Redes Globales de Información: VideoConferencia

José Antonio Jiménez Pérez
superjozantonio@msn.com
Jorge Barbero Pérez
diesdief@hotmail.com

Introducción:

El trabajo que hemos llevado a cabo consiste en la implementación de un sistema de videoconferencia utilizando Java Media Framework (JMF).

El programa consiste en una serie de clases. Para compilar el programa es necesario tener instados el Java Developer Kit (JDK), a ser posible la última versión, y las librerías de Java Media Framework (JMF). Si tenemos instalados esos 2 programas (JDK y JMF), compilamos nuestro programa mediante la orden "javac Comunicador.java". Una vez compilado, para ejecutar el programa basta con escribir la orden "java Comunicador".

Uso del programa:

Una vez ejecutado el programa, en la ventana de consola aparecerá un log del estado del programa y en la ventana gráfica tendremos disponible el entorno gráfico.

La primera vez que se ejecuta el programa no habrá disponibles ninguno de los datos necesarios para realizar la comunicación:

Para configurar el programa, lo primero que haremos será introducir nuestra dirección IP, para ello, si no la conocemos, podemos pulsar el botón "generar" que hay junto al cuadro de edición para la dirección IP local, esto nos la generará automáticamente. Después de tener introducida nuestra dirección IP, debemos escribir cuales van a ser los puertos de transmisión de audio y de video, para ello pondremos los que deseemos teniendo en cuenta que no deben ser consecutivos. Acto seguido deberemos introducir la dirección IP del otro equipo con el que queramos establecer comunicación y los puertos de recepción de audio y video. Estos puertos de recepción deben ser los mismos que se han puesto en el otro equipo como de transmisión, teniendo en cuenta que no pueden ser los mismos que los que hemos puesto para transmitir. Una vez hecho todo esto, y para asegurarnos que se utilizan los datos correctos, deberemos cerrar el programa usando el botón de cerrar de la ventana gráfica. Una vez cerrado volveremos a ejecutar el programa, ahora ya, si todo ha ido bien, los datos de configuración estarán ya introducidos.

Para realizar una transmisión de audio y/o de video es necesario pulsar su respectivo botón teniendo en cuenta que ha de haber algún dispositivo de audio o de video seleccionado (según el caso). Para detener la transmisión basta con pulsar sobre el botón de detener transmisión deseado.

Para realizar una recepción de audio y/o video se pulsa sobre su respectivo botón. Para detener la recepción basta con pulsar sobre el botón de detener recepción deseado.

Para cambiar la calidad de la imagen hay que editar el archivo "Transmisor.java" y modificar el valor de la constante JPEGQ que por defecto viene como 0.1f (un 10%). Posteriormente hay que volver a compilar.

Pruebas realizadas:

Para el audio hemos realizado pruebas de comunicación entre un equipo provisto de ADSL 256 de la red de Telefónica y otro equipo provisto de Cable 128 de la red de Supercable obteniendo unos resultados aceptables. También hemos realizado pruebas en red de área local (Ethernet 10Mbps) obteniendo unos resultados bastante buenos.

Para el video no hemos podido realizar las pruebas que desearíamos haber realizado ya que solo disponíamos de 1 dispositivo de captura de video. El dispositivo de captura de video se trata de una cámara analógica de video conectada a una tarjeta sintonizadota Pinnacle Studio PCTV a través de un cable de video compuesto utilizando el driver WDM bajo Windows XP. El resultado obtenido en área local ha sido muy bueno, la transmisión se realizaba usando una resolución de 320x240 obteniendo una tasa de alrededor de 15fps. Utilizando una calidad JPEG del 10%, la imagen perdía notablemente nitidez, pero la tasa se mantenía en los 15fps reduciendo la cantidad de información transmitida a menos de 1/3 (de unos 1500kbps originales a unos 400kbps). Haciendo pruebas entre ADSL 256 y Supercable 128, la imagen empezaba a llegar con un pequeño retraso, pero se tragaba todo el ancho de banda.

Problemas conocidos:

El sonido se retrasa aproximadamente 1sg, hemos intentado arreglarlo pero no hemos conseguido nada, hemos consultado con otros compañeros y nos han dicho que el problema está en las librerías JMF por lo que no podemos solucionarlo.

Clases

Clase “Comunicador”:

Esta clase principal del programa, es la que contiene el método *main()*. Se encarga de construir el entorno gráfico, detectar los dispositivos de captura, y lanzar los transmisores y los receptores cuando el usuario lo indique.

Constantes de la clase:

```
private static final String TITULO = "Videoconferencia";
```
- Título de la ventana del programa.

```
private static final int ANCHOTEXTO = 15;
```
- Ancho de los cuadros de texto.

Variables miembro:

```
private JTextField ipLocal;
private JTextField puertoLocal;
private JTextField puertoVideoLocal;
private JTextField ipRemota;
private JTextField puertoRemoto;
private JTextField puertoVideoRemoto;
```
- Cuadros de texto para introducir valores.

```
private JButton IP;
```
- Botón para generar la dirección IP local automáticamente.

```
private JButton Iniciar;
private JButton Recibir;
private JButton IniciarVideo;
private JButton RecibirVideo;
```
- Botones para controlar el programa.

```
private JLabel Estado;
```
- Etiqueta que almacena los nombres de los autores (nosotros).

```
private Config config;
```
- Almacena la configuración del programa.

```
private boolean iniciado;
private boolean recibiendo;
private boolean iniciadoVideo;
private boolean recibiendoVideo;
```
- Flags que indican el estado del programa.

```
private Vector deviceList;
private Vector deviceVideoList;
```
- Vectores de dispositivos.

```
private CaptureDeviceInfo captureAudioDevice;
private CaptureDeviceInfo captureVideoDevice;
```
- Información de los dispositivos seleccionados.

```
private JList listaAudioDevice;
private JList listaVideoDevice;
```
- Listas de dispositivos.

```
private TransmisorAudio tx;
private ReceptorAudio rx;
private TransmisorVideo tx2;
private ReceptorVideo rx2;
```
- Transmisores y receptores.

Constructor de la clase:

Inicializa las variables miembro y genera el entorno gráfico.

```
public Comunicador()
{
	System.out.println(Misc.ind() + "Inicializando variables...");
	captureAudioDevice = null;
	captureVideoDevice = null;
	//ponemos título a la ventana
	System.out.println(Misc.ind() + "Poniendo titulo...");
	setTitle(TITULO);
	//creamos una configuración con las IPs local y remota y los puertos
	System.out.println(Misc.ind() + "Creando configuracion...");
	Misc.aumentarInd();
	config = new Config();
	Misc.reducirInd();
	//creamos una lista de dispositivos de captura de audio (formato de audio lineal)
	System.out.println(Misc.ind() + "Creando lista de dispositivos de captura de audio...");
	Misc.aumentarInd();
	deviceList = CaptureDeviceManager.getDeviceList(new
AudioFormat(AudioFormat.LINEAR, 44100, 8, 1));
	for (int i = 0; i < deviceList.size(); i++)
		System.out.println(Misc.ind() +
((CaptureDeviceInfo)deviceList.elementAt(i)).getName());
	//comprobamos que haya dispositivos de captura de audio
	//en caso de haberlo, seleccionamos el primero por defecto
	if (deviceList.size() > 0)
		captureAudioDevice = (CaptureDeviceInfo)deviceList.firstElement();
	else
	{
		captureAudioDevice = null;
		System.out.println(Misc.ind() + "No se han encontrado dispositivos de captura
de audio...");
	}
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Lista de dispositivos de captura de audio creada...");
	//creamos una lista de dispositivos de captura de video (formato de video RGB)
	System.out.println(Misc.ind() + "Creando lista de dispositivos de captura de video...");
	Misc.aumentarInd();
	deviceVideoList = CaptureDeviceManager.getDeviceList(new
VideoFormat(VideoFormat.RGB, new Dimension(352, 288), Format.NOT_SPECIFIED, null,
Format.NOT_SPECIFIED));
	for (int i = 0; i < deviceVideoList.size(); i++)
		System.out.println(Misc.ind() +
((CaptureDeviceInfo)deviceVideoList.elementAt(i)).getName());
	//comprobamos que haya dispositivos de captura de video
	//en caso de haberlo, seleccionamos el primero por defecto
	if (deviceVideoList.size() > 0)
		captureVideoDevice = (CaptureDeviceInfo)deviceVideoList.firstElement();
	else
	{
		captureVideoDevice = null;
		System.out.println(Misc.ind() + "No se han encontrado dispositivos de captura
de video...");
	}
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Lista de dispositivos de captura de audio creada...");

	//creamos el panel principal de la aplicación con su layout
	GridBagLayout gridBagLayout = new GridBagLayout();
	GridBagConstraints gbc;
```

```
System.out.println(Misc.ind() + "Creando panel principal...");
Misc.aumentarInd();
JPanel p = new JPanel();
p.setLayout(gridBagLayout);
//creamos el subpanel de configuración local
System.out.println(Misc.ind() + "Creando panel de configuracion local...");
Misc.aumentarInd();
JPanel panelConfiguracionLocal = crearPanelConfiguracionLocal();
Misc.reducirInd();
//aplicamos su configuración y lo insertamos en el panel principal
System.out.println(Misc.ind() + "Insertando panel de configuracion local...");
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.anchor = GridBagConstraints.CENTER;
gbc.fill = GridBagConstraints.BOTH;
gbc.insets = new Insets(10, 5, 0, 0);
((GridBagLayout)p.getLayout()).setConstraints(panelConfiguracionLocal, gbc);
p.add(panelConfiguracionLocal);
//creamos el subpanel de configuración remota
System.out.println(Misc.ind() + "Creando panel de configuracion remota...");
Misc.aumentarInd();
JPanel panelConfiguracionRemota = crearPanelConfiguracionRemota();
Misc.reducirInd();
//aplicamos su configuración y lo insertamos en el panel principal
System.out.println(Misc.ind() + "Insertando panel de configuracion remota...");
gbc = new GridBagConstraints();
gbc.gridx = 1;
gbc.gridy = 1;
gbc.weightx = 1.0;
gbc.weighty = 1.0;
gbc.anchor = GridBagConstraints.CENTER;
gbc.fill = GridBagConstraints.BOTH;
gbc.insets = new Insets(10, 5, 0, 0);
((GridBagLayout)p.getLayout()).setConstraints(panelConfiguracionRemota, gbc);
p.add(panelConfiguracionRemota);
//creamos el subpanel de botones
System.out.println(Misc.ind() + "Creando panel de botones...");
Misc.aumentarInd();
JPanel panelBotones = crearPanelBotones();
Misc.reducirInd();
//aplicamos su configuración y lo insertamos en el panel principal
System.out.println(Misc.ind() + "Insertando panel de botones...");
gbc = new GridBagConstraints();
gbc.gridx = 1;
gbc.gridy = 2;
gbc.weightx = 1.0;
gbc.weighty = 1.0;
gbc.anchor = GridBagConstraints.CENTER;
gbc.fill = GridBagConstraints.BOTH;
gbc.insets = new Insets(10, 5, 0, 0);
((GridBagLayout)p.getLayout()).setConstraints(panelBotones, gbc);
p.add(panelBotones);
//creamos el subpanel de etiquetas
System.out.println(Misc.ind() + "Creando panel de etiquetas...");
Misc.aumentarInd();
JPanel panelEstado = crearPanelEstado();
Misc.reducirInd();
```

```
        //aplicamos su configuración y lo insertamos en el panel principal
        System.out.println(Misc.ind() + "Insertando panel de etiquetas...");
        gbc = new GridBagConstraints();
        gbc.gridx = 1;
        gbc.gridy = 3;
        gbc.weightx = 1.0;
        gbc.weighty = 1.0;
        gbc.anchor = GridBagConstraints.CENTER;
        gbc.fill = GridBagConstraints.BOTH;
        gbc.insets = new Insets(10, 5, 0, 0);
        ((GridBagLayout)p.getLayout()).setConstraints(panelEstado, gbc);
        p.add(panelEstado);
        Misc.reducirInd();
        //aplicamos la configuración al panel principal        System.out.println(Misc.ind()  +
"Insertando panel principal...");
        getContentPane().add(p);
        listaAudioDevice.addMouseListener(this);
        listaVideoDevice.addMouseListener(this);
        addWindowListener(this);
   pack();
   //hacemos visible el panel principal
   setVisible(true);
        System.out.println(Misc.ind() + "Tareas de inicializacion finalizadas...");
}
```

