra realiza un recorrido por todo el vector y elimina los usuarios que se han desconectado, y la ultima función realiza una búsqueda en el vector de usuarios y en caso de que exista nos devuelve su posición en el vector.

La lógica implementada en éste módulo es sencilla: se dispone de dos sockets que reciben información, en caso de que se haya recibido información en el socket de procesado multimedia, se ejecuta la función con la lógica de reenvío, y si se trata del socket de control, se procesa el mensaje recibido y se actualizan los parámetros necesarios dentro del módulo.

CAPÍTULO 4. IMPLEMENTACIÓN Y PRUEBAS

Java fue el lenguaje seleccionado para el desarrollo mayoritario de la aplicación, ya que al ser un lenguaje orientado a objetos, el desarrollo de una aplicación se reduce considerablemente respecto al tiempo de desarrollo en un lenguaje de programación convencional (C, C++, Basic). Al ser el tiempo de desarrollo un gran condicionante (apenas cuatro meses), esta era una solución óptima.

4.1. Tecnologías usadas

Después de un proceso de investigación y pruebas de las diferentes soluciones propuestas, se seleccionaron las tecnologías para la implementación final de la *MCU*.

4.1.1. Java JDK

Java Virtual Machine and Development Kit, éste kit nos ofrece la maquina virtual y las librerías necesarias para poder desarrollar aplicaciones basadas en el lenguaje Java.

4.1.2. Eclipse SDK

Es un entorno de desarrollo (IDE) de código abierto, incluye todas las utilidades necesarias para desarrollar aplicaciones, comparador de versiones, cliente CVS, etc… Inicialmente fue diseñado para desarrollar aplicaciones Java, en la actualidad gracias a la utilización de plugins, es posible desarrollar en la mayoría de lenguajes. La mayoría de nuestra aplicación se ha desarrollado utilizando éste software.

4.1.3. GCC

GNU Compiler Collection (son las siglas de GCC) es un conjunto de compiladores para diferentes lenguajes de programación desarrollados por el GNU Project. Es un software libre, que se distribuye a través de Free Software Foundation (FSF) con licencia GNU GPL. Se ha establecido como el compilador estándar para toda la comunidad OpenSource que desarrolla para entornos Unix, y además para otros sistemas operativos como puede ser el Mac OS X. En nuestro caso se ha utilizado para poder compilar el código C de nuestra aplicación.

4.1.4. XML

XML, sigla en inglés de eXtensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el

World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML. Se utiliza en diferentes partes de nuestro proyecto, desde la definición de una tarea para Ant (apartado 4.1.7), hasta para definir el contenido de un mensaje del protocolo de control (apartado 3.4.2.2).

4.1.5. Cygwin

Cygwin es una colección de software libre, estas aplicaciones fueron desarrolladas originalmente por Cygnus Solutions para permitir que varias versiones de Windows se comportasen como un sistema basado en Unix. Básicamente se ha utilizado para portar software que funciona en sistemas POSIX (como Linux, Freebsd, y UNIX).

Cygwin ofrece una librería que implementa la API de llamadas de sistema de POSIX y las transforma en llamadas Win32, un kit de herramientas de compilación, y los programas básicos necesarios para simular un entorno Unix.

4.1.6. VideoLan client (VLC)

Básicamente es un reproductor multimedia gratuito. Soporta la mayoría de codificaciones, que existen en la actualidad (Mp3, Divx, DVD, etc…) Una funcionalidad muy utilizada en el desarrollo de esta aplicación, ha sido el servidor de streaming integrado en el, que nos permite enviar flujos multimedia sobre diferentes protocolos (http, udp unicast, udp multicast, etc…). Estos flujos pueden ser transcodificados, modificar las características del flujo en tiempo real, tanto en la codificación como en las tasas de transmisión. Éste software se usa como fuente y receptor de los flujos multimedia de nuestra aplicación.

4.1.7. Apache Ant

Es una herramienta de desarrollo donde se pueden especificar tareas de todo tipo (compilar, ejecutar, mover ficheros, crear archivos compactos (JAR/WAR), ejecutar pruebas unitarias, etc…), estas tareas se definen en un fichero que usa sintaxis XML (build.xml). En nuestro proyecto se ha creado un fichero de tareas, que nos permite generar los archivos de configuración de nuestra aplicación, compilar la aplicación y finalmente ejecutarla de forma automatica.

4.1.8. Log4j

Es la aplicación de gestión de trazas por excelencia en Java. Se utiliza para controlar la aplicación, y para detectar los posibles errores en ella, además es

muy importante su uso en aplicaciones “standalone” (que se ejecutan de forma independiente), donde no existe una interfaz grafica para el usuario, como es el caso de nuestra MCU.

Utilizando log4j es posible modificar los parámetros de registro sin tener que modificar el código de nuestra aplicación, ya que dispone de un fichero de texto plano donde se definen los niveles en los que la aplicación debe ser registrada.

4.1.9. SAX (Simple API for XML)

Serial access parser API for XML. SAX nos ofrece un procesador de documentos XML. Se ha de implementar para cada tipo de documento una lógica propia, donde podremos definir las acciones a realizar a cada vez que SAX procese un campo de nuestro documento XML.

4.1.10. CVS (Concurrent Version System)

Es un sistema que nos permite compartir ficheros entre diferentes desarrolladores, y realiza control de versión de estos ficheros. Así se puede compartir de forma rapida y sencilla el contenido de un proyecto entre diferentes desarrolladores.

4.1.11. Excelsior JET

Excelsior JET es un kit de herramientas, para la optimización de aplicaciones desarrolladas en JAVA. Con JET es posible compilar la aplicación en modo nativo, tal y como lo haría un lenguaje no interpretado, y por lo tanto exprimir al máximo las capacidades del equipo en el que se éste ejecutando. Además de esto con JET se consigue una proteger los intereses del desarrollador, ya que al ser un fichero compilado, no es posible visualizar el código fuente de la aplicación.

4.1.12. Jain SIP

Para implementar la señalización de la MCU, se ha utilizado la librería Java de Jain Sip, concretamente la implementación de referencia ofrecida por NIST. Esta tecnología, ya había usada en varios proyectos de i2Cat, con lo cual se esperaba que el desarrollo del cliente SIP fuese muy rápido, y los problemas que apareciesen en su desarrollo no fuesen de gran dificultad.

Jain Sip ofrece al desarrollador una pila SIP, y todas las herramientas necesarias para poder programar aplicaciones que utilicen el protocolo SIP. Estas aplicaciones pueden ser variadas, desde un “User agent” (Cliente Sip), un SIP Proxy, hasta empotrarlo en un servidor de aplicaciones como por ejemplo Tomcat para poder realizar aplicaciones sobre el.

4.2. Escenario de pruebas

Para simplificar el entorno de pruebas, se decidió sustituir algunos componentes por otros de menor complejidad.

El software DVTS, y XDVShow fue substituido por el reproductor multimedia Videolan, éste dispone de un emisor/receptor de flujos multimedia basado en RTP/UDP, similar al que usan las aplicaciones que hemos comentado. Se ejecutan dos instancias de VideoLan [22] por cliente (sin permisos de realización), una encargada del envío de un flujo y otro de la recepción, y para el caso de realizador se ejecutan 3 instancias, una para enviar su flujo y dos para recibir el flujo de los participantes realizados.

