Manual de sistema

me.Chat

Integrantes:
Jaider Olivella Morales
Luis Daniel Fuentes

Universidad del Norte
Barranquilla, Colombia
2021-II

Apartados

1. Servidor

1.1. Server

1.2. Hilos

A. Chat
B. Imágenes
C. Cámara
D. BroadcastThread
E. ConexionAudio

1.4. Utils

A. SoundPacket
B. Utils
C. Message

2. Cliente

2.1. Cliente

2.2. Hilos

A. Chat
B. Imágenes
C. Cámara
D. EnviarImagenCamara
E. AudioChannel
F. Voz
G. Micrófono

1. Servidor

En este apartado se van a explicar las clases que conforman a la parte del servidor, el funcionamiento de cada una de ellas y las formas en las cuales estas fueron implementadas.

1.1 Server

Esta clase es la encargada de procesar las peticiones de los clientes o usuarios.

```
public Server(int port) throws IOException {
    this.port = port;
    ServerSocket s = new ServerSocket(this.port);
    ServerSocket s2 = new ServerSocket(this.port + 1);
    ServerSocket s3 = new ServerSocket(this.port + 2);
    ServerSocket s4 = new ServerSocket(this.port + 3);

    BroadcastThread bt = new BroadcastThread(this);
    bt.start();

    while (true) {
        Socket socket = s.accept();
        this.socket = socket;
        ChatThreadS hiloChat = new ChatThreadS(socket, this);
        hiloChat.start();
        hilosChat.add(hiloChat);

        Socket socket2 = s2.accept();
        ImagenesThreadS hi = new ImagenesThreadS(socket2);
        hi.start();
        hiloImagen.add(hi);

        Socket socket3 = s3.accept();
        ConexionAudioThreadS cc = new ConexionAudioThreadS(this, socket3);
        cc.start();
        this.addToClients(cc);

        Socket socket4 = s4.accept();
        CamaraThreadS hcs = new CamaraThreadS(socket4, this);
        hcs.start();
        hiloCamaras.add(hcs);

    }
}
```

En el constructor de la clase se inicializan los server sockets que va a usar el programa, cada uno para un hilo diferente para así poder

enviar y recibir información de manera más eficiente, ya que los hilos envían y reciben diferentes tipos de datos. Después en un bucle infinito, se espera hasta recibir la conexión de un cliente. Cada hilo se guarda en un ArrayList para así facilitar la transmisión de datos a todos los clientes.

```
public void transmision(String mensaje) throws IOException {
    for (ChatThreadS hc : hilosChat) {
        hc.enviarMensaje(mensaje);
    }
}

public void transmision(ChatThreadS hc, String mensaje) throws IOException {
    hc.enviarMensaje(mensaje);
}

public void transmision() throws IOException {
    for (ImagenesThreadS hiImagen : hiloImagen) {
        for (BufferedImage bi : images) {
            hiImagen.enviarMensaje(bi);
        }
    }
}

public void transmisionCamera(BufferedImage image) throws IOException {
    for (CamaraThreadS hcs : hiloCamaras) {
        hcs.enviarMensaje(image);
    }
}

public void flush() throws IOException {
    for (ChatThreadS hc : hilosChat) {
        hc.flush();
    }
}
```

También esta clase cuenta con los métodos para la transmisión de datos a todos los clientes, así como su método para eliminar la conexión de un cliente.

```
public void deleteUser(ChatThreadS hc, String name) {
    int index = hilosChat.indexOf(hc);
    hilosChat.remove(index);
    hiloImagen.remove(index);
    hiloCamaras.remove(index);
    index = names.indexOf(name);
    names.remove(index);
    images.remove(index);
}
```

1.1 Hilos

El servidor cuenta con 5 hilos encargados de recibir y enviar información a los clientes, el principal sería el chat, debido a que este se podría decir que indica el comportamiento de los otros.