Métodos de la clase:

Main():

Se ejecuta al iniciar el programa, se encarga de crear la clasel "*Comunicador*".

```
public static void main(String[] args) { new Comunicador(); }
```

crearEtiqueta():

Este método crea una etiqueta y la inserta en el panel:

```
private JLabel crearEtiqueta(JPanel p, int x, int y, String etiqueta)
{
        //creamos la etiqueta
        JLabel label = new JLabel(etiqueta);
        //configuramos la etiqueta
        System.out.println(Misc.ind() + "Insertando etiqueta " + etiqueta);
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.weightx = 0.0;
        gbc.weighty = 0.0;
        gbc.anchor = GridBagConstraints.EAST;
        gbc.fill = GridBagConstraints.NONE;
        gbc.insets = new Insets(5, 5, 0, 5);
        ((GridBagLayout)p.getLayout()).setConstraints(label, gbc);
        //insertamos la etiqueta
        p.add(label);
        //devolvemos la etiqueta
        return label;
}
```

crearCuadroTexto():

Este método crea un cuadro de texto, añade un notificador y lo inserta en el panel:

```
private JTextField crearCuadroTexto(JPanel p, int x, int y, int a, String texto)
{
	//creamos el cuadro de texto
	JTextField t = new JTextField(a);
	t.addActionListener(this);
	//configuramos el cuadro de texto
	System.out.println(Misc.ind() + "Insertando cuadro de texto " + texto);
	GridBagConstraints gbc = new GridBagConstraints();
	gbc.gridx = x;
	gbc.gridy = y;
	gbc.weightx = 0.0;
	gbc.weighty = 0.0;
	gbc.anchor = GridBagConstraints.WEST;
	gbc.fill = GridBagConstraints.NONE;
	gbc.insets = new Insets( 5,5,0,5);
	((GridBagLayout)p.getLayout()).setConstraints(t, gbc);
	//insertamos el cuadro de texto
	p.add(t);
	//colocamos texto por defecto al cuadro de texto
	t.setText(texto);
	//devolvemos el cuadro de texto
	return t;
}
```

crearBoton():

Este método crea un botón, añade un notificador y lo inserta en el panel:

```
private JButton crearBoton(JPanel p, int x, int y, String texto)
{
	//creamos el boton
	JButton t = new JButton(texto);
	t.addActionListener(this);
	//configuramos el boton
	System.out.println(Misc.ind() + "Insertando boton " + texto);
	GridBagConstraints gbc = new GridBagConstraints();
	gbc.gridx = x;
	gbc.gridy = y;
	gbc.weightx = 0.0;
	gbc.weighty = 0.0;
	gbc.anchor = GridBagConstraints.WEST;
	gbc.fill = GridBagConstraints.NONE;
	gbc.insets = new Insets( 5,5,0,5);
	((GridBagLayout)p.getLayout()).setConstraints(t, gbc);
	//insertamos el boton
	p.add(t);
	//devolvemos el boton
	return t;
}
```

crearPanelScroll():

Este método crea un panel con scroll y lo inserta en el panel:

```
private void crearPanelScroll(JPanel p, int x, int y, JList lista)
{
	//creamos el panel scroll
	JScrollPane panelScrollAudioDevices = new JScrollPane(lista,
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
	//configuramos el panel scroll
	System.out.println(Misc.ind() + "Insertando panel con scroll...");
	GridBagConstraints gbc = new GridBagConstraints();
	gbc.gridx = x;
	gbc.gridy = y;
	gbc.weightx = 0.0;
	gbc.weighty = 0.0;
	gbc.anchor = GridBagConstraints.CENTER;
	gbc.fill = GridBagConstraints.BOTH;
	gbc.insets = new Insets(10, 5, 0, 0);
	((GridBagLayout)p.getLayout()).setConstraints(panelScrollAudioDevices, gbc);
	p.add(panelScrollAudioDevices);
}
```

crearLista():

Este método crea una lista de dispositivos de captura:

```
private JList crearLista(Vector deviceList, String prototipo)
{
	//generamos un vector de nombres de dispositivo
	Vector listaNombresAudioDevice = new Vector();
	for (int i = 0; i < deviceList.size(); i++)
	listaNombresAudioDevice.add(((CaptureDeviceInfo)deviceList.elementAt(i)).getName())
;
	//creamos la lista de dispositivos
	JList listaAudioDevice = new JList(listaNombresAudioDevice);
	listaAudioDevice.addKeyListener(this);
	//asignamos una máscara para el ancho de la lista
	listaAudioDevice.setPrototypeCellValue(prototipo);
	//devolvemos la lista
	return listaAudioDevice;
}
```

crearPanelConfiguracionLocal():

Este método crea el panel para la configuración local del programa:

```
private JPanel crearPanelConfiguracionLocal()
{
	JPanel p = new JPanel();
	GridBagLayout gridBagLayout = new GridBagLayout();
	p.setLayout(gridBagLayout);
	//creamos una etiqueta para la IP local
	System.out.println(Misc.ind() + "Creando etiqueta de IP...");
	crearEtiqueta(p, 0, 0, "Dirección IP:");
	//creamos un cuadro de texto para la IP local
	System.out.println(Misc.ind() + "Creando cuadro de texto local...");
	ipLocal = crearCuadroTexto(p, 1, 0, ANCHOTEXTO, config.destino().ipLocal());
	//creamos un boton de búsqueda automática de la IP local
	System.out.println(Misc.ind() + "Creando boton de busqueda de IP...");
	IP = crearBoton(p, 2, 0, "Generar");
	//creamos una etiqueta para el puerto de transmisión de audio
	System.out.println(Misc.ind() + "Creando etiqueta de puerto...");
```

```
            crearEtiqueta(p, 0, 1, "Puerto de emisión de audio:");
            //creamos un cuadro de texto para el puerto de transmisión de audio
            System.out.println(Misc.ind() + "Creando cuadro de texto puerto...");
            puertoLocal = crearCuadroTexto(p, 1, 1, ANCHOTEXTO,
config.destino().puertoAudioLocal());
            //creamos una etiqueta para el puerto de transmisión de video
            System.out.println(Misc.ind() + "Creando etiqueta de puerto de video...");
            crearEtiqueta(p, 0, 3, "Puerto de emisión de video:");
            //creamos un cuadro de texto para el puerto de transmisión de video
            System.out.println(Misc.ind() + "Creando cuadro de texto puerto...");
            puertoVideoLocal = crearCuadroTexto(p, 1, 3, ANCHOTEXTO,
config.destino().puertoVideoLocal());
            //creamos una etiqueta para los dispositivos de captura de audio
            System.out.println(Misc.ind() + "Creando etiqueta de dispositivos de captura de
audio...");
            crearEtiqueta(p, 0, 4, "Dispositivos de captura de audio:");
            //creamos una lista de nombres de dispositivos de captura de audio
            System.out.println(Misc.ind() + "Creando lista de nombres de dispositivos de captura de
audio...");
            listaAudioDevice = crearLista(deviceList, "xxxxxxxxxxxxxxxxxxxxxxxxx");
            //creamos el panel con scroll para dispositivos de audio
            System.out.println(Misc.ind() + "Creando panel con scroll de dispositivos de audio...");
            crearPanelScroll(p, 0, 5, listaAudioDevice);
            //creamos una etiqueta para los dispositivos de captura de video
            System.out.println(Misc.ind() + "Creando etiqueta de dispositivos de captura de
video...");
            crearEtiqueta(p, 1, 4, "Dispositivos de captura de video:");
            //creamos una lista de nombres de dispositivos de captura de video
            System.out.println(Misc.ind() + "Creando lista de nombres de dispositivos de captura de
video...");
            listaVideoDevice = crearLista(deviceVideoList, "xxxxxxxxxxxxxxxxxxxxxxxxx");
            //creamos el panel con scroll para dispositivos de video
            System.out.println(Misc.ind() + "Creando panel con scroll de dispositivos de video...");
            crearPanelScroll(p, 1, 5, listaVideoDevice);
            //terminamos de configurar el panel
            System.out.println(Misc.ind() + "Poniendo titulo al panel...");
            TitledBorder titledBorder = new TitledBorder(new EtchedBorder(), "Configuración
local");
            p.setBorder(titledBorder);
            return p;
}
```

crearPanelConfiguracionRemota():

Este método crea el panel para la configuración del equipo remoto:

```
private JPanel crearPanelConfiguracionRemota()
{
            JPanel p = new JPanel();
            GridBagLayout gridBagLayout = new GridBagLayout();
            p.setLayout(gridBagLayout);
            //creamos una etiqueta para para la dirección IP de destino
            System.out.println(Misc.ind() + "Creando etiqueta de IP remota...");
            crearEtiqueta(p, 0, 0, "Dirección IP:");
            //creamos un cuadro de texto para la IP del destino
            System.out.println(Misc.ind() + "Creando cuadro de texto de IP remota...");
            ipRemota = crearCuadroTexto(p, 1, 0, ANCHOTEXTO, config.destino().ipRemota());
            //creamos una etiqueta para para el puerto de audio de destino
            System.out.println(Misc.ind() + "Creando etiqueta de puerto de audio remoto...");
            crearEtiqueta(p, 0, 1, "Puerto de recepcion de audio:");
            //creamos un cuadro de texto para el puerto de audio remoto
```

```
        System.out.println(Misc.ind() + "Creando cuadro de texto de puerto de audio remoto...");
        puertoRemoto = crearCuadroTexto(p, 1, 1, ANCHOTEXTO,
config.destino().puertoAudioRemoto());
        //creamos una etiqueta para para el puerto de video de destino
        System.out.println(Misc.ind() + "Creando etiqueta de puerto de video remoto...");
        crearEtiqueta(p, 0, 2, "Puerto de recepcion de video:");
        //creamos un cuadro de texto para el puerto de video remoto
        System.out.println(Misc.ind() + "Creando cuadro de texto puerto...");
        puertoVideoRemoto = crearCuadroTexto(p, 1, 2, ANCHOTEXTO,
config.destino().puertoVideoRemoto());
        //terminamos de configurar el panel
        System.out.println(Misc.ind() + "Poniendo titulo al panel...");
        TitledBorder titledBorder = new TitledBorder(new EtchedBorder(), "Configuración
remota");
        p.setBorder( titledBorder);
        return p;
}
```

crearPanelBotones():