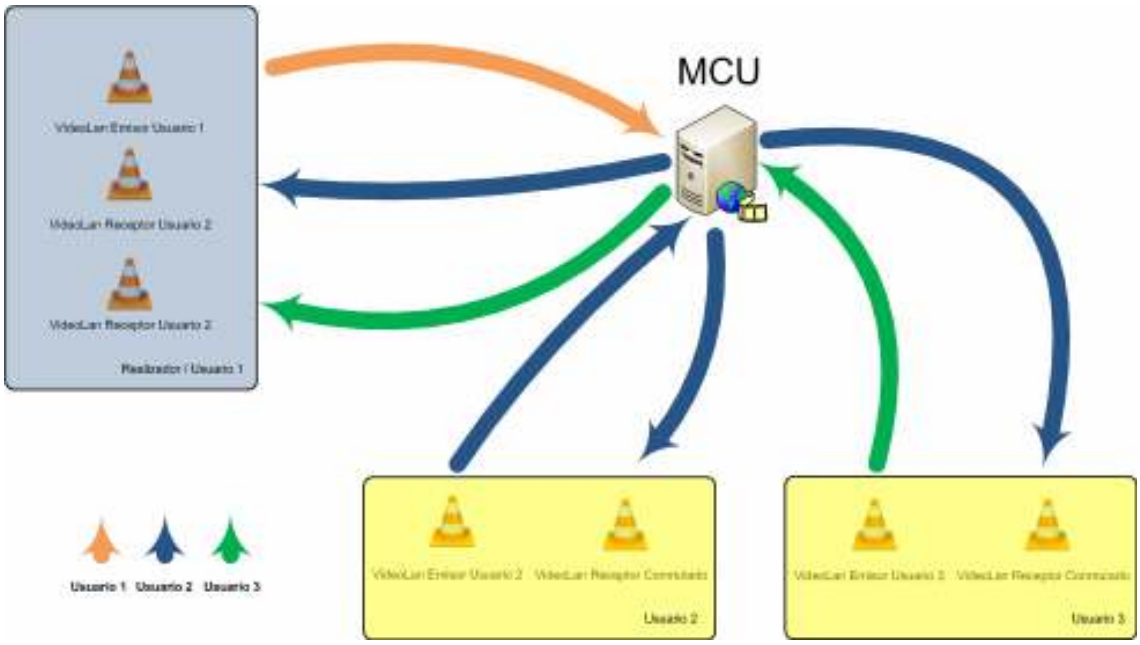


Fig. 4.1 Entorno de pruebas software

Los videos que se usaron en las pruebas proporcionaban tasas de entre 1Mbps hasta 20 Mbps aunque estas tasas pueden variar en el tiempo debido al uso de codificación VBR (Tasa de codificación variable), a mayor movimiento en la secuencia mayor tasa.

Los equipos usados en las pruebas disponían de la siguiente configuración:

- Pentium IV 3,2 Ghz
- 1 GB de memoria RAM DDRII 400Mhz
- Tarjeta grafica ATI Radeon X300
- Tarjeta de Red Marvell Yukon 88E8053 PCI-E Gigabit Ethernet
- Disco duro IDE 80 GB
- Windows XP Professional con SP2 o Ubuntu Linux 6.06 LTS

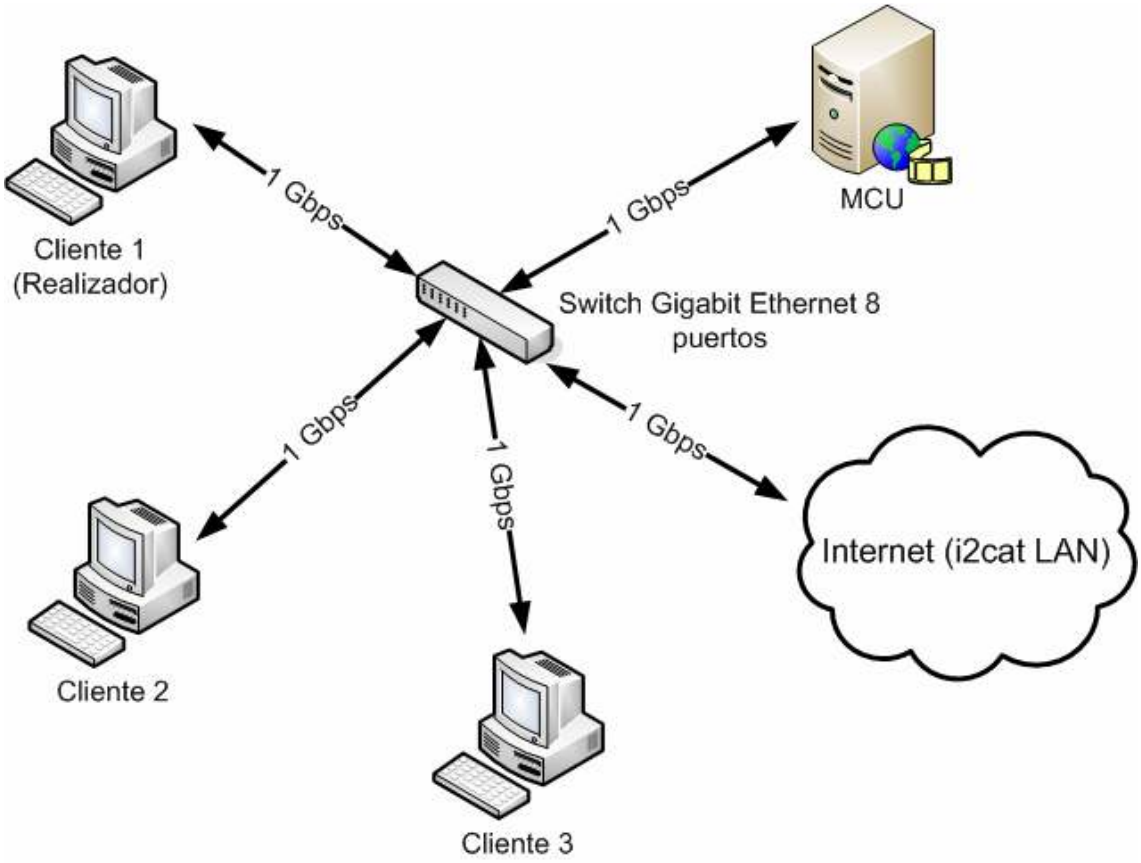


Fig. 4.2 Entorno de pruebas red

En los aspectos referentes a la red, se intentó usar la máxima capacidad disponible de las interfaces de red de nuestros equipos.

Al realizar cálculos teóricos de ancho de banda necesario en la red, se observo que un switch Fast Ethernet (100 Mbps) no era suficiente para soportar las transmisiones de datos, así que para evitar que este dispositivo generase un cuello de botella que afectaría al rendimiento de nuestra aplicación, se adquirió un nuevo switch con soporte para Gigabit Ethernet.

4.3. Pruebas de concepto (prototipos)

Uno de los puntos débiles de éste proyecto era el procesador multimedia, para ello se estudiaron gran variedad de soluciones, de todas ellas se seleccionaron varias, y se realizaron pequeñas pruebas de concepto (prototipos) para su utilización en el proyecto. Estas pruebas básicamente se realizaron para poder saber a antes de desarrollar la aplicación por completo cual seria la capacidad de procesado sobre diferentes condiciones.

Durante estas pruebas se generaron prototipos con arquitecturas de un único hilo, y arquitecturas multi hilo. En el caso de arquitecturas de un único hilo, este hilo será el encargado de recibir el datagrama, y reenviarlo hacia su destinatario. Para el caso de una arquitectura multi hilo, existe un hilo por cada usuario conectado, lo cual a priori debería mejorar el rendimiento de la

aplicación a costa de una gestión de los hilos bastante más compleja. Durante las pruebas realizadas en arquitecturas multi hilo no se obtuvo el resultado esperado, ya que se perdía demasiada información en recepción, posteriormente se realizaron lecturas de informes de otros desarrolladores donde no recomendaban la utilización de arquitecturas multi hilo debido básicamente a problemas de sincronismo de los flujos.

En el primer prototipo desarrollado (único hilo y sin buffer de recepción), la falta de el buffer generaba grandes pérdidas al utilizar flujos con tasas superiores a los 6Mbps, se decidio implementar un segundo prototipo que disponía de un buffer intermedio, pero no se observaron grandes mejoras entre ambas soluciones.

Los dos primeros prototipos, se realizaron utilizando sockets bloqueantes (java.io), posteriormente se decidió implementar de nuevo los prototipos usando sockets no bloqueantes (ofrecidos por java.nio).

La diferencia entre un socket bloqueante, y uno no bloqueante es que el bloqueante bloquea la ejecución del hilo hasta que se recibe la información, en cambio usando un socket no bloqueante es posible atender diferentes entradas de información de forma simultanea. Lamentablemente el uso de sockets no bloqueantes no mejoró significativamente el rendimiento de la aplicación, después de éste primer acercamiento hacia el uso de sockets no bloqueantes y la poca mejora observada se dejo de investigar en esta línea. Los malos resultados obtenidos en estos prototipos apuntan a una implementación ineficaz de los sockets no bloqueantes o al bajo rendimiento de la pila IP ofrecida por Windows XP.

Para mejorar el rendimiento de estas pruebas, se realizaron compilaciones usando el Excelsior JET [23], con el objetivo de mejorar el rendimiento al no tener que intervenir la maquina virtual, para interpretar el lenguaje Java. Se observaron mejoras en el rendimiento pero éste aun no llegaba a ser el mínimo necesario para nuestra aplicación.

El ultimo prototipo para mejorar la capa de transmisión, fue la implementación en C de un reenviador de paquetes basado en el Packet Reflector, éste prototipo mejoró significativamente los resultados obtenidos, aún así todavía se generaban pequeñas pérdidas flujos de alta tasa (superiores a 10 Mbps).