A. Chat

Este hilo es el encargado de recibir y enviar la información proveniente de los mensajes de texto de los clientes, para esto se usa las clases de BufferedWriter y BufferedReader, que escribe y lee texto en un flujo de salida de caracteres, almacenando los caracteres en el búfer para permitir la escritura eficiente de caracteres individuales, matrices y cadenas.

```
@Override
public void run() {
    try {
        dataInputStream = new DataInputStream(socket.getInputStream());
        dataOutputStream = new DataOutputStream(socket.getOutputStream());
        reader = new BufferedReader(new InputStreamReader(dataInputStream));
        writer = new BufferedWriter(new OutputStreamWriter(dataOutputStream));

        while (true) {
            String data = reader.readLine().trim();

            if (data.equals("/addUser")) {
                data = reader.readLine().trim();
                String name = data;
                data = reader.readLine().trim();
                String path = data;

                boolean nameRepeted = false;

                for (String nombre : server.getNames()) {
                    if (name.equals(nombre)) {
                        nameRepeted = true;
                    }
                }

                if (!nameRepeted) {
                    server.addName(name);
                    server.transmision("/clear");
                    server.addUsuarios(name + "," + path);
                    server.transmision("/nuevoUsuario");

                    for (String usuario : server.getNames()) {
                        server.transmision(usuario);
                    }

                    server.transmision("/finUsuario");
                    data = reader.readLine().trim();
```

```
        } else {
            server.transmision(this, "/nameRepeted");
        }

    } else if (data.equals("/deleteUser")) {
        server.transmision("/deleteUser");
        data = reader.readLine().trim();
        server.transmision(data);
        server.deleteUser(this, data);
    } else if (data.equals("/cameraOn")) {
        server.transmision("/cameraOn");
        data = reader.readLine().trim();
        server.transmision(data);
    } else if (data.equals("/cameraOff")) {
        server.transmision("/cameraOff");
        data = reader.readLine().trim();
        server.transmision(data);
    } else {
        server.transmision(data);
    }
```

En el método run lo que se hace es recibir los mensajes de texto, en la parte de cliente, antes de los mensajes, ingresos o salida de los usuario, se envían estos comandos para el que el programa sepa cuando hay que añadir un usuario o cuando se va a enviar mensaje, cuando se añade un usuario por ejemplo, el programa antes del nombre de usuario, recibe "/addUser", lo que permite guardar el nombre y imagen de perfil en respectivos ArrayList, así ocurre parecido con las diferentes acciones que se pueden hacer, como activar la camara, eliminar un usuario entre otras, por esto, este hilo es el principal, ya que recibo qué proceso debe realizar, y los métodos de transmisión envían el mensaje a cada hilo de chat, como se vio en la clase de server.

B. Imágenes

Este hilo solamente recibe la imagen de perfil de los usuarios cuando ingresan al programa, y luego tiene un método para envíar la foto de perfil de los usuarios como un arreglo de bytes o los diferentes clientes.

```java
public void enviarMensaje(BufferedImage image) throws IOException {
    ByteArrayOutputStream ous = new ByteArrayOutputStream();
    ImageIO.write(image, "png", ous);
    socket.getOutputStream().write(ous.toByteArray());
}

@Override
public void run() {
    try {
        while (true) {
            BufferedImage img = ImageIO.read(socket.getInputStream());
            if (img != null) {
                server.addImages(img);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, e);
    }
}
```

C. Cámara

Este hilo recibe y envía la imagen de la cámara de los usuarios que la tengan activada, igual que el hilo de imágenes, envía la imagen como un arreglo de bytes, y cuando la recibe la guarda como BufferedImage.

```java
public void enviarMensaje(BufferedImage image) throws IOException {
    ByteArrayOutputStream ous = new ByteArrayOutputStream();
    ImageIO.write(image, "png", ous);
    socket.getOutputStream().write(ous.toByteArray());
}

@Override
public void run() {

    try {
        while (true) {
            try {
                BufferedImage img = ImageIO.read(socket.getInputStream());
                if (img != null) {
                    server.transmisionCamera(img);
                    sleep(50);
                }
            } catch (IIOException e) {
                System.out.println("error " + e);
                server.transmision("/camaraBug");
            } catch (InterruptedException ex) {
                Logger.getLogger(CamaraThreadS.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
    } catch (IOException ex) {
        Logger.getLogger(CamaraThreadS.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

D. BroadcastThread

Este hilo recibe las conexiones del cliente con el hilo de voz, elimina conexiones muertas y añade a la cola de reproducción las entradas nuevas.