Este método crea el panel con los botones de control del programa:

```
private JPanel crearPanelBotones()
{
        JPanel p = new JPanel();
        GridBagLayout gridBagLayout = new GridBagLayout();
        p.setLayout(gridBagLayout);
        //creamos un boton para iniciar la transmisión de audio
        System.out.println(Misc.ind() + "Creando boton de inicio transmision de audio...");
        Iniciar = crearBoton(p, 0, 0, "Iniciar transmisión audio");
        iniciado = false;
        //creamos un boton para iniciar la recepcion de audio
        System.out.println(Misc.ind() + "Creando boton de inicio recepcion de audio...");
        Recibir = crearBoton(p, 1, 0, "Iniciar recepción audio");
        recibiendo = false;
        //creamos un boton para iniciar la transmision de video
        System.out.println(Misc.ind() + "Creando boton de inicio transmision de video...");
        IniciarVideo = crearBoton(p, 0, 1, "Iniciar transmisión video");
        iniciadoVideo = false;
        //creamos un boton para iniciar la recepcion de video
        System.out.println(Misc.ind() + "Creando boton de inicio recepcion de video...");
        RecibirVideo = crearBoton(p, 1, 1, "Iniciar recepción video");
        recibiendoVideo = false;
        //terminamos de configurar el panel
        TitledBorder titledBorder = new TitledBorder(new EtchedBorder(), "Panel de botones");
        p.setBorder(titledBorder);
        return p;
}
```

crearPanelEstado():

Este método crea el panel de "*Acerca de…*":

```
private JPanel crearPanelEstado()
{
        JPanel p = new JPanel();
        GridBagLayout gridBagLayout = new GridBagLayout();
        p.setLayout(gridBagLayout);

        System.out.println(Misc.ind() + "Creando etiqueta de estado...");
        Estado = crearEtiqueta(p, 0, 0, "José Antonio Jiménez Pérez / Jorge Barbero Pérez");
        //terminamos de configurar el panel
```

```
        TitledBorder titledBorder = new TitledBorder(new EtchedBorder(), "Acerca de...");
        p.setBorder(titledBorder);
        return p;
}
```

actionPerformed():

Este método gestiona los eventos del programa tales como la pulsación de los botones o el cambio de los cuadros de texto:

```
public void actionPerformed(ActionEvent event)
{
        //Obtenemos la fuente del evento
        Object source = event.getSource();
        //Buscamos de donde proviene el evento
        //si proviene del boton Iniciar
        if (source == Iniciar)
        {
                //comprobamos si ya está iniciado
                if (!iniciado)
                {
                        //comprobamos si hay dispositivos de captura de audio
                        if (captureAudioDevice != null)
                        {
                                //actualizamos la etiqueta del boton
                                System.out.println(Misc.ind() + "Se ha pulsado el boton de
iniciar transmision de audio...");
                                Iniciar.setText("Detener transmisión audio");

                                //iniciamos la transmisión
                                iniciarTransmisionAudio();
                                iniciado = true;
                        }
                        else
                                System.out.println(Misc.ind() + "No hay ningun dispositivo de
captura de audio...");
                }
                else
                {
                        //actualizamos la etiqueta del boton
                        System.out.println(Misc.ind() + "Se ha pulsado el boton de detener
transmision de audio...");
                        Iniciar.setText("Iniciar transmisión audio");
                        //detenemos la transmisión
                        detenerTransmisionAudio();
                        iniciado = false;
                }
        }
        //si proviene del boton IniciarVideo
        else if (source == IniciarVideo)
        {
                //comprobamos si ya está iniciado
                if (!iniciadoVideo)
                {
                        //comprobamos si hay dispositivos de captura de video
                        if (captureVideoDevice != null)
                        {
                                //actualizamos la etiqueta del boton
                                System.out.println(Misc.ind() + "Se ha pulsado el boton de
iniciar transmision de video...");
                                IniciarVideo.setText("Detener transmisión video");
```

```
                                        //iniciamos la transmisión
                                        iniciarTransmisionVideo();
                                        iniciadoVideo = true;
                                }
                                else
                                        System.out.println(Misc.ind() + "No hay ningun dispositivo de
captura de video...");
                        }
                        else
                        {
                                //actualizamos la etiqueta del boton
                                System.out.println(Misc.ind() + "Se ha pulsado el boton de detener
transmision de video...");
                                IniciarVideo.setText("Iniciar transmisión video");
                                //detenemos la transmisión
                                detenerTransmisionVideo();
                                iniciadoVideo = false;
                        }
                }
                //si proviene del boton Recibir
                else if (source == Recibir)
                {
                        //comprobamos si ya está iniciado
                        if (!recibiendo)
                        {
                                //actualizamos la etiqueta del boton
                                System.out.println(Misc.ind() + "Se ha pulsado el boton de iniciar
recepcion de audio...");
                                Recibir.setText("Detener recepción audio");
                                //iniciamos la recepcion
                                iniciarRecepcionAudio();
                        }
                        else
                        {
                                //actualizamos la etiqueta del boton
                                System.out.println(Misc.ind() + "Se ha pulsado el boton de detener
recepcion de audio...");
                                Recibir.setText("Iniciar recepción audio");
                                //detenemos la recepcion
                                detenerRecepcionAudio();
                        }
                        recibiendo = !recibiendo;
                }
                //si proviene del boton RecibirVideo
                else if (source == RecibirVideo)
                {
                        //comprobamos si ya está iniciado
                        if (!recibiendoVideo)
                        {
                                //actualizamos la etiqueta del boton
                                System.out.println(Misc.ind() + "Se ha pulsado el boton de iniciar
recepcion de video...");
                                RecibirVideo.setText("Detener recepción video");
                                //iniciamos la recepcion
                                iniciarRecepcionVideo();
                        }
                        else
                        {
                                //actualizamos la etiqueta del boton
```

```
                        System.out.println(Misc.ind() + "Se ha pulsado el boton de detener
recepcion de video...");
                        RecibirVideo.setText("Iniciar recepción video");
                        //detenemos la recepcion
                        detenerRecepcionVideo();
                }
                recibiendoVideo = !recibiendoVideo;
        }
        //si proviene del cuadro de texto ipLocal
        else if (source == ipLocal)
        {
                //actualizamos la configuracion
                config.destino().ipLocal(ipLocal.getText().trim());
                System.out.println(Misc.ind() + "IP local cambiada a " +
config.destino().ipLocal());
        }
        //si proviene del cuadro de texto puertoLocal
        else if (source == puertoLocal)
        {
                //actualizamos la configuracion
                config.destino().puertoAudioLocal(puertoLocal.getText().trim());
                System.out.println(Misc.ind() + "Puerto local de audio cambiado a " +
config.destino().puertoAudioLocal());
        }
        //si proviene del cuadro de texto puertoVideoLocal
        else if (source == puertoVideoLocal)
        {
                //actualizamos la configuracion
                config.destino().puertoVideoLocal(puertoVideoLocal.getText().trim());
                System.out.println(Misc.ind() + "Puerto local de video cambiado a " +
config.destino().puertoVideoLocal());
        }
        //si proviene del cuadro de texto ipRemota
        else if (source == ipRemota)
        {
                //actualizamos la configuracion
                config.destino().ipRemota(ipRemota.getText().trim());
                System.out.println(Misc.ind() + "IP remota cambiada a " +
config.destino().ipRemota());
        }
        //si proviene del cuadro de texto puertoRemoto
        else if (source == puertoRemoto)
        {
                //actualizamos la configuracion
                config.destino().puertoAudioRemoto(puertoRemoto.getText().trim());
                System.out.println(Misc.ind() + "Puerto remoto de audio cambiado a " +
config.destino().puertoAudioRemoto());
        }
        //si proviene del cuadro de texto puertoVideoRemoto
        else if (source == puertoVideoRemoto)
        {
                //actualizamos la configuracion
                config.destino().puertoVideoRemoto(puertoVideoRemoto.getText().trim());
                System.out.println(Misc.ind() + "Puerto remoto de video cambiado a " +
config.destino().puertoVideoRemoto());
        }
        //si proviene del boton de generar la IP
        else if (source == IP)
        {
                System.out.println(Misc.ind() + "Obteniendo IP local...");
```

```
                try
                {
                        //Intentamos generar la dirección IP
                        String host = InetAddress.getLocalHost().getHostAddress();
                        //actualizamos el cuadro de texto implicado
                        ipLocal.setText(host);
                        //actualizamos la configuracion
                        config.destino().ipLocal(host);
                        System.out.println(Misc.ind() + "IP local cambiada a " + host);
                }
                //capturamos la excepcion
                catch(UnknownHostException e)
                {
                        System.out.println(Misc.ind() + "Fallo al obtener la IP local...");
                        ipLocal.setText("");
                        config.destino().ipLocal("");
                }
        }
}
```

iniciarTransmisionAudio():

Este método crea el transmisor de audio e inicia la transmisión:

```
private void iniciarTransmisionAudio()
{
        //creamos el transmisor de audio
        System.out.println(Misc.ind() + "Creando transmisor de audio");
        Misc.aumentarInd();
        tx = new TransmisorAudio(config.destino());
        Misc.reducirInd();
        //iniciamos el transmisor de audio
        System.out.println(Misc.ind() + "Iniciando transmisor de audio");
        Misc.aumentarInd();
        tx.start(captureAudioDevice);
        Misc.reducirInd();
        System.out.println(Misc.ind() + "Transmisor de audio iniciado");
}
```

detenerTransmisionAudio():

Este método destruye el transmisor de audio:

```
private void detenerTransmisionAudio()
{
        //detenemos el transmisor de audio
        System.out.println(Misc.ind() + "Destruyendo transmisor de audio");
        Misc.aumentarInd();
        tx.stop();
        Misc.reducirInd();
        tx = null;
        System.out.println(Misc.ind() + "Transmisor de audio destruido");
}
```

iniciarRecepcionAudio():

Este método crea el receptor de audio e inicia la recepción:

```
private void iniciarRecepcionAudio()
{
	//creamos el receptor de audio
	System.out.println(Misc.ind() + "Creando receptor de audio");
	Misc.aumentarInd();
	rx = new ReceptorAudio(config.destino());
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Receptor de audio iniciado");
}
```

detenerRecepcionAudio():

Este método destruye el receptor de audio:

```
private void detenerRecepcionAudio()
{
	//detenemos el receptor de audio
	System.out.println(Misc.ind() + "Destruyendo receptor de audio");
	Misc.aumentarInd();
	rx.close();
	rx = null;
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Receptor de audio destruido");
}
```

iniciarTransmisionVideo():

Este método crea el transmisor de video e inicia la transmisión:

```
private void iniciarTransmisionVideo()
{
	//creamos el transmisor de video
	System.out.println(Misc.ind() + "Creando transmisor de video");
	Misc.aumentarInd();
	tx2 = new TransmisorVideo(config.destino());
	Misc.reducirInd();
	//iniciamos el transmisor de video
	System.out.println(Misc.ind() + "Iniciando transmisor de video");
	Misc.aumentarInd();
	tx2.start(captureVideoDevice);
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Transmisor de video iniciado");
}
```

detenerTransmisionVideo():