A continuación se muestra una tabla donde se pueden observar los resultados obtenidos, ejecutando los prototipos en un equipo con sistema operativo Windows XP Professional.

Tabla 4.1 Resultados obtenidos con Windows XP Profesional.

Método	Flujo		Comentarios	Uso de CPU	Uso de memoria (RAM)	Valoración del usuario	
	Baja tasa	Alta tasa					
Java.io (único hilo)	Hasta 2 Mbps	3.5 Mbps	Flujos con tasas superiores a los 6 Mbps dan demasiadas pérdidas.	5 % - 6%	48 MB aprox.	Reproducción aceptable hasta 6Mbps A mayores tasas el flujo presenta pérdidas.	✓ ✗
Java.io (único hilo) + 5 MB buffer		-	Se consiguen visualizaciones aceptables con flujos de hasta 10 Mbps.	5 % - 6%	48 MB aprox.	Reproducción aceptable hasta 10 Mbps A mayores tasas el flujo presenta pérdidas.	✓ ✗
Java.nio (único hilo)			Flujos con tasas superiores a los 8 Mbps dan demasiadas pérdidas.	6 % - 7%	55 MB aprox.	Reproducción aceptable hasta 8 Mbps A mayores tasas el flujo presenta demasiadas pérdidas.	✓ ✗
Java.io (multi hilo)		8 Mbps (SD)	Todos los flujos presentan pérdidas.	N/D	N/D	No valido	✗
Java.nio (multi hilo)			Flujos con tasas superiores a los 8 Mbps dan demasiadas pérdidas.	15% - 20%	80 MB aprox.	Reproducción aceptable hasta 8 Mbps A mayores tasas el flujo presenta demasiadas pérdidas.	✓ ✗
Packet Reflector		-	Flujos con tasas superiores a los 15 Mbps dan demasiadas pérdidas.	4% - 8%	2,5 MB aprox.	Reproducción aceptable hasta 15 Mbps A mayores tasas el flujo presenta demasiadas pérdidas.	✓ ✗
Excelsior JET (Código nativo)		20 Mbps (HD)	Se aprecian mejoras de rendimiento respecto a la versión interpretada.	3% – 4%	29 MB aprox.	Reproducción aceptable hasta 10 Mbps A mayores tasas el flujo presenta demasiadas pérdidas.	✓ ✗

Debido a los malos resultados obtenidos al usar flujos de alta tasa (Aproximadamente 10Mbps), se decidieron realizar nuevas pruebas usando Linux como sistema operativo. La distribución usada en estas pruebas es una Ubuntu 6.06 LTS [24].

A continuación se muestran los resultados obtenidos:

Tabla 4.2 Resultados obtenidos con Linux

<table>
<tr><th rowspan="2">Método</th><th colspan="2">Flujo</th><th rowspan="2">Comentarios</th><th rowspan="2">Uso de CPU</th><th rowspan="2">Uso de memoria (RAM)</th><th rowspan="2">Valoración del usuario</th></tr>
<tr><th>Baja tasa</th><th>Alta tasa</th></tr>
<tr><td>Java.io (único hilo) + 5 MB buffer</td><td rowspan="3">Hasta 2 Mbps</td><td rowspan="3">3.5 Mbps
-
8 Mbps (SD)
-
20 Mbps (HD)</td><td>No se presentan pérdidas en ninguna de las pruebas.</td><td>5 % - 6%</td><td>N/D</td><td>Visualización correcta. ✓</td></tr>
<tr><td>Java.nio único y multi hilo</td><td colspan="4">Pruebas no realizadas al conseguir un resultado correcto con java.io</td></tr>
<tr><td>Packet Reflector</td><td>No se presentan pérdidas en ninguna de las pruebas.</td><td>5 % - 6%</td><td>2,5 MB</td><td>Visualización correcta. ✓</td></tr>
</table>

Si comparamos las tablas entre si, se observa claramente qué prototipos funcionan mejor en sistemas operativos Linux, que en sistemas Windows. Esto puede ser debido a la implementación de la pila IP del sistema operativo Windows (winsock.dll), o a otros factores externos a nosotros, como puede ser la implementación del controlador ofrecido por el fabricante de la tarjeta de red. Además de estas consideraciones generales, para el caso del Packet Reflector, las pruebas en Windows se realizaban sobre Cygwin, ya que el código se desarrollo para arquitecturas UNIX, y se ha de suponer que el uso de este emulador hace disminuir el rendimiento de la aplicación.

UNIX defiende el modelo de una red distribuida, donde existen muchos intercambios de información a través de una red. Es de suponer que la arquitectura interna de este sistema operativo estará optimizada para que sea muy eficiente en estos casos.

CAPÍTULO 5. PLANIFICACIÓN

En este capitulo se describen todas las tareas realizadas, el tiempo estimado y el que realmente se utilizó, finalmente se hace una valoración de la desviación entre ambos.

5.1. Planificación del proyecto

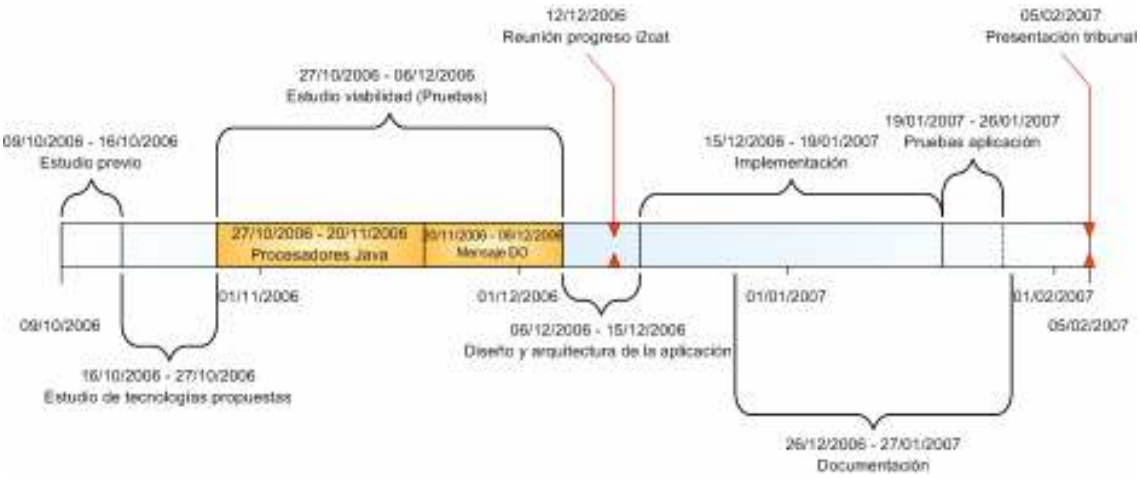


Fig. 5.1 Escala de tiempo del proyecto

El proyecto básicamente se dividió en 5 áreas:

- Estudio
- Diseño
- Implementación
- Pruebas
- Documentación

Estudio: En esta área se enmarcan todas las tareas relativas a la investigación y adquisición de nuevos conocimientos.

Diseño: Comprende todas las tareas relativas al diseño de la solución final, tanto a nivel de diseño de software, como a la definición de la arquitectura de red utilizada.

Implementación: Son todas las tareas de programación de los diseños propuestos.

Pruebas: Son las tareas encargadas de comprobar el correcto funcionamiento de la aplicación.

Documentación: Básicamente forman parte de esta tarea, la redacción de la memoria, y la preparación de reuniones y presentaciones.

A continuación se muestra una tabla donde se muestran de forma más detallada las tareas y el tiempo invertido en cada una de ellas.