```java
@Override
public void run() {
    while (true) {
        try {
            ArrayList<ClientConnection> toRemove = new ArrayList<ClientConnection>();
            for (ClientConnection cc : s.getClients()) {
                if (!cc.isAlive()) {
                    toRemove.add(cc);
                }
            }
            s.getClients().removeAll(toRemove);
            if (s.getBroadCastQueue().isEmpty()) {
                Utils.sleep(10);
                continue;
            } else {
                Message m = s.getBroadCastQueue().get(0);
                for (ClientConnection cc : s.getClients()) {
                    if (cc.getChId() != m.getChId()) {
                        cc.addToQueue(m);
                    }
                }
                s.getBroadCastQueue().remove(m);
            }
        } catch (Throwable t) {
        }
    }
}
```

D. ConexiónAudio

Este hilo maneja las conexiones de audio con los clientes, añade dicho datos para emitirlos al servidor y los elimina su los datos de audio son muy viejos.

```java
@Override
public void run() {
    try {
        out = new ObjectOutputStream(s.getOutputStream());
        in = new ObjectInputStream(s.getInputStream());
    } catch (IOException ex) {
        try {
            s.close();
        } catch (IOException ex1) {
        }
        stop();
    }
    while (true) {
        try {
            if (s.getInputStream().available() > 0) {
                Message toBroadcast = (Message) in.readObject();
                if (toBroadcast.getChId() == -1) {
                    toBroadcast.setChId(chId);
                    toBroadcast.setTimestamp(System.nanoTime() / 1000000L);
                    serv.addToBroadcastQueue(toBroadcast);
                } else {
                    continue;
                }
            }
            try {
                if (!toSend.isEmpty()) {
                    Message toClient = toSend.get(0);
                    if (!(toClient.getData() instanceof SoundPacket) || toClient.getTimestamp() + toClient.getTtl() < System.nanoTime() / 1000000L) {
                        continue;
                    }
                    out.writeObject(toClient);
                    toSend.remove(toClient);
                } else {
                    Utils.sleep(10);
                }
```

1.3 Utils

Las siguientes clases son clases de utilidad que tienen tanto el servidor como los clientes, para el correcto funcionamiento del canal de audio.

A. SoundPacket

Esta clase es la que tiene el formato de audio que se recibe, en este caso es de 11.025khz, 8 bit en mono stereo

```java
public static AudioFormat defaultFormat = new AudioFormat(11025f, 8, 1, true, true);
public static int defaultDataLenght = 1200;
private byte[] data;

public SoundPacket(byte[] data) {
    this.data = data;
}

public byte[] getData() {
    return data;
}
```

B. Message

Esta clase contiene los objetos que se van a enviar.

```
private long chId;
private long timestamp,
        ttl = 2000;
private Object data;

public Message(long chId, long timestamp, Object data) {
    this.chId = chId;
    this.timestamp = timestamp;
    this.data = data;
}
```

2. Cliente

En este apartado se van a explicar las clases que conforman a la parte del cliente y el funcionamiento de cada una de ellas.

2.1 Cliente

Esta es la clase principal, aquí primero que todo se muestra la interfaz de usuario y se configuran ciertas propiedades que tendrán los objetos en la interfaz.

```
public static void main(String[] args) throws IOException {
    inicio = new inicio();
    inicio.setVisible(true);
    inicio.setLocationRelativeTo(null);
    inicio.setSize(1200, 700);
    chatTable = inicio.getTable1();
    usuariosConectados = inicio.getTable12();
    propiedadesTabla2();

    cliente = new cliente();
    inicio.setCliente(cliente);
}
```

También tiene los métodos que permiten mediante el ip y el puerto, establecer una conexión con el servidor.

```
public void connect(String ip, int port) {
    try {
        socketClient = new Socket(ip, port);
        hc = new ChatThreadC(socketClient, cliente, ip, port);
        hc.start();

        s = new Socket(ip, port + 1);
        ImagenesThreadC hi = new ImagenesThreadC(s, cliente);
        hi.start();

        cv = new VozThread(ip, port + 2);
        cv.start();
        MicrofonoThread.amplification = 0;

    } catch (Exception e) {
        System.out.println("error " + e);
    }
}
```

También tiene el método para iniciar una reunión, llamando al servidor en un hilo para que no detenga la ejecución del programa, por lo que el cliente o usuario que inicia una reunión, contiene el servidor, por lo que si este usuario cierra el programa el servidor también se cerrará.