Este método destruye el transmisor de video:

```
private void detenerTransmisionVideo()
{
	//detenemos el transmisor de video
	System.out.println(Misc.ind() + "Destruyendo transmisor de video");
	Misc.aumentarInd();
	tx2.stop();
	Misc.reducirInd();
	tx2= null;
	System.out.println(Misc.ind() + "Transmisor de video destruido");
}
```

iniciarRecepcionVideo():

Este método crea el receptor de video e inicia la recepción:

```
private void iniciarRecepcionVideo()
{
	//creamos el receptor de video
	System.out.println(Misc.ind() + "Creando receptor de video");
	Misc.aumentarInd();
	rx2 = new ReceptorVideo(config.destino());
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Receptor de video iniciado");
}
```

detenerRecepcionVideo():

Este método destruye el receptor de video:

```
private void detenerRecepcionVideo()
{
	//detenemos el receptor de video
	System.out.println(Misc.ind() + "Destruyendo receptor de video");
	Misc.aumentarInd();
	rx2.close();
	rx2 = null;
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Receptor de video destruido");
}
```

windowClosing():

Este método se genera al cerrar la ventana del programa. Se encarga de guardar la configuración del programa y detener los transmisores y receptores que estén activos:

```
public void windowClosing(WindowEvent event)
{
	//comprobamos si se está transmitiendo o recibiendo algo
	//en caso afirmativo lo detenemos
	if (iniciado)
		detenerTransmisionAudio();
	if (recibiendo)
		detenerRecepcionAudio();
	if (iniciadoVideo)
		detenerTransmisionVideo();
	if (recibiendoVideo)
		detenerRecepcionVideo();
	//actualizamos la configuración a partir de los cuadros de texto
	config.destino().ipLocal(ipLocal.getText().trim());
	config.destino().puertoAudioLocal(puertoLocal.getText().trim());
	config.destino().puertoVideoLocal(puertoVideoLocal.getText().trim());
	config.destino().ipRemota(ipRemota.getText().trim());
	config.destino().puertoAudioRemoto(puertoRemoto.getText().trim());
	config.destino().puertoVideoRemoto(puertoVideoRemoto.getText().trim());
	//guardamos la configuración en disco
	System.out.println(Misc.ind() + "Guardando configuracion...");
	Misc.aumentarInd();
	config.write();
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Configuracion guardada...");
	System.exit( 0);
}
```

keyReleased():

Este método se ejecuta al soltar una tecla del teclado:

```
public void keyReleased(KeyEvent event)
{
	//llamamos al método que gestiona esto
	seleccion(event.getSource());
}
```

mouseClicked():

Este método se ejecuta al hacer clic con el ratón:

```
public void mouseClicked(MouseEvent e)
{
	//llamamos al método que gestiona esto
	seleccion(e.getSource());
}
```

seleccion():

Este método es llamado por los 2 anteriores (*mouseClicked()* y *keyReleased()*) y se encarga de gestionar esos 2 eventos:

```
private void seleccion(Object source)
{
	//si el evento proviene de la lista de dispositivos de audio
	if(source == listaAudioDevice)
	{
		//capturamos el índice
		int index = listaAudioDevice.getSelectedIndex();
		//si hay elementos, actualizamos el dispositivo de captura de audio
		if(index != -1)
		{
			System.out.println(Misc.ind() + "Actualizando el dispositivo de captura
de audio...");
			captureAudioDevice = (CaptureDeviceInfo)deviceList.elementAt(index);
			Misc.aumentarInd();
			System.out.println(Misc.ind() + "Dispositivo de captura de audio
cambiado a " + captureAudioDevice.getName());
			Misc.reducirInd();
		}
	}
	//si el evento proviene de la lista de dispositivos de video
	else if(source == listaVideoDevice)
	{	//capturamos el índice
		int index = listaVideoDevice.getSelectedIndex();
		//si hay elementos, actualizamos el dispositivo de captura de video
		if(index != -1)
		{
			System.out.println(Misc.ind() + "Actualizando el dispositivo de captura
de video...");
			captureVideoDevice =
(CaptureDeviceInfo)deviceVideoList.elementAt(index);
			Misc.aumentarInd();
			System.out.println(Misc.ind() + "Dispositivo de captura de video
cambiado a " + captureVideoDevice.getName());
			Misc.reducirInd();
		}
	}
}
```

Métodos no utilizados:

```
public void windowClosed(WindowEvent event) {}
public void windowDeiconified(WindowEvent event) {}
public void windowIconified(WindowEvent event) {}
public void windowActivated(WindowEvent event) {}
public void windowDeactivated(WindowEvent event) {}
public void windowOpened(WindowEvent event) {}
public void keyPressed(KeyEvent event) {}
public void keyTyped(KeyEvent event) {}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
```

Clase “Transmisor”:

Esta clase se utiliza para ser heredada por otra clase que implemente una transmisión RTP ya que implementa los métodos necesarios y que son independientes de las particularidades de cada transmisor (origen, destino, tipo de datos).

Constantes de la clase:

private static final float JPEGQ = 0.1f;

- Almacena el valor de la calidad de la imagen del compresor JPEG.

Variables miembro:

protected Target destino;

- Guarda la configuración de puertos del programa, dependiendo del tipo de recepción, se utilizará unos puertos u otros.

protected MediaLocator locator;

- Guarda el localizador del dispositivo de captura.

protected Processor procesador;

- Guarda el procesador que devuelve la secuencia a enviar.

protected boolean fallo;

- Indica si ha habido un fallo.

protected Integer stateLock;

- Guarda el estado de bloqueo del procesador.

protected DataSource salida;

- Guarda la fuente de datos de salida.

protected RTPManager rtpManagers[];

- Tabla que guarda los manejadores RTP, en caso de haber varios.

protected int puertosLocales[];

- Tabla que guarda una lista de los puertos locales, en caso de haber varios.

Constructor de la clase:

Inicializa las variables miembro de la clase.

```
public Transmisor(Target destino)
{
	//inicializamos las variables
	System.out.println(Misc.ind() + "Inicializando las variables");
	this.destino = destino;
	procesador = null;
	locator = null;
	fallo = false;
	stateLock = new Integer(0);
	salida = null;
	rtpManagers = null;
}
```

Métodos de la clase:

stop():

Este método detiene la transmisión:

```
public void stop()
{
	synchronized(this)
	{
		//si existe el procesador
```

```
                if (procesador != null)
                {
                        //detenemos el procesador
                        System.out.println(Misc.ind() + "Detenemos el procesador");
                        procesador.stop();
                        System.out.println(Misc.ind() + "Cerramos el procesador");
                        //cerramos el procesador
                        procesador.close();
                        procesador = null;
                        //eliminamos los manejadores RTP
                        System.out.print(Misc.ind() + "Eliminamos los destinos");
                        for (int i = 0; i < rtpManagers.length; i++)
                        {
                                //eliminamos el destino
                                System.out.print(".");
                                rtpManagers[i].removeTargets("Session finalizada.");
                                //eliminamos el manejador
                                rtpManagers[i].dispose();
                        }
                        System.out.println();
                }
        }
}
```

crearProcesador():

Este método crea el procesador que será el encargado de procesar la entrada de datos y proporcionar una salida válida para la comunicación:

```
protected String crearProcesador()
{
        //comprobamos que existe un locator
        System.out.println(Misc.ind() + "Comprobando que existe el locator");
        if (locator == null)
                return "No hay seleccionado ningun dispositivo de captura de audio";
        DataSource ds;
        DataSource clone;
        //creamos una fuente de datos
        System.out.println(Misc.ind() + "Creando una fuente de datos para audio");
        try
        {
                ds = javax.media.Manager.createDataSource(locator);
        }
        //capturamos la excepcion
        catch (Exception e)
        {
                return "No se pudo encontrar una fuente de datos para audio";
        }
        //llamamos al metodo que crea el procesador a partir de la fuente de datos
        System.out.println(Misc.ind() + "Creando el procesador");
        try
        {
                procesador = javax.media.Manager.createProcessor(ds);
                procesador.addControllerListener(this);
        }
        //capturamos la excepcion
        catch (NoProcessorException npe)
        {
                return "No se pudo crear el procesador";
        }
        //capturamos otra excepcion
```

```
		catch (IOException ioe)
		{
			return "Excepcion de E/S al crear el procesador";
		}
		System.out.println(Misc.ind() + "Configurando el procesador");
		Misc.aumentarInd();
		//esperamos a que el procesador esté creado
		if (!waitForState(procesador, Processor.Configured))
		{
			Misc.reducirInd();
			return "No se pudo configurar el procesador";
		}
		Misc.reducirInd();
		//obtenemos la lista de las pistas
		System.out.println(Misc.ind() + "Obteniendo las pistas");
		TrackControl [] tracks = procesador.getTrackControls();
		//comprobamos que haya pistas
		System.out.println(Misc.ind() + "Comprobando que hay pistas");
		if (tracks == null || tracks.length < 1)
			return "No se pudo encontrar ninguna pista en el procesador";
		//creamos un descriptor de contenido
		System.out.println(Misc.ind() + "Creando un descriptor de contenido");
		ContentDescriptor cd = new ContentDescriptor(ContentDescriptor.RAW_RTP);
		//aplicamos el descriptor de contenido
		procesador.setContentDescriptor(cd);
		Format supported[];
		Format chosen;
		boolean atLeastOneTrack = false;
		//recorremos las pistas
		System.out.println(Misc.ind() + "Comprobando las pistas");
		Misc.aumentarInd();
		for (int i = 0; i < tracks.length; i++)
		{
			//obtenemos el formato de la pista
			System.out.println(Misc.ind() + "Obteniendo formato de la pista " + i);
			Format format = tracks[i].getFormat();
			//si la pista está activa
			if (tracks[i].isEnabled())
			{
				//obtenemos un formato válido
				System.out.println(Misc.ind() + "Comprobando que tenga un formato
valido");
				supported = tracks[i].getSupportedFormats();
				//comprobamos que contenga datos
				System.out.println(Misc.ind() + "Comprobando que contenga datos");
				if (supported.length > 0)
				{
					if (supported[0] instanceof VideoFormat)
					{
						//comprobamos las dimensiones del video
						chosen = checkForVideoSizes(tracks[i].getFormat(),
supported[0]);
					}
					else
						chosen = supported[0];
					//aplicamos el formato a la pista
					tracks[i].setFormat(chosen);
					System.err.println(Misc.ind() + "Pista " + i + " transmitiendo
como:");
					Misc.aumentarInd();
```

```
                                System.err.println(Misc.ind() + chosen);
                                Misc.reducirInd();
                                atLeastOneTrack = true;
                        }
                        else
                                tracks[i].setEnabled(false);
                }
                else
                        tracks[i].setEnabled(false);
        }
        Misc.reducirInd();
        //comprobamos que haya al menos una pista válida
        System.out.println(Misc.ind() + "Comprobando que hay alguna pista utilizable para
RTP");
        if (!atLeastOneTrack)
                return "No se pudo utilizar ninguna de las pistas para una transmisión RTP";
        //esperamos a que termine la configuración del procesador
        System.out.println(Misc.ind() + "Finalizando la creacion del procesador");
        Misc.aumentarInd();
        if (!waitForState(procesador, Controller.Realized))
        {
                Misc.reducirInd();
                return "No se pudo llevar a cabo el procesador...";
        }
        Misc.reducirInd();

        //cambiamos la calidad del formato JPG
        setJPEGQuality(procesador, JPEGQ);

        //obtenemos la salida del procesador
        System.out.println(Misc.ind() + "Obteniendo salida de datos");
        salida = procesador.getDataOutput();

        return null;
}
```

checkForVideoSizes():