Tabla 5.1 Tareas realizadas

Área	Tarea	Descripción	Tiempo estimado	Tiempo invertido
Estudio	Estudio previo	Familiarización con los conceptos SIP, MCU, Alta definición. Búsqueda de aplicaciones similares a la que se desarrollara en el proyecto.	25 horas	25 horas
Estudio	Proposición soluciones	Búsqueda de soluciones y en una reunión con el tutor se ponen en común entre ambos.	15 horas	15 horas
Estudio	Evaluación propuestas	Se establece un estudio de las tecnologías a usar, y se descartan las consideradas no viables.	35 horas	35 horas
Pruebas	Prototipos multimedia	Realización de prototipos del procesador multimedia con diferentes arquitecturas, y se realizaron pruebas para descartar los menos eficientes.	70 horas	90 horas
Pruebas	Prototipo control	Realización de modificaciones necesarias sobre Jain Sip, para poder añadirse el mensaje DO, una vez añadido se realizaron pruebas de funcionamiento, tanto en modo P2P como en modo Proxy.	50 horas	65 horas
Diseño	Arquitectura de red	Definición de la forma en la que los equipos iban a interactuar, tanto en el plano de señalización, como en el plano multimedia.	10 horas	15 horas
Diseño	Arquitectura de la aplicación	Definición de la arquitectura de la aplicación, y las líneas generales que debían seguir los módulos que formaban parte de ella.	25 horas	20 horas
Implementación	Gestor de salas	Desarrollo del diseño del gestor de salas, y pruebas de mapeos con Hibernate sobre una base de datos MySQL.	25 horas	20 horas
Implementación	Mcu Manager	Implementación de los métodos necesarios para la gestión global de la MCU.	15 horas	15 horas
Implementación	Módulo SIP	Implementación del módulo SIP añadiendo el protocolo de control que se había desarrollado en el prototipo.	45 horas	35 horas
Implementación	Procesador multimedia	Implementación del módulo encargado del procesado de los flujos multimedia.	35 horas	30 horas
Pruebas	Pruebas aplicación	Pruebas de la aplicación funcionando al completo, en diferentes entornos, y con diferentes cargas de trabajo.	25 horas	25 horas
Documentación	Memoria	Redactado de una pequeña memoria sobre el proyecto realizado.	140horas	150 horas

El total de horas invertidas para la realización de éste proyecto asciende a 540 horas, una dedicación un poco superior al valor que la EPSC considera óptimo para la realización del TFC.

Se observa una pequeña desviación respecto al tiempo de desarrollo previsto de aproximadamente 25 horas más respecto al estimado. Estos pequeños desajustes son debidos a que el desarrollo de los prototipos fue más costoso de lo previsto, pero finalmente se compensó en parte, ya que en el desarrollo de la aplicación definitiva se reutilizó gran cantidad de conocimientos adquiridos en estos prototipos.

CAPÍTULO 6. CONCLUSIONES Y TRABAJOS FUTUROS

En este capitulo se muestran los objetivos conseguidos con esta primera implementación de la MCU, y se indican los trabajos futuros a desarrollar sobre esta MCU. También se añade un estudio del impacto medioambiental de la aplicación y las conclusiones personales del autor.

6.1. Objetivos conseguidos

En el capítulo de diseño se enunciaron los requisitos mínimos que al inicio del proyecto se definieron, éstos eran:

- Utilización de señalización SIP estándar.
- Capacidad de control de una multiconferencia simultanea.
- Implementación de un sencillo procesador multimedia basado en Java.
- Diseño y desarrollo de un gestor de salas capaz de controlar más de una sesión de multiconferencia de forma simultánea.
- Compatibilidad e integración con el cliente SIP en desarrollo por el equipo de i2Cat.
- Limitación a tres usuarios por multiconferencia.

Opcionalmente y para mejorar la eficiencia de la aplicación se propuso la creación de un segundo procesador multimedia, basado en C.

Al dar por finalizado el desarrollo de este TFC, se consideran cumplidos estos requisitos, incluso el requisito opcional, así que podemos considerar que el desarrollo del proyecto ha sido el correcto y cumpliendo los plazos estimados para su ejecución.

6.2. Impacto medioambiental

A priori, es difícil de extraer del desarrollo de éste proyecto unos impactos medioambientales, pero si nos paramos a pensar más detenidamente, se puede llegar a la conclusión de que si que puede tener efectos sobre el medio ambiente.

La necesidad de hacer reuniones entre personas que no se encuentran situadas en la misma ciudad, país o continente, cada vez es mayor en un mundo globalizado. Estos desplazamientos suponen un grave daño al medio ambiente, ya que se necesitan grandes cantidades de energía para realizarlos. En la mayoría de casos, esta energía es generada a través de fuentes energéticas no renovables y por lo tanto contaminantes (gasolina en coche, electricidad en un tren, queroseno en un avión, etc…), esta generación realiza emisiones de gases contaminantes que afectan negativamente al medioambiente.

Por lo tanto, al realizar una videoconferencia de alta definición entre participantes de diferentes lugares, hace disminuir la necesidad de reuniones

entre personas y así conseguimos reducir las emisiones de gases contaminantes a la atmósfera.

Además, como efecto colateral debemos considerar la optimización en el uso de los recursos naturales, ya que al reducir el ancho de banda necesario para realizar una multiconferencia, también se contribuye a reducir la cantidad de fibra óptica a instalar. Con lo cual los minerales usados para su fabricación podrán gestionarse de manera más sostenible.

6.3. Conclusiones personales

Una vez finalizado el proyecto, me siento satisfecho del resultado obtenido de todo el trabajo realizado, ya que se han conseguido realizar todos los objetivos que se marcaron en un inicio.

Al inicio del desarrollo parecía casi imposible conseguir los resultados que finalmente se ha conseguido, por lo tanto todo el esfuerzo realizado durante estos cuatro meses ha valido la pena. Si no se hubiesen cumplido todos los requisitos, no hubiese sido un fracaso, pero no se podría valorar todo el trabajo que ha existido detrás de el.

A raíz de éste proyecto he conocido varias tecnologías que en la carrera no había usado, estas novedades han conseguido motivarme e intentar dar lo mejor de mí en el desarrollo de éste proyecto.

6.4. Trabajos futuros

Como trabajos futuros a realizar sobre esta primera implementación, a corto y medio plazo, se deberían realizar las siguientes tareas:

- Utilización de DVTS y XDVShow como fuentes y receptores de los flujos multimedia.
- Pruebas de rendimiento del módulo multimedia, con flujos de tasas superiores a 20 Mbps (No disponibles actualmente en i2Cat).
- Modificación del hardware del equipo usado como MCU, que ofrezca mayor capacidad de procesado, y añadir dos interficies más con capacidad Gigabit Ethernet, para mejorar el rendimiento de la aplicación.
- Creación del servicio Web encargado de la creación y administración de multiconferencias donde intervenga la MCU.

A largo plazo:

- Substitución de las tarjetas Gigabit Ethernet por una tarjeta con capacidad de 10Gbps y realizar pruebas para valorar su posible utilización en multiconferencias sobre *Ultragrid* (Tasas de 1Gbps).
- Utilización de la tecnología ofrecida por SAGE [25], para permitir al realizador la visualización a alta resolución de los videos de forma simultánea en más de una pantalla.

BIBLIOGRAFÍA

[1] – **Alta Definición:** Especificaciones sobre estándares de alta definición, 720p, 1080i, 1080p (en línea). [Última Consulta: 12 de diciembre de 2006]. URL: http://www.atsc.org/guide_default.html

[2] - **DVTS:** Información sobre software DVTS, diseño, descarga de versiones, etc… (en línea). [Última Consulta: 20 de diciembre de 2006]. URL: http://www.sfc.wide.ad.jp/DVTS/

[3] – **XDVShow:** Información sobre el reproductor XDVShow, manuales de instalación y uso. (en línea). [Última Consulta: 20 de diciembre de 2006]. URL: http://www.sfc.wide.ad.jp/DVTS/software/xdvshow/

[4] – **Ultragrid:** Web del proyecto Ultragrid, información sobre el software, y publicaciones relacionadas con HD sobre IP (en línea). [Última Consulta: 20 de diciembre de 2006]. URL: http://ultragrid.east.isi.edu/

[5] – **SIP:** RFC 3261 Session Initiation Protocol, documentación de referencia. (En línea). [Última Consulta: 10 de enero de 2007]. URL: http://www.ietf.org/rfc/rfc3261.txt

[6] – **HomeSip:** Web del proyecto HomeSip, información sobre el proyecto, publicaciones, y drafts presentados (en línea). [Última Consulta: 8 de enero de 2007]. URL: http://www.enseirb.fr/cosynux/HomeSIP/

[7] – **SIMPLE:** Web del grupo de trabajo del IETF para SIMPLE, información sobre el proyecto, RFC's y diferentes drafts. (en línea). [Última Consulta: 20 de enero de 2007]. URL: http://www.ietf.org/html.charters/simple-charter.html

[8] – **JAIN SIP:** Web oficial de JAIN SIP, información de la librería, código fuente, descarga de ficheros binarios (en línea). [Última Consulta: 16 de enero de 2007]. URL: https://jain-sip.dev.java.net/

[9] – **DO+XML:** Draft presentado por Telcordia, donde se define el mensaje SIP tipo DO. (en línea). [Última Consulta: 12 de enero de 2007]. URL: http://www.research.telcordia.com/iapp/draft-tsang-sip-appliances-do-00.txt

[10]- **SIP-SOAP:** Draft presentado por Ubiquity, se especifica el uso de SOAP en mensajes SIP (en línea). [Última Consulta: 21 de enero de 2007]. URL: http://quimby.gnus.org/internet-drafts/draft-deason-sip-soap-00.txt