```
public void createMeeting(int port) throws IOException {
    new Thread() {
        @Override
        public void run() {
            try {
                Server servidor = new Server(port);
            } catch (Exception ex) {
                System.exit(0);
            }
        }
    }.start();
}
```

Los siguientes métodos son para mostrar los mensajes de texto, también el programa cuenta que cuando un usuario pone el nombre de otro usuario con @, por ejemplo “@carlos”, a este usuario se reproducirá un pequeño sonido, que le indica que los están llamando.

```java
public void showMessage(ImageIcon foto2, String sender, String msg, String nombre, boolean aux) {
    if (aux) {
        if (sender.equals(nombre)) {
            getChatModel().addRow(new Object[]{new JLabel(foto2), "Te has unido a la reunion "});
        } else {
            getChatModel().addRow(new Object[]{new JLabel(foto2), sender + msg});
        }
    } else {
        if (sender.equals(nombre)) {
            getChatModel().addRow(new Object[]{new JLabel(foto2), "Has avandonado la reunion"});
        } else {
            getChatModel().addRow(new Object[]{new JLabel(foto2), sender + msg});
        }
    }

}
```

```java
public void showMessage(ImageIcon foto2, String msg, String nombre) {
    String data[] = msg.split(":");
    String[] splited = data[1].split("\\s+");
    if (data[0].equals(nombre)) {
        getChatModel().addRow(new Object[]{new JLabel(foto2), "tu: " + data[1]});
    } else {
        if (splited[1].equals("@" + nombre)) {
            try {
                getChatModel().addRow(new Object[]{new JLabel(foto2), msg});
                String soundName = "src/audio/audio.wav";
                AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(new File(soundName).getAbsoluteFile());
                Clip clip = AudioSystem.getClip();
                clip.open(audioInputStream);
                clip.start();
            } catch (LineUnavailableException ex) {
                Logger.getLogger(cliente.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {
                Logger.getLogger(cliente.class.getName()).log(Level.SEVERE, null, ex);
            } catch (UnsupportedAudioFileException ex) {
                Logger.getLogger(cliente.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else {
            getChatModel().addRow(new Object[]{new JLabel(foto2), msg});
        }
    }
    chatTable.scrollRectToVisible(chatTable.getCellRect(chatTable.getRowCount() - 1, chatTable.getColumnCount(), true));
}
```

Los siguientes métodos son para actualizar los paneles cuando el usuario activa o apagar su cámara.

```java
public void cameraOn(BufferedImage img, int index) {
    ImageIcon icon = new ImageIcon(img);
    Image imageChange = icon.getImage();
    Image newimg = imageChange.getScaledInstance(202, 108, java.awt.Image.SCALE_SMOOTH);
    icon = new ImageIcon(newimg);
    userPanel u = new userPanel(icon);
    userPanels.set(index, u);
    inicio.getCallPanel().removeAll();
    updatePanels();
}

public void stopCamera(int indexCamera) {
    ImageIcon icon = images.get(indexCamera);
    Image imageChange = icon.getImage();
    Image newimg = imageChange.getScaledInstance(202, 108, java.awt.Image.SCALE_SMOOTH);
    icon = new ImageIcon(newimg);
    userPanel u = new userPanel(icon);
    userPanels.set(indexCamera, u);
    inicio.getCallPanel().removeAll();
    inicio.getCallPanel().revalidate();
    inicio.getCallPanel().repaint();
    updatePanels();
}
```

Los siguientes métodos añaden las imágenes a los paneles de cada usuario y elimina un usuario y sus datos cuando este abandona la reunión.

```java
public void addClientImage(BufferedImage img) {
    ImageIcon icon = new ImageIcon(img);
    images.add(icon);
    Image imageChange = icon.getImage();
    Image newimg = imageChange.getScaledInstance(202, 108, java.awt.Image.SCALE_SMOOTH);
    icon = new ImageIcon(newimg);
    userPanel u = new userPanel(icon);
    userPanels.add(u);
}

public void deleteUser(String name) {
    int i = names.indexOf(name);
    names.remove(i);
    images.remove(i);
    userPanels.remove(i);
    getUsuariosConectadosModel().removeRow(i);
    inicio.getCallPanel().removeAll();
    inicio.getCallPanel().revalidate();
    inicio.getCallPanel().repaint();
    updatePanels();
}
```

Y este método actualiza los paneles de los usuarios.