Este método comprueba, en el caso de una secuencia de video, si tiene unas dimensiones adecuadas para la transmisión RTP:

```
private Format checkForVideoSizes(Format original, Format supported)
{
        int width, height;
        //obtenemos las dimensiones originales
        Dimension size = ((VideoFormat)original).getSize();
        Format jpegFmt = new Format(VideoFormat.JPEG_RTP);
        Format h263Fmt = new Format(VideoFormat.H263_RTP);
        //si se trata del formato JPG
        if (supported.matches(jpegFmt))
        {
                //el alto y el ancho han de ser divisibles por 8
                width = (size.width % 8 == 0 ? size.width : (int)(size.width / 8) * 8);
                height = (size.height % 8 == 0 ? size.height : (int)(size.height / 8) * 8);
        }
        //si se trata del formato H263
        else if (supported.matches(h263Fmt))
        {
                //seleccionamos un tamaño valido
                if (size.width < 128)
                {       width = 128;
```

```
                        height = 96;
                }
                else if (size.width < 176)
                {       width = 176;
                        height = 144;
                }
                else
                {       width = 352;
                        height = 288;
                }
        }
        //en otro caso
        else
                return supported;
        //devolvemos el nuevo formato de video
        return (new VideoFormat(null, new Dimension(width, height), Format.NOT_SPECIFIED,
null, Format.NOT_SPECIFIED)).intersects(supported);
}
```

setJPEGQuality():

Este método ajusta el nivel de calidad para el formato JPEG, a menor calidad, menos cantidad de información a transmitir:

```
protected void setJPEGQuality(Player p, float val)
{       //obtenemos los controles
        Control cs[] = p.getControls();
        QualityControl qc = null;
        //generamos un formato de video JPEG
        VideoFormat jpegFmt = new VideoFormat(VideoFormat.JPEG);
        //recorremos todos los controles
        for (int i = 0; i < cs.length; i++)
        {       //si estamos en el control de calidad
                if (cs[i] instanceof QualityControl && cs[i] instanceof Owned)
                {       Object owner = ((Owned)cs[i]).getOwner();
                        //comprobamos si se trata de un codec
                        if (owner instanceof Codec)
                        {       //generamos una tabla de formatos soportados
                                Format fmts[] =
((Codec)owner).getSupportedOutputFormats(null);
                                //recorremos la tabla
                                for (int j = 0; j < fmts.length; j++)
                                {       //si el formato es JPG
                                        if (fmts[j].matches(jpegFmt))
                                        {       //obtenemos el control de calidad
                                                qc = (QualityControl)cs[i];
                                                //ajustamos el control de calidad
                                                qc.setQuality(val);
                                                System.err.println("  - Ajustando calidad a " +
val + " sobre " + qc);
                                                break;
                                        }
                                }
                        }
                        if (qc != null)
                                break;
                }
        }
}
```

waitForState():

Este método crea una espera que dura hasta que finaliza la tarea de crear o iniciar el procesador:

```
protected synchronized boolean waitForState(Processor p, int state)
{
	//añadimos un evento escucha
	p.addControllerListener(new StateListener());
	fallo = false;
	//miramos si el procesador se está configurando
	if (state == Processor.Configured)
	{
		System.out.println(Misc.ind() + "Configurando...");
		//configuramos el procesador
		p.configure();
	}
	//miramos si el procesador se está realizando
	else if (state == Processor.Realized)
	{
		System.out.println(Misc.ind() + "Finalizando configuracion...");
		//realizamos el procesador
		p.realize();
	}
	//esperamos a que termine
	System.out.print(Misc.ind() + "Esperando");
	while (p.getState() < state && !fallo)
	{
		//comprobamos el estado de bloqueo
		System.out.print(".");
		synchronized (getStateLock())
		{
			//hacemos una pausa
			try
			{
				getStateLock().wait();
			}
			catch (InterruptedException ie)
			{
				return false;
			}
		}
	}
	System.out.println();
	if (fallo)
		return false;
	else
		return true;
}
```

getStateLock():

Este método devuelve el estado de bloqueo del procesador:

```
private Integer getStateLock()
{
	return stateLock;
}
```

setFailed():

Este método produce la activación de la variable que indica si ha habido un fallo:

```
void setFailed()
{
	fallo = true;
}
```

addTarget():

Este método añade un nuevo destino para el manejador RTP:

```
public void addTarget(int localPort, RTPManager mgr, String ip, int port)
{
	try
	{
		//creamos la sesion
		SessionAddress addr = new SessionAddress(InetAddress.getByName(ip), new
Integer( port).intValue());
		//añadimos la sesion
		mgr.addTarget(addr);
	}
	//capturamos la excepcion
	catch (Exception e)
	{
		e.printStackTrace();
	}
}
```

controllerUpdate():

Este método se ejecuta cada vez que se actualiza el controlador e indica si se ha producido un cambio en la longitud de la transmisión o un fin de esta:

```
public void controllerUpdate(ControllerEvent ce)
{
	//si se actualiza la duracion
	if (ce instanceof DurationUpdateEvent)
	{
		//obtenemos la duracion
		Time duration = ((DurationUpdateEvent) ce).getDuration();
		System.out.println(Misc.ind() + "duracion: " + duration.getSeconds());
	}
	//si obtenemos el final de la secuencia
	else
	if (ce instanceof EndOfMediaEvent)
	{
		System.out.println(Misc.ind() + "Fin de la comunicacion");
	}
}
```

update():

Hay 2 métodos update(), el primero se ejecuta al producirse una recepción mostrando la información en el log:

```
public void update(ReceiveStreamEvent event)
{
	//creamos un buffer
	StringBuffer sb = new StringBuffer();
	//si se recibe una cadena de inactividad
	if (event instanceof InactiveReceiveStreamEvent)
	{
```

```
                sb.append("Cadena recivida inactiva");
        }
        //si se recibe un adios
        else if (event instanceof ByeEvent)
        {
                sb.append("Hasta luego");
        }
        else
        {
                System.out.println(Misc.ind() + "Recibido evento de cadena: "+ event);
        }
        System.out.println(Misc.ind() + sb.toString());
}
```

El segundo se ejecuta al recibir un evento remoto:

```
public void update(RemoteEvent event)
{
        //si se recibe un informe de recepcion
        if (event instanceof ReceiverReportEvent)
        {
                //obtenemos el informe de recepcion
                ReceiverReport rr = ((ReceiverReportEvent) event).getReport();
                //creamos un buffer
                StringBuffer sb = new StringBuffer();
                //vamos dando formato al buffer
                sb.append("Recepcion");
                //si el informe existe
                if (rr != null)
                {
                        //obtenemos el participante
                        Participant participant = rr.getParticipant();
                        //concatenamos diversa informacion
                        if (participant != null)
                        {
                                sb.append(" desde " + participant.getCNAME());
                                sb.append(" ssrc = " + rr.getSSRC());
                        }
                        else
                        {
                                sb.append(" ssrc = " + rr.getSSRC());
                        }
                        System.out.println(Misc.ind() + sb.toString());
                }
        }
        else
        {
                System.out.println(Misc.ind() + "Evento remoto: " + event);
        }
}
```

Clase “TransmisorAudio”:

Esta clase se utiliza para transmitir el audio proveniente de un dispositivo de captura y enviarlo a través de una comunicación RTP. Esta clase hereda de la clase “*Transmisor*”, en la cual se encuentrar todas las variables miembro que utiliza.

Constructor de la clase:

Inicializa las variables miembro de la clase.

```
public TransmisorAudio(Target destino)
{
	//inicializamos las variables
	super(destino);
}
```

Métodos de la clase:

start():

Este método crea el procesador y el transmisor RTP y comienza la transmisión RTP:

```
public String start(CaptureDeviceInfo audio)
{
	String aux;
	//calculamos el locator del dispositivo
	System.out.println(Misc.ind() + "Calculando locator");
	locator = audio.getLocator();
	//creamos un procesador para el dispositivo
	System.out.println(Misc.ind() + "Creando procesador");
	Misc.aumentarInd();
	if ((aux = crearProcesador()) != null)
	{
		System.out.println(Misc.ind() + "No se pudo crear el procesador:");
		Misc.aumentarInd();
		System.out.println(aux);
		Misc.reducirInd();
		Misc.reducirInd();
		return aux;
	}
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Procesador creado");

	//creamos un transmisor
	System.out.println(Misc.ind() + "Creando transmisor");
	Misc.aumentarInd();
	if ((aux = crearTransmisor(destino)) != null)
	{
		System.out.println(Misc.ind() + "No se pudo crear el transmisor:");
		Misc.aumentarInd();
		System.out.println(Misc.ind() + aux);
		Misc.reducirInd();
		System.out.println(Misc.ind() + "Cerrando el procesador");
		Misc.reducirInd();
		//si no se pudo crear el transmisor, cerramos el procesador
		procesador.close();
		procesador = null;
		return aux;
	}
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Transmisor creado");
```

```
        //iniciamos el procesador
        System.out.println(Misc.ind() + "Iniciamos el procesador");
        procesador.start();
        return null;
}
```

crearTransmisor():

Este método crea el transmisor RTP:

```
private String crearTransmisor(Target destino)
{
        //creamos un buffer a partir del audio
        System.out.println(Misc.ind() + "Creando el buffer de salida de audio");
        PushBufferDataSource pbds = (PushBufferDataSource)salida;
        //obtenemos la lista de secuencias del buffer
        System.out.println(Misc.ind() + "Creando tabla de secuencias");
        PushBufferStream pbss[] = pbds.getStreams();
        //creamos una tabla de manejadores RTP
        System.out.println(Misc.ind() + "Creando tabla de manejadores RTP");
        rtpManagers = new RTPManager[pbss.length];
        //creamos una tabla de puertos locales de audio
        System.out.println(Misc.ind() + "Creando tabla de puertos locales");
        puertosLocales = new int[pbss.length];
        SessionAddress localAddr, destAddr;
        SendStream sendStream;
        int puerto;
        SourceDescription srcDesList[];
        //vamos recorriendo todas las secuencias
        System.out.println(Misc.ind() + "Recorremos todas las secuencias");
        Misc.aumentarInd();
        for (int i = 0; i < pbss.length; i++)
        {
                try
                {
                        //creamos un nuevo manejador RTP
                        System.out.println(Misc.ind() + "Creando manejador RTP " + i);
                        rtpManagers[i] = RTPManager.newInstance();
                        //añadimos un nuevo puerto
                        System.out.print(Misc.ind() + "Creando puerto local RTP: ");
                        puerto = (new Integer(destino.puertoAudioLocal()).intValue()) + 2 * i;
                        System.out.println(puerto);
                        puertosLocales[i] = puerto;
                        //calculamos la IP local
                        System.out.print(Misc.ind() + "Calculando la IP local: ");
                        localAddr = new
SessionAddress(InetAddress.getByName(destino.ipLocal()), puerto);
                        System.out.println(localAddr.getDataAddress());
                        //inicializamos el manejador RTP
                        System.out.println(Misc.ind() + "Inicializando manejador RTP");
                        rtpManagers[i].initialize(localAddr);
                        rtpManagers[i].addReceiveStreamListener(this);
                        rtpManagers[i].addRemoteListener(this);
                        //añadimos un destino al manejador RTP
                        System.out.println(Misc.ind() + "Anadiendo destino al manejador RTP");
                        int targetPort = (new Integer(destino.puertoAudioLocal())).intValue();
                        addTarget(puertosLocales[i], rtpManagers[i], destino.ipRemota(),
targetPort + 2 * i);
                        //Creamos una secuencia de envio
                        System.out.println(Misc.ind() + "Creando secuencia de envio RTP");
```

```
                        sendStream = rtpManagers[i].createSendStream(salida, i);
                        //iniciamos la secuencia de envio
                        sendStream.start();
                }
                //capturamos la excepción
                catch (Exception e)
                {
                        Misc.reducirInd();
                        e.printStackTrace();
                        return e.getMessage();
                }
        }
        Misc.reducirInd();
        System.out.println(Misc.ind() + "Todas las secuencias recorridas");
        return null;
}
```

Clase “TransmisorVideo”:

Esta clase se utiliza para transmitir el video proveniente de un dispositivo de captura y enviarlo a través de una comunicación RTP. Esta clase hereda de la clase “*Transmisor*”, en la cual se encuentrar todas las variables miembro que utiliza.