[11]- **RTP Proxy:** Web de la aplicación, descarga del código fuente, e información sobre RTP Proxy (en línea). [Última Consulta: 29 de noviembre de 2006]. URL: http://ftp.iptel.org/pub/rtpproxy/

[12]- **Media Proxy:** Web oficial de los desarrolladores de Media Proxy, en su interior información sobre el proyecto, manuales, etc… (en línea). [Última Consulta: 29 de noviembre de 2006]. URL: http://www.ag-projects.com/

[13]- **Phyton:** Web lenguaje de programación phyton (en línea). [Última Consulta: 8 de enero de 2007]. URL: http://www.python.org/

[14]- **JAVA Sun:** Web oficial de Java Sun,(en línea). [Última Consulta: 20 de enero de 2007]. URL: http://java.sun.com/

[15]- **Java.nio:** Web con la API de Java.nio (en línea). [Última Consulta: 1 de enero de 2007]. URL: .http://java.sun.com/j2se/1.4.2/docs/api/java/nio/package-summary.html

[16]- **Java.io:** Web con la API de Java.io. (en línea). [Última Consulta: 25 de diciembre de 2006]. URL: http://java.sun.com/j2se/1.4.2/docs/api/java/io/package-summary.html

[17]- **Packet Reflector:** Publicación de documentación sobre la versión optimizada de Packet Reflector (en línea). [Última Consulta: 20 de enero de 2007]. URL: http://www.cesnet.cz/doc/techzpravy/2003/rtpreflector/

[18]- **Cesnet:** Web oficial del CESNET (En línea). [Última Consulta: 20 de diciembre de 2006]. URL: http://www.cesnet.cz/

[19]- **Sax:** Web oficial SAX, manuales de iniciación, API, etc… (en línea). [Última Consulta: 16 de enero de 2007]. URL: http://www.saxproject.org/

[20]- **Rum:** Código y documentación del software rum, precursor del packet reflector (En línea). [Última Consulta: 10 de enero de 2007]. URL: http://spirit.lboro.ac.uk/mug/mug.html

[21]- **Cygwin:** Web oficial Cygwin, manuales y descargas del software (en línea). [Última Consulta: 15 de enero de 2007]. URL: http://www.cygwin.com/

[22]- **VideoLan:** Web oficial VideoLan, manuales y descargas del software (en línea). [Última Consulta: 18 de enero de 2007]. URL: http://www.videolan.org/

[23]- **Excelsior JET:** Web oficial Excelsior JET, contiene información sobre el software (en línea). [Última Consulta: 1 de diciembre de 2006]. URL http://www.excelsior-usa.com/jet.html

[24]- **Ubuntu Linux:** Web oficial Ubuntu (En línea). [Última Consulta: 10 de enero de 2006]. URL http://www.ubuntu.com/

[25]- **SAGE:** Web oficial del Scalable Adaptive Graphics Environment, información sobre el proyecto, y publicaciones realizadas (en línea). [Última Consulta: 17 de enero de 2006]. URL: http://www.evl.uic.edu/cavern/sage/index.php

[26]- **Singleton:** Definición y ejemplos del patrón de diseño Singleton (en línea). [Última Consulta: 28 de enero de 2007]. URL: http://en.wikipedia.org/wiki/Singleton_pattern

[27]- **Evento i2Cat:** Documento informativo evento de multiconferencia i2Cat (en línea). [Última Consulta: 29 de enero de 2007]. URL: http://www.i2Cat.net/i2Cat/ImgsPortal/dancingq_demoplan.pdf

Algunos de los conceptos definidos, se han obtenido de: *Wikipedia – La enciclopedia libre* (En línea) [Última Consulta: 30 de enero de 2007]. URL: http://es.wikipedia.org

Información extra de videoconferencias de alta definición extraída de: *The HDoIP wiki - a reference guide to HDoIP* (En línea) [Última Consulta: 12 de diciembre de 2006]. URL: http://hdwiki.i2Cat.net/

AGRADECIMIENTOS

A mis excompañeros de trabajo, que siempre tuvieron un momento para echarme un cable, Roger Massa y Antonio Abajo.

Sin olvidarme de los compañeros de Mediacat, Alberto González y Pedro Díaz por su colaboración en la implementación y pruebas de los diferentes prototipos del software, a,Daniel Rodríguez por echarme una mano con la pila SIP, y finalmente Xavier Calvo, ya que sin su ayuda con el Word, no habría acabado esta memoria.

En especial agradecer el apoyo, la orientación, y la confianza depositada en mi, a mi tutor Toni Oller y a mi hermano Juan López.

ANEXOS

ANEXO A. ÍNDICE DE FIGURAS Y TABLAS

Figuras

Tablas

ANEXO B. ACRÓNIMOS

CESNET

CESNET es la institución encargada del desarrollo y mantenimiento de la red académica de la república checa.

FCC

Federal **C**ommunication **C**ommission, Comisión Federal de Comunicaciones de Estados Unidos, es un agencia independiente del gobierno estadounidense responsable de la regulación de las comunicaciones interestatales e internacionales por radio televisión y cable.

IETF

Internet **E**ngineering **T**ask **F**orce, en castellano Grupo de Trabajo en Ingeniería de Internet, es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, tales como transporte, encaminamiento, seguridad.

IP

Internet **P**rotocol, protocolo de Internet. Es la parte del Protocolo TCP/IP encargada del direccionamiento (identificación del origen y destino).

IPTV

Internet **P**rotocol **T**ele**V**ision (IPTV) se ha convertido en la denominación más común para los sistemas de distribución por suscripción de señales de televisión y/ó vídeo usando conexiones de banda ancha sobre el protocolo IP.

ISP

Internet **S**ervice **P**rovider, es el término genérico para representar a cualquier empresa u organización que provee servicios de acceso a Internet.

JAR

Java **AR**chive es un fichero ZIP que se usa para distribuir un conjunto de clases Java.

MIME

Multi-Purpose **I**nternet **M**ail **E**xtensions, son una serie de convenciones o especificaciones dirigidas a que se puedan intercambiar a través de Internet todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario.

MCU

Multipoint **C**ontrol **U**nit es un dispositivo (Hardware) o Software, encargado de realizar conmutaciones de flujos de datos.

NAT

Network **A**ddress **T**ranslator, es una aplicación no técnica y sencilla que determinado dispositivo o aplicación software es capaz de cambiar la dirección IP de origen o destino por otra dirección definida previamente. Se puede utilizar para dar salida a redes públicas a ordenadores que se encuentran con direccionamiento privado o para proteger máquinas públicas.

NHK

Nihon **H**ohsoh **K**yokai, es la emisora estatal Japonesa, emite tanto televisión como radio, se uso su nombre para definir un estándar de alta definición analógico.

NIST

National **I**nstitute for **S**tandards and **T**echnology, agencia del gobierno de EUA, que es responsable por establecer y proveer estándares de todo tipo.

NTSC

NTSC es un sistema de codificación y transmisión de televisión analógica desarrollada en Estados Unidos en torno a 1940, y que se emplea en la actualidad en la mayor parte de América y Japón, entre otros países. El nombre viene del comité de expertos que lo desarrolló, el **N**ational **T**elevision **S**ystem(s) **C**ommittee.

PAL

Phase **A**lternating **L**ine (línea alternada en fase). Éste es el nombre con que se designa al sistema de codificación empleado en la transmisión de señales de televisión en color en la mayor parte del mundo.

POSIX

Portable **O**perating **S**ystem **I**nterface, la **X** viene de UNIX como seña de identidad de la API.

RFC

Request **F**or **C**omments. Conjunto de archivos de carácter técnico donde se describen los estándares o recomendaciones de cualquier cosa. Entre otros los de la propia Internet.

RTP

Real **T**ime **P**rotocol, protocolo de Tiempo Real. Protocolo utilizado para la transmisión de información en tiempo real como por ejemplo audio y video en una videoconferencia.

SDP

Session **D**escription **P**rotocol. Protocolo utilizado para la descripción de sesiones.

SECAM

Séquentiel Couleur avec Mémoire en francés o "Color secuencial con memoria". Es un sistema para la codificación de televisión en color analógica.

SIP

Session Initiation Protocol, es un protocolo estándar para la inicialización de sesiones interactivas.

SMPTE

Society of Motion Picture and Television Engineers es una organización de profesionales encargada de definir estándares para la industria de la televisión y el cine.

SOAP

Simple Object Access Protocol es un protocolo estándar creado por el W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

UDP

User **D**atagram **P**rotocol (Protocolo de datagrama a nivel de usuario), perteneciente a la familia de protocolos TCP/IP, y esta situado en la capa de transporte.