```
public void updateUsers() {
    for (userPanel u : userPanels) {
        inicio.getCallPanel().add(u);
    }
    inicio.getCallPanel().revalidate();
    inicio.getCallPanel().repaint();
}
```

2.2 Hilos

En este apartado se mostrarán los hilos que se usaron para el intercambio de datos con el servidor, comenzando con hilo de Chat que al igual que en el servidor actúa como principal, que indica los procesos a realizar por los diferentes hilos.

A. Chat

Este hilo recibe y envía los datos de los mensajes de texto de los usuarios, al igual que la clase de chat de servidor, utiliza la clase BufferedWriter y BufferedReader.

```java
public void deleteUser(String name) {
    try {
        enviarComands(name, false);
        sendMessage("/deleteUser");
        sendMessage(name);
    } catch (Exception e) {
        System.out.println("error " + e);
    }
}

public void cameraOn(String name) {
    sendMessage("/cameraOn");
    sendMessage(name);
}

public void cameraOff(String name) {
    sendMessage("/cameraOff");
    sendMessage(name);
}

public void insertarCliente(String nombre, String foto) {
    try {
        this.nombre = nombre;
        this.foto = foto;
        c.setNombre(this.nombre);
        hcc.setNameOwner(this.nombre);
        c.getUsuariosConectadosModel().setRowCount(0);
        sendMessage("/addUser");
        sendMessage(this.nombre);
        sendMessage(foto);
        sendMessage("/closeUser");
        enviarComands(nombre, true);
    } catch (Exception e) {
        System.out.println("error " + e);
    }
}
```

```java
public void sendMessage(String msg) {
    try {
        writer.write(msg);
        writer.write("\r\n");
        writer.flush();
    } catch (IOException ex) {
        Logger.getLogger(ChatThreadC.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void enviarMensaje(String nombre, String mensaje) {
    try {
        sendMessage("/starMessage");
        sendMessage(nombre + ": " + mensaje + "\n");
        sendMessage("/stopMessage");
    } catch (Exception e) {
        System.out.println("error " + e);
    }
}

public void enviarComands(String nombre, boolean auxiliar) {
    try {
        if (auxiliar) {
            sendMessage("/joinChat");
            sendMessage(nombre);
        } else {
            sendMessage("/leftChat");
            sendMessage(nombre);
        }
    } catch (Exception e) {
        System.out.println("error " + e);
    }

}
```

Los anteriores métodos simplemente envían mensajes al servidor con los comandos específicos dependiendo el método y el proceso que quiere hacer, el metodo enviarMensaje, antes de enviar el mensaje de los usuarios, envía el comando “/startMessage” y después del mensaje envía “/stopMessage” y como se vio en la parte del hilo de chat del servidor, al recibir estos comandos se hacía la transmisión a todos los hilos de chat del server. Así ocurre con cada método y el comando cambia así que como en alguno se guarda los nombre de usuario.

```java
public void run() {
    try {
        String msg = "", texto = "";
        int i = 0;

        while (msg != null) {
            sleep(100);
            msg = reader.readLine();

            if (msg.equals("")) {
                msg = reader.readLine();
            }

            if (msg.equals("/nuevoUsuario")) {
                msg = reader.readLine();
                c.getUsuariosConectadosModel().setRowCount(0);
                sleep(5000);
                while (!msg.equals("/finUsuario")) {
                    c.updateImages(msg, i);
                    msg = reader.readLine();
                    c.updateConnectedUsers();
                    i++;
                }
                c.updatePanels();
                c.makeAdmin();

            } else if (msg.equals("/starMessage")) {
                msg = reader.readLine();
                foto2 = c.getImages().get(c.getIndex(msg));
                Image imgage = foto2.getImage();
                Image newimg = imgage.getScaledInstance(50, 50, java.awt.Image.SCALE_SMOOTH);
                foto2 = new ImageIcon(newimg);
                c.showMessage(foto2, msg, this.nombre);
                msg = reader.readLine();
            } else if (msg.equals("/clear")) {
                c.clearPanel();
                i = 0;
```