Constructor de la clase:

Inicializa las variables miembro de la clase.

```
public TransmisorVideo (Target destino)
{
	//inicializamos las variables
	super(destino);
}
```

Métodos de la clase:

start():

Este método crea el procesador y el transmisor RTP y comienza la transmisión RTP:

```
public String start(CaptureDeviceInfo audio)
{
	String aux;
	//calculamos el locator del dispositivo
	System.out.println(Misc.ind() + "Calculando locator");
	locator = video.getLocator();
	//creamos un procesador para el dispositivo
	System.out.println(Misc.ind() + "Creando procesador");
	Misc.aumentarInd();
	if ((aux = crearProcesador()) != null)
	{
		System.out.println(Misc.ind() + "No se pudo crear el procesador:");
		Misc.aumentarInd();
		System.out.println(aux);
		Misc.reducirInd();
		Misc.reducirInd();
		return aux;
	}
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Procesador creado");
	//creamos un transmisor
	System.out.println(Misc.ind() + "Creando transmisor");
	Misc.aumentarInd();
	if ((aux = crearTransmisor(destino)) != null)
	{
		System.out.println(Misc.ind() + "No se pudo crear el transmisor:");
		Misc.aumentarInd();
		System.out.println(Misc.ind() + aux);
		Misc.reducirInd();
		System.out.println(Misc.ind() + "Cerrando el procesador");
		Misc.reducirInd();
		//si no se pudo crear el transmisor, cerramos el procesador
		procesador.close();
		procesador = null;
		return aux;
	}
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Transmisor creado");
```

```
        //iniciamos el procesador
        System.out.println(Misc.ind() + "Iniciamos el procesador");
        procesador.start();
        return null;
}
```

crearTransmisor():

Este método crea el transmisor RTP:

```
private String crearTransmisor(Target destino)
{
        //creamos un buffer a partir del audio
        System.out.println(Misc.ind() + "Creando el buffer de salida de video");
        PushBufferDataSource pbds = (PushBufferDataSource)salida;
        //obtenemos la lista de secuencias del buffer
        System.out.println(Misc.ind() + "Creando tabla de secuencias");
        PushBufferStream pbss[] = pbds.getStreams();
        //creamos una tabla de manejadores RTP
        System.out.println(Misc.ind() + "Creando tabla de manejadores RTP");
        rtpManagers = new RTPManager[pbss.length];
        //creamos una tabla de puertos locales de audio
        System.out.println(Misc.ind() + "Creando tabla de puertos locales");
        puertosLocales = new int[pbss.length];
        SessionAddress localAddr, destAddr;
        SendStream sendStream;
        int puerto;
        SourceDescription srcDesList[];
        //vamos recorriendo todas las secuencias
        System.out.println(Misc.ind() + "Recorremos todas las secuencias");
        Misc.aumentarInd();
        for (int i = 0; i < pbss.length; i++)
        {
                try
                {
                        //creamos un nuevo manejador RTP
                        System.out.println(Misc.ind() + "Creando manejador RTP " + i);
                        rtpManagers[i] = RTPManager.newInstance();
                        //añadimos un nuevo puerto
                        System.out.print(Misc.ind() + "Creando puerto local RTP: ");
                        puerto = (new Integer(destino.puertoVideoLocal()).intValue()) + 2 * i;
                        System.out.println(puerto);
                        puertosLocales[i] = puerto;
                        //calculamos la IP local
                        System.out.print(Misc.ind() + "Calculando la IP local: ");
                        localAddr = new
SessionAddress(InetAddress.getByName(destino.ipLocal()), puerto);
                        System.out.println(localAddr.getDataAddress());
                        //inicializamos el manejador RTP
                        System.out.println(Misc.ind() + "Inicializando manejador RTP");
                        rtpManagers[i].initialize(localAddr);
                        rtpManagers[i].addReceiveStreamListener(this);
                        rtpManagers[i].addRemoteListener(this);
                        //añadimos un destino al manejador RTP
                        System.out.println(Misc.ind() + "Anadiendo destino al manejador RTP");
                        int targetPort = (new
Integer(destino.puertoVideoLocal()/*destino.puertoAudioRemoto()*/)).intValue();
                        addTarget(puertosLocales[i], rtpManagers[i], destino.ipRemota(),
targetPort + 2 * i);
                        //Creamos una secuencia de envio
                        System.out.println(Misc.ind() + "Creando secuencia de envio RTP");
```

```
                        sendStream = rtpManagers[i].createSendStream(salida, i);
                        //iniciamos la secuencia de envio
                        sendStream.start();
                }
                //capturamos la excepción
                catch (Exception e)
                {
                        Misc.reducirInd();
                        e.printStackTrace();
                        return e.getMessage();
                }
        }
        Misc.reducirInd();
        System.out.println(Misc.ind() + "Todas las secuencias recorridas");
        return null;
}
```

Subclase “StateListener”:

Esta subclase de la clase “*Transmisor*” se utiliza para mantener un escucha del estado del procesador.

Métodos de la clase:

controllerUpdate():

Este método se ejecuta al producirse un evento en el procesador:

```
public void controllerUpdate(ControllerEvent ce)
{
	//si se cierra el controlador, hacemos fallo
	if (ce instanceof ControllerClosedEvent)
		setFailed();
	//si se produce un evento del controlador
	if (ce instanceof ControllerEvent)
	{
		synchronized (getStateLock())
		{
			//informamos
			getStateLock().notifyAll();
		}
	}
}
```

Clase “Receptor”:

Esta clase se utiliza para ser heredada por otra clase que implemente una recepción RTP ya que implementa los métodos necesarios y que son independientes de las particularidades de cada receptor (origen, destino, tipo de datos).

Constantes de la clase:

protected static final int BUFFERLEN = 100;

- almacena el valor del tamaño del buffer de recepción en milisegundos.

Variables miembro:

protected Target destino;

- Guarda la configuración de puertos del programa, dependiendo del tipo de recepción, se utilizará unos puertos u otros.

protected RTPManager rtpManager;

- Guarda el manejador RTP que permite la comunicación.

private PlayerWindow playerWindow;

- Guarda la ventana de reproducción que se utilizará para reproducir lo que se reciba.

private Object dataSync;

- Objeto utilizado para llevar a cabo la sincronización.

private boolean dataReceived;

- Indica si se han recibido datos.

Constructor de la clase:

Inicializa las variables miembro de la clase.

```
public Receptor(Target destino)
{
	//inicializamos las variables
	System.out.println(Misc.ind() + "Inicializando variables");
	this.destino = destino;
	playerWindow = null;
	dataSync = new Object();
	dataReceived = false;
}
```

Métodos de la clase:

close():

Cierra el reproductor y libera los recursos asociados a este:

```
protected void close()
{
	System.out.println(Misc.ind() + "Cerrando ventana de reproduccion");
	//cerramos la ventana de reproduccion
	try { playerWindow.close(); }
	catch (Exception e) {}
	//cerramos todos el destino del manejador RTP
	System.out.println(Misc.ind() + "Cerrando sesion de recepcion");
	rtpManager.removeTargets("Cerrando sesion de recepcion");
	//eliminamos el manejador RTP
	rtpManager.dispose();
	rtpManager = null;
}
```

controllerUpdate():

Este método se ejecuta cada vez que se actualiza el controlador e inicializa la ventana de ejecución y la hace visible:

```
public synchronized void controllerUpdate(ControllerEvent ce)
{
	//creamos un reproductor
	Player p = (Player)ce.getSourceController();
	if (p == null)
		return;
	//si el evento está completamente generado
	if (ce instanceof RealizeCompleteEvent)
	{
		//inicializamos la ventana de reproduccion
		PlayerWindow pw = playerWindow;
		if (pw == null)
		{
			System.err.println(Misc.ind() + "Error interno");
			System.exit(-1);
		}
		pw.initialize();
		//visualizamos la ventana de reproduccion
		pw.setVisible(true);
		//iniciamos el reproductor
		p.start();
	}
	//si hay un evento de error del controlador
	if (ce instanceof ControllerErrorEvent)
	{
		//eliminamos el ControllerListener
		p.removeControllerListener(this);
		//cerramos la ventana de reproduccion
		PlayerWindow pw = playerWindow;
		if (pw != null)
		{
			pw.close();
			playerWindow = null;
		}
			System.err.println(Misc.ind() + "Error Interno de recepcion: " + ce);
	}
}
```

update():

Hay 3 métodos update(), el primero se ejecuta al producirse un evento remoto mostrando la información en el log del programa:

```
public void update(RemoteEvent event)
{
	//si se trata de un informe de entrega
	if (event instanceof ReceiverReportEvent)
	{
		//cogemos el informe
		ReceiverReport rr = ((ReceiverReportEvent) event).getReport();
		//creamos un buffer
		StringBuffer sb = new StringBuffer();
		//le vamos dando formato al buffer
		sb.append("Recepcion");
		//si hay informe
		if(rr != null)
		{
```

```
                        //obtenemos el participante
                        Participant participant = rr.getParticipant();
                        //añadimos información al buffer
                        if( participant != null)
                        {
                                sb.append(" desde " + participant.getCNAME());
                                sb.append(" ssrc = " + rr.getSSRC());
                        }
                        else
                        {
                                sb.append(" ssrc = " + rr.getSSRC());
                        }
                }
        }
        //si se trata de un informe de envio
        else if(event instanceof SenderReportEvent)
        {
                //cogemos el informe
                SenderReport sr = ((SenderReportEvent) event).getReport();
                //creamos un buffer
                StringBuffer sb = new StringBuffer();
                //le vamos dando formato al buffer
                sb.append("Emision");
                //si hay informe
                if( sr != null)
                {
                        //obtenemos el participante
                        Participant participant= sr.getParticipant();
                        //añadimos información al buffer
                        if( participant != null)
                        {
                                sb.append( " desde " + participant.getCNAME());
                                sb.append( " ssrc = " + sr.getSSRC());
                        }
                        else
                        {
                                sb.append( " ssrc = " + sr.getSSRC());
                        }
                }
        }
        //si no es nada de lo anterior
        else
        {
                System.out.println("Evento remoto: " + event);
        }
}
```