UML

Unified **M**odelling **L**anguage, es el lenguaje de modelado de sistemas de software más conocido en la actualidad; aún cuando todavía no es un estándar oficial, está apoyado en gran manera por la OMG.

WAR

Web **A**pplication Archiver, es un fichero ZIP que se usa para distribuir un conjunto de clases Java

XML

e**X**tensible **M**arkup **L**anguage (lenguaje de marcado ampliable o extensible) desarrollado por el World Wide Web Consortium (W3C).

ANEXO C. MANUAL DE INSTALACIÓN Y USO

En este manual se mostraran los pasos a realizar para la ejecución del software desarrollado.

Requisitos recomendados:

- Pentium IV 3,2 Ghz
- 1 GB de memoria RAM DDRII 400Mhz
- Tarjeta de Red Gigabit Ethernet
- Disco duro IDE 80 GB
- Ubuntu Linux 6.06 LTS

Requisitos mínimos:

- Pentium IV 2,4 Ghz
- 512Mb de memoria RAM
- Tarjeta de Red Fast Ethernet
- Disco duro IDE 80 GB
- Microsoft Windows

Pasos previos:

Ubuntu Linux:
- Instalación de la JVM de SUN (http://java.sun.com).
- Instalación de Apache Ant (http://ant.apache.org/).
- Cliente CVS o Eclipse SDK.

Windows:
- Instalación de la JVM de SUN para Windows (http://java.sun.com).
- Instalación de Apache Ant para Windows (http://ant.apache.org/).
- Instalación de Cygwin (http://www.cygwin.com/).
- Cliente CVS o Eclipse SDK.

Una vez instalados las aplicaciones necesarias, se realizaran los preparativos para la instalación del software MCU.

En caso de tratarse de una plataforma Windows, es necesario la descarga de los módulos de Cygwin (Binutils, y GCC), en el caso de ser una Ubuntu Linux, se debería acceder al administrador de software “Synaptics” y instalar los paquetes Binutils y GCC 4, en caso de ser necesaria alguna dependencia extra el mismo administrador la instalar por defecto.

Finalizados los preparativos se realizara la descarga del módulo mcu del proyecto machine:

1º Paso descarga del módulo

La configuración del cliente CVS es la siguiente:

Host: broadband6.upc.es
Repository path: /home/cvs/
Connection type: extssh

Una vez configurado se ha de descargar el módulo situando en "/machine/mcu".

A modo de ejemplo mostramos unas capturas de la descarga de un módulo con el gestor de CVS integrado en eclipse.

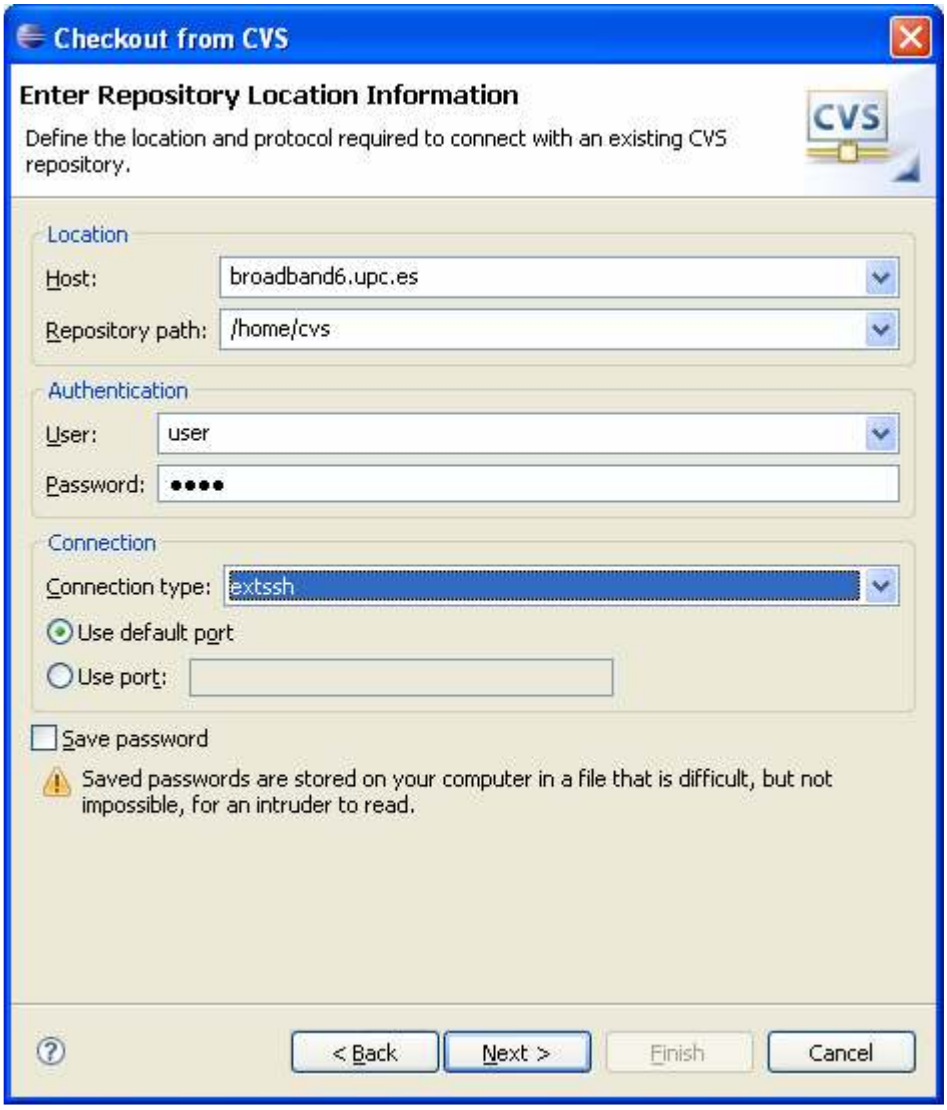


Fig C.1 Configuración CVS

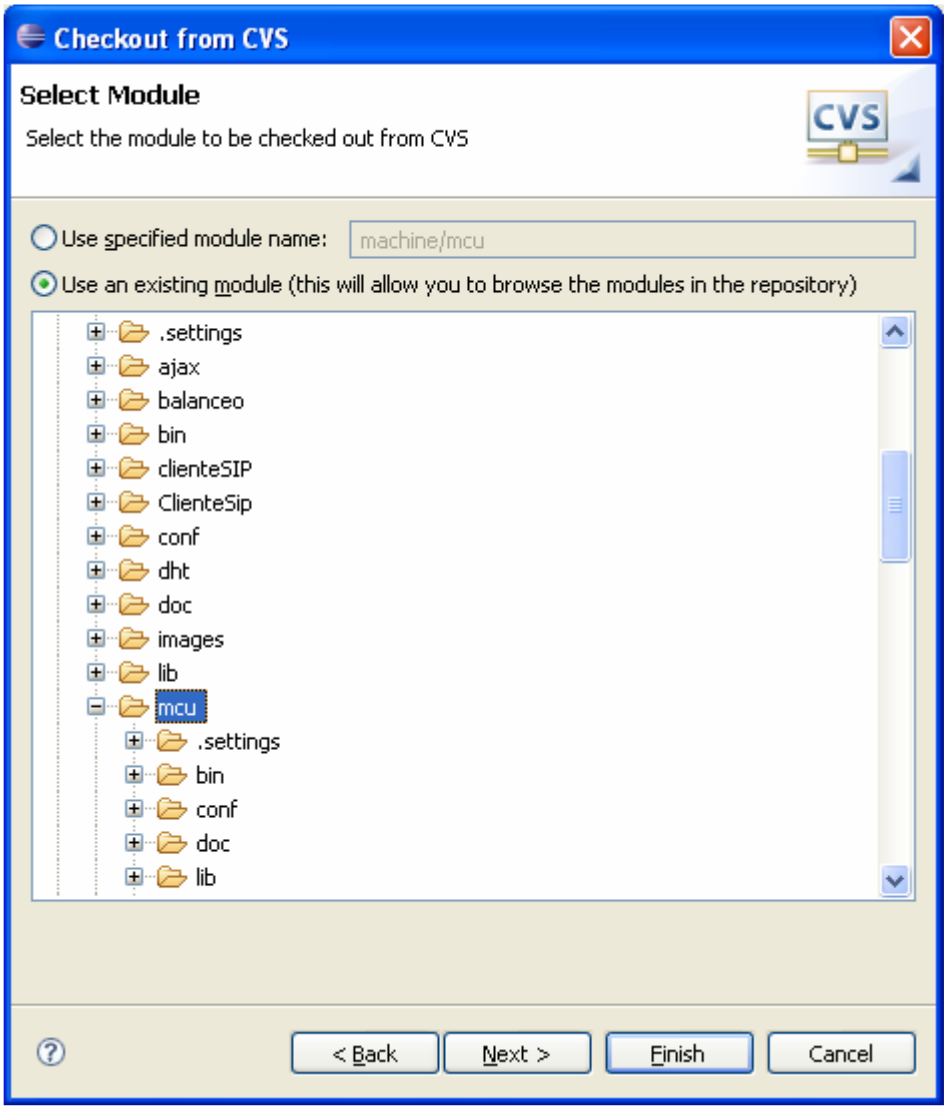


Fig. C.2 Selección módulo machine/mcu

2º Paso compilación del módulo multimedia

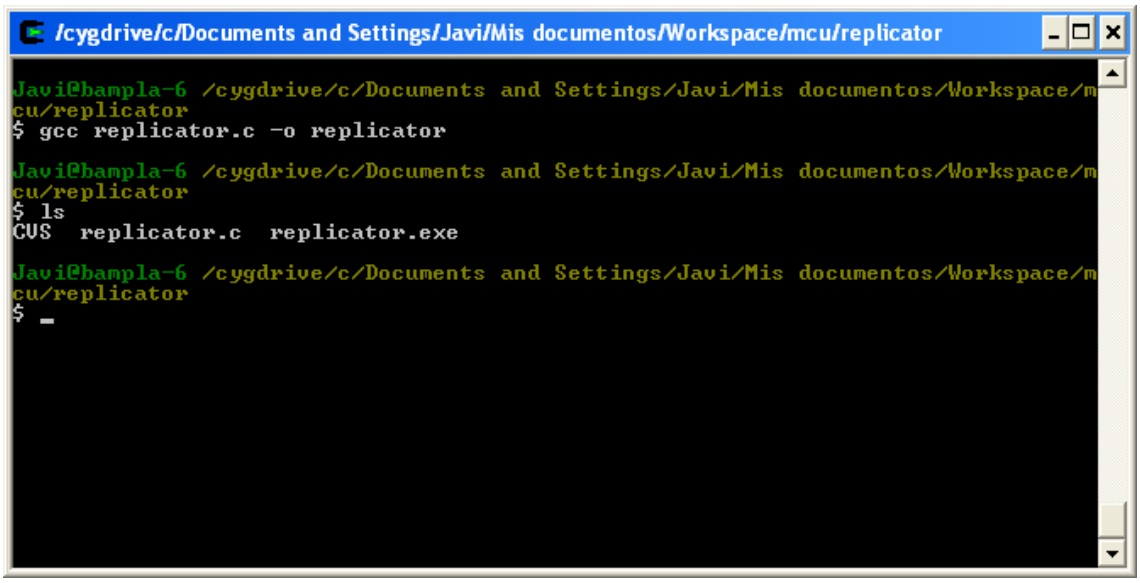
Para plataformas Windows iniciar una ventana de Terminal Cygwin (Existirá un icono en el escritorio), en caso de tratarse de una Ubuntu Linux, acceder al menú de aplicaciones -> sistema -> Terminal.

C.3 Terminal Cygwin

Una vez iniciado el Terminal acceder a la carpeta donde se haya descargado la MCU, en nuestro caso se encuentra instalado en "C:\Documents and Settings\Javi\Mis documentos\Workspace\mcu\"

Una vez en la raíz del proyecto acceder al módulo multimedia, situado en la carpeta "replicator".

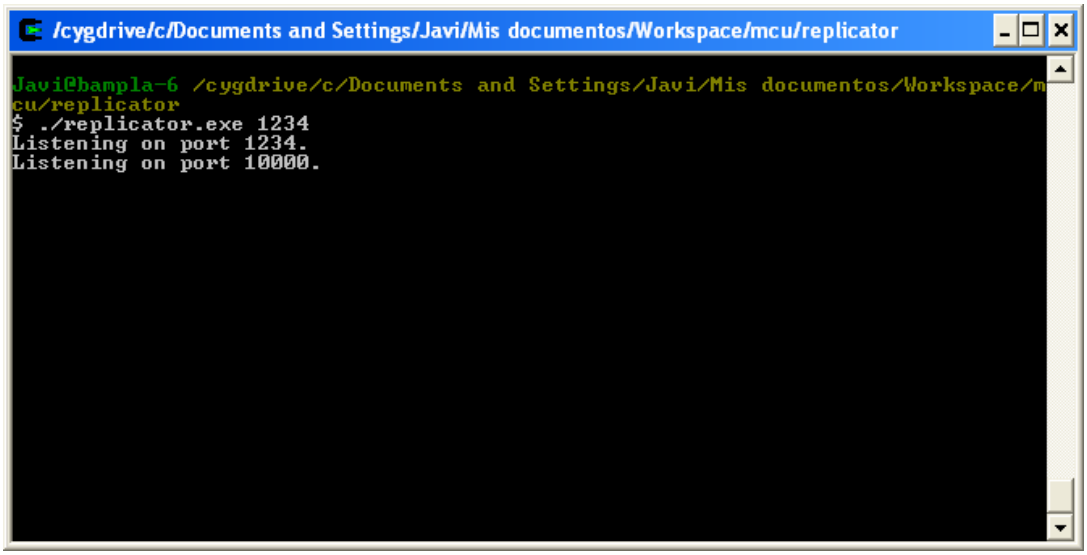
Finalmente realizar la compilación del módulo en cuestión, el comando a ejecutar es el siguiente: "gcc replicator.c –o replicator", aparecerá un nuevo fichero con extensión ".exe" en Windows o con permisos de ejecución en Linux, con el nombre "replicator".



C.4 Compilación del módulo multimedia

3º Paso ejecución del módulo multimedia

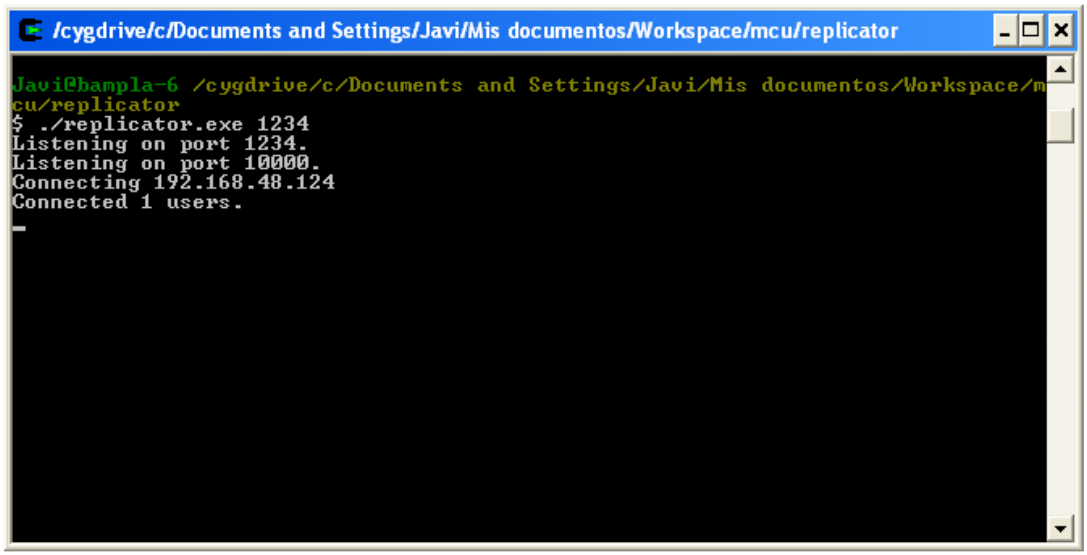
Desde el terminal ya utilizado, ejecutar el archivo generado, para ello utilizar el comando "./replicator.exe 1234" para Windows o "./replicator 1234" en Linux, la aplicación desarrollada es capaz de funcionar en cualquier puerto para la recepción de flujos multimedia, y utiliza el puerto 10000 UDP, para recepción de mensajes de control.



C.5 Modulo multimedia funcionando

Este paso es posible eliminarlo, si desde la propia MCU se ejecuta un proceso que llame a este ejecutable, para realizar tareas de debugging se ha decidido no utilizar temporalmente.

Al enviar un nuevo usuario información hacia la MCU, este módulo mostrara por pantalla la cantidad de usuarios simultáneos conectados y la dirección IP del usuario conectado.