El segundo se ejecuta al iniciar sesión un nuevo participante:

```
public synchronized void update(SessionEvent evt)
{
        //si hay un participante nuevo
        if (evt instanceof NewParticipantEvent)
        {
                //creamos el participante nuevo
                Participant p = ((NewParticipantEvent)evt).getParticipant();
                System.err.println(Misc.ind() + "  - Un nuevo participante se ha unido: " +
p.getCNAME());
        }
}
```

El tercero se ejecuta al recibir una secuencia nueva mostrando la información en el log:

```
public synchronized void update(ReceiveStreamEvent evt)
{
	//obtenemos el origen de la cadena
	RTPManager rtpManager = (RTPManager)evt.getSource();
	//obtenemos el participante
	Participant participant = evt.getParticipant();
	//obtenemos la secuencia
	ReceiveStream stream = evt.getReceiveStream();
	//si la secuencia es un "Payload" nos salimos
	if (evt instanceof RemotePayloadChangeEvent)
	{
		System.err.println(Misc.ind() + "  - Recivido un evento RTP no soportado.");
		System.exit(0);
	}
	//si es una nueva cadena recibida
	else if (evt instanceof NewReceiveStreamEvent)
	{
		try
		{
			//obtenemos la secuencia
			stream = ((NewReceiveStreamEvent)evt).getReceiveStream();
			//obtenemos la fuente
			DataSource ds = stream.getDataSource();
			//segun sea una secuencia de control o no, escribimos en el log una cosa u otra
			RTPControl ctl =
(RTPControl)ds.getControl("javax.media.rtp.RTPControl");
			if (ctl != null)
				System.err.println(Misc.ind() + "  - Recibida nueva secuencia
RTP: " + ctl.getFormat());
			else
				System.err.println(Misc.ind() + "  - Recibida nueva secuencia
RTP");
			//si hay participante o no, escribimos una cosa u otra
			if (participant == null)
				System.err.println(Misc.ind() + "  - El emisor no pudo ser
identificado");
			else
				System.err.println(Misc.ind() + "  - La secuencia proviene de: " +
participant.getCNAME());
			//creamos un reproductor a partir de la fuente de datos
			Player p = javax.media.Manager.createPlayer(ds);
			if (p == null)
				return;
			//inicializamos el reproductor
			p.addControllerListener(this);
			p.realize();
			//generamos una ventana de reproducción
			PlayerWindow pw = new PlayerWindow(p, stream);
			playerWindow = pw;

			//Notificamos que se ha recibido una nueva secuencia
			synchronized(dataSync)
			{
				dataReceived = true;
				dataSync.notifyAll();
			}
		}
```

```
                    //capturamos la excepcion
                    catch (Exception e)
                    {
                            System.err.println(Misc.ind() + "Excepcion NewReceiveStreamEvent " +
e.getMessage());
                            return;
                    }
            }
            //si es parte de una cadena anterior
            else if (evt instanceof StreamMappedEvent)
            {
                    //si hay una secuencia y con una fuente de datos
                    if (stream != null && stream.getDataSource() != null)
                    {
                            //usamos la fuente de datos fuente de datos
                            DataSource ds = stream.getDataSource();
                            //Obtenemos el formato
                            RTPControl ctl =
(RTPControl)ds.getControl("javax.media.rtp.RTPControl");
                            System.err.println(Misc.ind() + "  - La secuencia anteriormente no
identificada ");
                            //escribimos el formato
                            if (ctl != null)
                                    System.err.println(Misc.ind() + "  - " + ctl.getFormat());
                            System.err.println(Misc.ind() + "  - ha sido identificada como de: " +
participant.getCNAME());
                            //creamos un manejador de sesion RTP
                            RTPSessionMgr rtpMgr = (RTPSessionMgr)evt.getSessionManager();
                            //obtenemos una sesion de direccion
                            SessionAddress addr = rtpMgr.getRemoteSessionAddress();
                    }
            }
            //si es una despedida
            else if (evt instanceof ByeEvent)
            {
                    //crea,ps im buffer
                    StringBuffer sb = new StringBuffer();
                    //aplicamos un formato
                    sb.append("ADIOS");
                    //generamos una razón
                    String reason = ((ByeEvent)evt).getReason();
                    //añadimos al buffer información adicional
                    sb.append(" desde " + participant.getCNAME());
                    sb.append(" ssrc = " + stream.getSSRC());
                    sb.append(" razon = "' + reason + '"');

                    if (playerWindow != null)
                    {
                            playerWindow.close();
                            playerWindow = null;
                    }
            }
}
```

Clase “ReceptorAudio”:

Esta clase se utiliza para recibir y reproducir el audio proveniente de la comunicación RTP. Esta clase hereda de la clase “*Receptor*”, en la cual se encuentrar todas las variables miembro que utiliza.

Constructor de la clase:

Inicializa las variables miembro de la clase y añade el origen de la comunicación.

```
public ReceptorAudio(Target destino)
{
	//inicializamos las variables
	super(destino);
	//añadimos el destino remoto
	System.out.println(Misc.ind() + "Creando destino");
	Misc.aumentarInd();
	addTarget();
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Destino creado");
}
```

Métodos de la clase:

addTarget():

Añade un origen de la comunicación:

```
public void addTarget()
{
	try
	{
		InetAddress ipAddr;
		SessionAddress localAddr;
		SessionAddress destAddr;
		//creamos un manejador RTP
		System.out.println(Misc.ind() + "Creando el manejador RTP");
		rtpManager = RTPManager.newInstance();
		rtpManager.addSessionListener(this);
		rtpManager.addReceiveStreamListener(this);
		rtpManager.addRemoteListener(this);
		//Obtenemos la dirección IP des destino a partir de una cadena
		System.out.print(Misc.ind() + "Obteniendo IP del destino: ");
		ipAddr = InetAddress.getByName(destino.ipRemota());
		System.out.println(ipAddr.getHostAddress());
		//Obtenemos el puerto remoto
		System.out.print(Misc.ind() + "Obteniendo puerto remoto: ");
		int remotePort= new Integer(destino.puertoAudioRemoto()).intValue();
		System.out.println(remotePort);
		//Creamos una sesion local usando el puerto remoto
		System.out.println(Misc.ind() + "Creando sesion local: " + destino.ipLocal() + ":"
+ remotePort);
		localAddr = new SessionAddress(InetAddress.getByName(destino.ipLocal()),
remotePort);
		//Creamos una sesión remota usando el puerto remoto
		System.out.println(Misc.ind() + "Creando sesion remota: " +
ipAddr.getHostAddress() + ":" + remotePort);
		destAddr = new SessionAddress(ipAddr, remotePort);

		//inicializamos el manejador RTP
		System.out.println(Misc.ind() + "Inicializando el manejador RTP");
```

```
                    rtpManager.initialize(localAddr);
                    //Cambiamos el tamaño del buffer
                    System.out.println(Misc.ind() + "Cambiando el tamano del buffer a " +
BUFFERLEN + "msg");
                    BufferControl bc =
(BufferControl)rtpManager.getControl("javax.media.control.BufferControl");
                    if (bc != null)
                    {
                              bc.setBufferLength(BUFFERLEN);
                    }
                    //añadimos la dirección del destino al manejador RTP
                    System.out.println(Misc.ind() + "Anadiendo destino");
                    rtpManager.addTarget(destAddr);
          }
          //capturamos la excepcion
          catch (Exception e)
          {
                    System.err.println(Misc.ind() + "No se pudo crear la sesion RTP: " +
e.getMessage());
                    return;
          }
}
```

Clase “ReceptorVideo”:

Esta clase se utiliza para recibir y reproducir el video proveniente de la comunicación RTP. Esta clase hereda de la clase “*Receptor*”, en la cual se encuentrar todas las variables miembro que utiliza.

Constructor de la clase:

Inicializa las variables miembro de la clase y añade el origen de la comunicación.

```
public ReceptorVideo(Target destino)
{
	//inicializamos las variables
	super(destino);
	//añadimos el destino remoto
	System.out.println(Misc.ind() + "Creando destino");
	Misc.aumentarInd();
	addTarget();
	Misc.reducirInd();
	System.out.println(Misc.ind() + "Destino creado");
}
```

Métodos de la clase:

addTarget():

Añade un origen de la comunicación:

```
public void addTarget()
{
	try
	{
		InetAddress ipAddr;
		SessionAddress localAddr;
		SessionAddress destAddr;
		//creamos un manejador RTP
		System.out.println(Misc.ind() + "Creando el manejador RTP");
		rtpManager = RTPManager.newInstance();
		rtpManager.addSessionListener(this);
		rtpManager.addReceiveStreamListener(this);
		rtpManager.addRemoteListener(this);
		//Obtenemos la dirección IP des destino a partir de una cadena
		System.out.print(Misc.ind() + "Obteniendo IP del destino: ");
		ipAddr = InetAddress.getByName(destino.ipRemota()/*senderAddress*/);
		System.out.println(ipAddr.getHostAddress());
		//Obtenemos el puerto remoto
		System.out.print(Misc.ind() + "Obteniendo puerto remoto: ");
		int remotePort= new
Integer(destino.puertoVideoRemoto()/*senderPort*/).intValue();
		System.out.println(remotePort);
		//Creamos una sesion local usando el puerto remoto
		System.out.println(Misc.ind() + "Creando sesion local: " + destino.ipLocal() + ":"
+ remotePort/*local_port*/);
		localAddr = new SessionAddress(InetAddress.getByName(destino.ipLocal()),
remotePort/*local_port*/);
		//Creamos una sesión remota usando el puerto remoto
		System.out.println(Misc.ind() + "Creando sesion remota: " +
ipAddr.getHostAddress() + ":" + remotePort);
		destAddr = new SessionAddress(ipAddr, remotePort);
		//inicializamos el manejador RTP
		System.out.println(Misc.ind() + "Inicializando el manejador RTP");
```

```
                    rtpManager.initialize(localAddr);
                    //Cambiamos el tamaño del buffer
                    System.out.println(Misc.ind() + "Cambiando el tamano del buffer a " +
BUFFERLEN + "msg");
                    BufferControl bc =
(BufferControl)rtpManager.getControl("javax.media.control.BufferControl");
                    if (bc != null)
                    {
                              bc.setBufferLength(BUFFERLEN);
                    }
                    //añadimos la dirección del destino al manejador RTP
                    System.out.println(Misc.ind() + "Anadiendo destino");
                    rtpManager.addTarget(destAddr);
          }
          //capturamos la excepcion
          catch (Exception e)
          {
                    System.err.println(Misc.ind() + "No se pudo crear la sesion RTP: " +
e.getMessage());
                    return;
          }
}
```

Subclase “PlayerWindow”:

Esta subclase de la clase “*Recepcion*” se utiliza para mostrar una ventana de reproduccín.

Variables miembro:

```
private Player player;
private ReceiveStream stream;
private String senderPort;
private String senderAddress;
```

Constructor de la clase:

Inicializa las variables miembro de la clase.

```
public PlayerWindow(Player p, ReceiveStream strm)
{
        player = p;
        stream = strm;
        //fijamos el título
        super.setTitle( "Ventana de reproducción");
}
```

Métodos de la clase:

setTitle():

Fija el título de la ventana de reproducción:

```
//metodo que fija el titulo
public void setTitle(String address, int port)
{
        senderAddress= address;
        senderPort= port + "";
        String title = senderAddress + ":" + senderPort;
        super.setTitle(title);
}
```

initialize():

Inicializa la ventana de reproducción:

```
public void initialize() { add(new PlayerPanel(player)); }
```

close():

Cierra la ventana de reproducción:

```
public void close()
{
        player.close();
        setVisible(false);
        dispose();
}
```

addNotify():

Añade un notificador:

```
public void addNotify()
{
        super.addNotify();
        pack();
}
```

Subclase “PlayerPanel”:

Esta subclase de la clase “*Recepcion*” se utiliza para mantener un panel de reproducción.