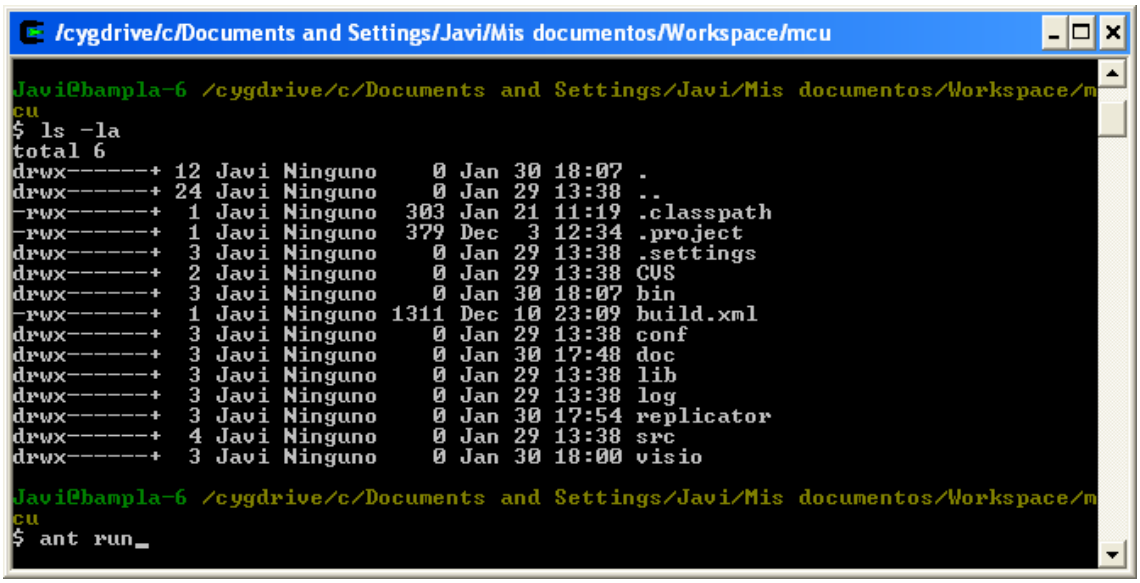
C.6 Usuario enviando flujo multimedia hacia la MCU

4º Paso ejecución de la MCU

Para automatizar la ejecución de la aplicación en un entorno de pruebas se desarrollo un fichero de tareas para Ant, donde se encuentran las tareas básicas de preparación, compilación, etc… del proyecto.

Para ejecutar la aplicación MCU, se ha de abrir un nuevo terminal, y desde el, acceder de nuevo a la carpeta raíz del proyecto.

Una vez allí, se ha de ejecutar el Ant pasando como parámetro la tarea "run", que es la encargada del lanzamiento de la aplicación, en caso de que sea necesario realizar alguna tarea previa, Ant la ejecutara de forma automática.



C.7 Ejecución mediante Apache Ant

Al tratarse de una aplicación independiente y gestionable remotamente, todas las acciones que se pueden realizar sobre la MCU, se han de realizar a través del protocolo de control. En la salida por pantalla aparecerán diferentes mensajes de control de errores, pero no será posible modificar parámetros de forma local.

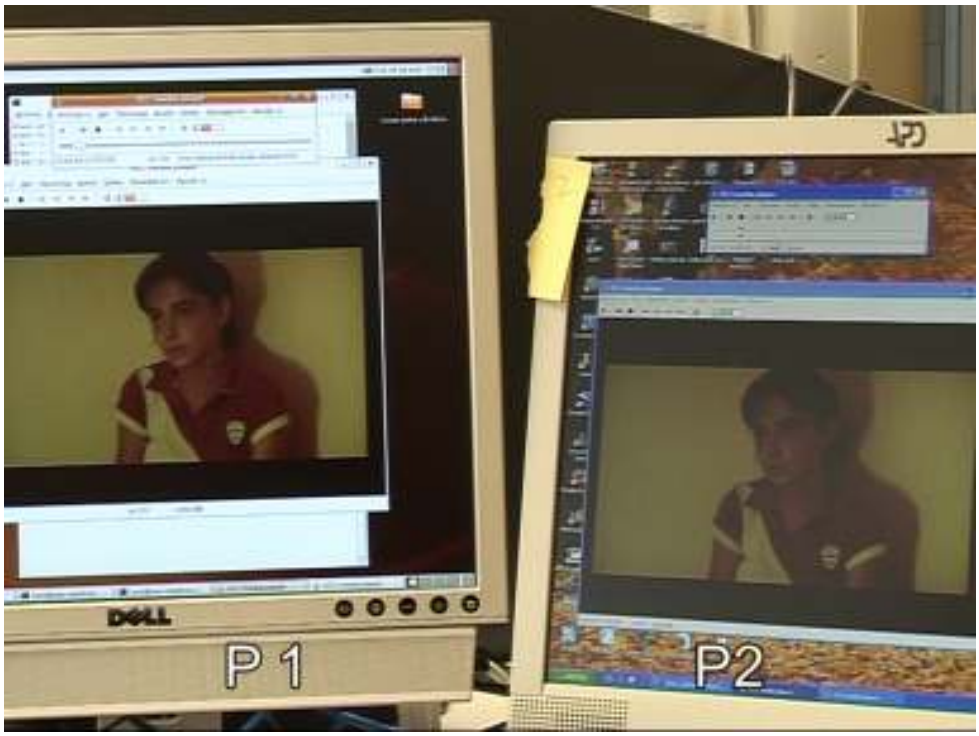
5º Paso realización de una multiconferencia

Una vez iniciada la aplicación se lanzaran automáticamente los reproductores de video.

El realizador recibirá los dos flujos de los participantes y enviara su propio flujo, a continuación se muestra una imagen (C.8) del un realizador en funcionamiento.

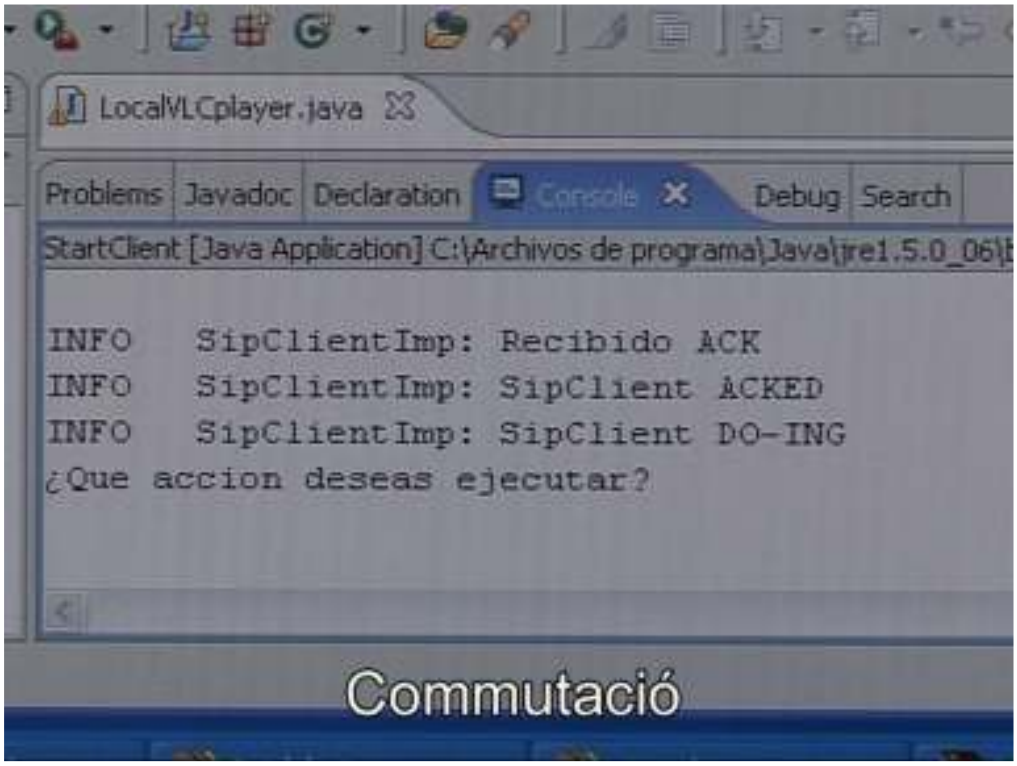
C.8 Realizador recibiendo todos los flujos de la multiconferencia

Los demás participantes recibirán el flujo seleccionado por el realizador, en esta imagen se observa como ambos participantes reciben el mismo flujo.



C.9 Participantes en una multiconferencia

Finalmente el usuario encargado de la realización puede ejecutar tareas sobre la MCU, por ejemplo conmutar, finalizar o expulsar a un participante.



C.10 Consola de gestión (realizador)