Variables miembro:

```
private Component vc, cc;
```

Constructor de la clase:

Inicializa las variables miembro de la clase.

```
public PlayerPanel(Player p)
{
	setLayout(new BorderLayout());
	if ((vc = p.getVisualComponent()) != null)
		add("Center", vc);
	if ((cc = p.getControlPanelComponent()) != null)
		add("South", cc);
}
```

Métodos de la clase:

getPreferredSize():

Devuelve el tamaño del panel:

```
public Dimension getPreferredSize()
{
	int w = 0, h = 0;
	if (vc != null)
	{
		Dimension size = vc.getPreferredSize();
		w = size.width;
		h = size.height;
	}
	if (cc != null)
	{
		Dimension size = cc.getPreferredSize();
		if (w == 0)
		w = size.width;
		h += size.height;
	}
	if (w < 160)
		w = 160;
	return new Dimension(w, h);
}
```

Clase “Config”:

Esta clase se utiliza para cargar y guardar la configuración del programa principal y no tener que estar introduciéndola de nuevo cada vez que se inicie.

Constantes de la clase:

private static final String ARCHIVO = "conf.dat";

- Nombre del archivo de configuración.

Variables miembro:

private String pathPrefix;

- Ruta al archivo de configuración.

private Target destino;

- Objeto que guarda los datos de la comunicación.

Constructor de la clase:

Obtiene la ruta al directorio home del usuario y llama al método que lee los datos del archivo.

```
public Config()
{
	//obtenemos el directorio home del usuario
	pathPrefix = System.getProperty("user.home") + File.separator;
	System.out.println(Misc.ind() + "Directorio de usuario: "' + pathPrefix + '"");
	//leemos la configuración de disco
	read();
}
```

Métodos de la clase:

Read():

Este método lee los valores de configuración de un archivo y los almacena en la variable miembro “*destino*”:

```
public void read()
{
	//añadimos el nombre del archivo a la ruta
	String path = pathPrefix + ARCHIVO;
	System.out.println(Misc.ind() + "Usando fichero de configuracion: "' + path + '"");
	try
	{
		//abrimos el archivo para lectura
		FileInputStream fin = new FileInputStream(path);
		//creamos un buffer de entrada
		BufferedInputStream bin = new BufferedInputStream(fin);
		//creamos una secuencia de datos a partir del buffer
		DataInputStream din = new DataInputStream(bin);
		//leemos de disco los datos y los almacenamos
		destino = new Target(readString(din), readString(din), readString(din),
readString(din), readString(din), readString(din));
		//cerramos el archivo
		fin.close();
	}
	//capturamos la excepcion
	catch (FileNotFoundException e)
	{
		//creamos unos datos por defecto
		System.out.println(Misc.ind() + "¡No se encuentra el archivo 'conf.dat'!");
```

```
                destino = new Target();
        }
        //capturamos la excepcion
        catch(IOException e)
        {
                // creamos unos datos por defecto
                System.out.println(Misc.ind() + "¡No se encuentra el archivo 'conf.dat'!");
                destino = new Target();
        }
        System.out.println(Misc.ind() + "Dirección IP local: " + destino.ipLocal());
        System.out.println(Misc.ind() + "Puerto de audio local: " + destino.puertoAudioLocal());
        System.out.println(Misc.ind() + "Puerto de video local: " + destino.puertoVideoLocal());
        System.out.println(Misc.ind() + "Direccion IP remota: " + destino.ipRemota());
        System.out.println(Misc.ind() + "Puerto de audio remoto: " +
destino.puertoAudioRemoto());
        System.out.println(Misc.ind() + "Puerto de video remoto: " +
destino.puertoVideoRemoto());
}
```

Write():

Este método guarda los valores de configuración en un archivo:

```
public void write()
{
        //añadimos el nombre del archivo a la ruta
        String path = pathPrefix + ARCHIVO;
        System.out.println(Misc.ind() + "Usando fichero de configuracion: '" + path + "'");
        try
        {       //abrimos el fichero para escritura
                FileOutputStream fout = new FileOutputStream(path);
                //creamos un buffer de escritura
                BufferedOutputStream bout = new BufferedOutputStream(fout);
                //creamos una secuencia de escritura
                DataOutputStream dout = new DataOutputStream(bout);
                //escribimos los valores a disco
                writeString(dout, destino.ipLocal());
                System.out.println(Misc.ind() + "Direccion IP local: " + destino.ipLocal());
                writeString(dout, destino.puertoAudioLocal());
                System.out.println(Misc.ind() + "Puerto de audio local: " +
destino.puertoAudioLocal());
                writeString(dout, destino.puertoVideoLocal());
                System.out.println(Misc.ind() + "Puerto de video local: " +
destino.puertoVideoLocal());
                writeString(dout, destino.ipRemota());
                System.out.println(Misc.ind() + "Direccion IP remota: " + destino.ipRemota());
                writeString(dout, destino.puertoAudioRemoto());
                System.out.println(Misc.ind() + "Puerto de audio remoto: " +
destino.puertoAudioRemoto());
                writeString(dout, destino.puertoVideoRemoto());
                System.out.println(Misc.ind() + "Puerto de video remoto: " +
destino.puertoVideoRemoto());
                //limpiamos la secuencia de salida
                dout.flush();
                //limpiamos el buffer
                dout.close();
                //cerramos el archivo
                fout.close();
        }//capturamos la excepcion
        catch( IOException e)
        {
```

```
                System.out.println("¡Error escribiendo el archivo 'conf.dat'!");
        }
}
```

readString():

Este método lee una cadena de un fichero y devuelve el resultado:

```
        public String readString(DataInputStream din)
        {
                String s = null;

                try
                {
                        //leemos la longitud de la cadena
                        short length = din.readShort();
                        //si la longitud > 0, leemos la cadena
                        if(length > 0)
                        {
                                byte buf[] = new byte[length];
                                din.read(buf, 0, length);
                                s = new String(buf);
                        }
                }
                //capturamos la excepcion
                catch(IOException e)
                {
                        System.err.println( e);
                }

                return s;
        }
```

writeString():

Este método escribe una cadena en un fichero:

```
public void writeString(DataOutputStream dout, String str)
{
        try
        {
                //si la cadena existe
                if(str != null)
                {
                        //escribimos la longitud de la cadena
                        dout.writeShort(str.length());
                        //escribimos la cadena
                        dout.writeBytes(str);
                }
                //en caso contrario, escribimos 0
                else
                        dout.writeShort(0);
        }
        //capturamos la excepcion
        catch( Exception e)
        {
                e.printStackTrace();
        }
}
```

destino():

Este método devuelve el objeto que contiene la configuración:

```
public Target destino() { return destino; }
```

Clase “Target”:

Esta clase se utiliza para guardar la configuración de puertos y direcciones IP del programa principal y utilizarlos en el resto de clases.

Variables miembro:

private String ipLocal;
- Dirección IP local.

private String puertoAudioLocal;
- Puerto para la transmisión de audio.

private String puertoVideoLocal;
- Puerto para la transmisión de video.

private String ipRemota;
- Dirección IP remota.

private String puertoAudioRemoto;
- Puerto para la recepción de audio.

private String puertoVideoRemoto;
- Puerto para la recepción de video.

Constructor de la clase:

Hay 2 constructores, al primero se le pasa como parámetro los datos de la configuración a almacenar, al segundo no se le pasan parámetros e inicializa las variables miembro con la cadena vacía “”.

```
public Target(String ipLocal, String localPort, String localVideoPort, String ip, String audioPort,
String videoPort)
{	this.ipLocal = ipLocal;
	puertoAudioLocal = localPort;
	puertoVideoLocal = localVideoPort;
	ipRemota = ip;
	puertoAudioRemoto = audioPort;
	puertoVideoRemoto = videoPort;
}

public Target()
{	ipLocal = "";
	puertoAudioLocal = "";
	puertoVideoLocal = "";
	ipRemota = "";
	puertoAudioRemoto = "";
	puertoVideoRemoto = "";
}
```

Métodos de la clase:

Estos métodos devuelven el contenido de las variables miembro:

```
public String ipLocal() { return ipLocal; }
public String puertoAudioLocal() { return puertoAudioLocal; }
public String puertoVideoLocal() { return puertoVideoLocal; }
public String ipRemota() { return ipRemota; }
public String puertoAudioRemoto() { return puertoAudioRemoto; }
public String puertoVideoRemoto() { return puertoVideoRemoto; }
```

Estos métodos cambian el contenido de las variables miembro:

```
public void ipLocal(String a) { ipLocal = a; }
public void puertoAudioLocal(String a) { puertoAudioLocal = a; }
```

```
public void puertoVideoLocal(String a) { puertoVideoLocal = a; }
public void ipRemota(String a) { ipRemota = a; }
public void puertoAudioRemoto(String a) { puertoAudioRemoto = a; }
public void puertoVideoRemoto(String a) { puertoVideoRemoto = a; }
```

Clase “Misc”:

Esta clase se utiliza para albergar los métodos que dan formato al log de la consola del programa.

Variables miembro:

private static String ind = "";

- Guarda parte del encabezado de las líneas del log del programa.

Métodos de la clase:

Ind():

Este método devuelve el encabezado de la línea del log del programa.

```
public static String ind()
{
	return "[" + getTimestamp() + "]-> " + ind;
}
```

aumentarInd():

Este método aumenta la indentación desde la que comenzará el texto del log que aparece después del encabezado:

```
public static void aumentarInd()
{
	ind = ind + "   ";
}
```

reducirInd():

Este método disminuye la indentación desde la que comenzará el texto del log que aparece después del encabezado:

```
public static void reducirInd()
{
	//comprobamos que la cadena no tenta el tamaño mínimo
	if (ind.length() > 0)
		ind = ind.substring(3);
}
```

formatTime():

Este método da formato de 2 dígitos a números de 1 dígito:

```
private static String formatTime(int time)
{
	String timeStr;
	//si el valor es menor que 10 añadimos un 0 a la izquierda
	if(time < 10)
	{
		timeStr = "0" + time;
	}
	//sino no añadimos nada
	else
	{
		timeStr = "" + time;
	}
		//devolvemos la cadena con formato
	return timeStr;
}
```

getTimestamp():

Este método devuelve la hora del sistema en una cadena:

```
public static String getTimestamp()
{
	String timestamp;
	Calendar calendar = Calendar.getInstance();
	//generamos la hora
	int hour = calendar.get(Calendar.HOUR_OF_DAY);
	//damos formato a la hora
	String hourStr = formatTime(hour);
	//generamos los minutos
	int minute = calendar.get(Calendar.MINUTE);
	//damos formato a los minutos
	String minuteStr = formatTime(minute);
	//generamos los segundos
	int second = calendar.get(Calendar.SECOND);
	//damos formato a los segundos
	String secondStr = formatTime(second);
	//damos formato a la cadena final
	timestamp = hourStr + ":" + minuteStr + ":" + secondStr;
	//devolvemos la cadena
	return timestamp;
}
```