```
} else if (msg.equals("/cameraOn")) {
    msg = reader.readLine();
    hcc.setNameCameraStart(msg);
    hcc.setCameraOn(true);
    synchronized (hcc) {
        hcc.notify();
    }
} else if (msg.equals("/cameraOff")) {
    msg = reader.readLine();
    hcc.setNameCameraStart(msg);
    hcc.setCameraOn(false);
    sleep(10);
    c.stopCamera(c.getIndexCamera(msg));
} else if (msg.equals("/nameRepeted")) {
    c.getInicio().getChat().setVisible(false);
    c.getInicio().setVisible(true);
    socketClient.close();
    JOptionPane.showMessageDialog(null, "Nombre de usuario ya esta en uso");
    System.exit(0);
    break;
} else if (msg.equals("/deleteUser")) {
    msg = reader.readLine();
    c.deleteUser(msg);
    c.updateConnectedUsers();
} else if (msg.equals("/joinChat")) {
    msg = reader.readLine();
    foto2 = c.getImages().get(c.getIndex2(msg));
    Image imgage = foto2.getImage();
    Image newimg = imgage.getScaledInstance(50, 50, java.awt.Image.SCALE_SMOOTH);
    foto2 = new ImageIcon(newimg);
    c.showMessage(foto2, msg, " se ha unido a la reunion", this.nombre, true);
} else if (msg.equals("/leftChat")) {
    msg = reader.readLine();
    foto2 = c.getImages().get(c.getIndex2(msg));
    Image imgage = foto2.getImage();
    Image newimg = imgage.getScaledInstance(50, 50, java.awt.Image.SCALE_SMOOTH);
    foto2 = new ImageIcon(newimg);
    c.showMessage(foto2, msg, " abandono la reunion", this.nombre, false);
} else if (msg.equals("/closeMeeting")) {
    System.exit(0);
```

El método run de este hilo recibe los mensajes del servidor, cuando es un mensaje lo muestra en el chat, cuando se añade un usuario guarda en un ArrayList en la clase cliente, el nombre y en otro ArrayList la imagen de perfil, cuando se activa la cámara, mediante el método notify comienza a ejecutarse el hilo de cámara, ya que este hilo está esperando con el método wait, hasta que el usuario encienda la cámara.

B. Imágenes

Este hilo envía la foto de perfil que el usuario seleccionó igualmente como todos los hilos de imagen como un arreglo de bytes, también recibe la foto de perfil de los usuarios como arreglo de bytes, y lo pasa a BufferedImage y actualiza los paneles de usuario.

```
public void enviarImagen(BufferedImage image) throws IOException {
    ByteArrayOutputStream ous = new ByteArrayOutputStream();
    ImageIO.write(image, "png", ous);
    socket.getOutputStream().write(ous.toByteArray());
}

@Override
public void run() {
    try {
        while (true) {
            BufferedImage img = ImageIO.read(socket.getInputStream());
            if (img != null) {
                cliente.addClientImage(img);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, e);
    }
}
```

C. Cámara

Este hilo como se dijo en el hilo de chat, está esperando que el usuario encienda la cámara.

```java
@Override
public void run() {
    int i = 0;
    boolean aux = true;
    synchronized (this) {
        try {
            this.wait();
            while (true) {
                try {
                    if (cameraOn && nameAux.equals(nameOwner) && aux) {
                        sendCamera.setCameraOn(true);
                        aux = false;
                        synchronized (sendCamera) {
                            sendCamera.notify();
                        }
                    }

                    if (cameraOn) {
                        BufferedImage img = ImageIO.read(socket.getInputStream());
                        if (img != null) {
                            cliente.cameraOn(img, cliente.getIndexCamera(nameAux));
                        }
                    } else if (!cameraOn) {
                        sendCamera.setCameraOn(false);
                        cliente.stopCamera(cliente.getIndexCamera(nameAux));
                        aux = true;
                        this.wait();
                    }

                } catch (IOException ex) {
                    Logger.getLogger(CamaraThreadC.class.getName()).log(Level.SEVERE, null, ex);
                    sendCamera.setCameraOn(false);
                    cliente.stopCamera(cliente.getIndexCamera(nameAux));
                    aux = true;
                    this.wait();
                } catch (InterruptedException ex) {
                    Logger.getLogger(CamaraThreadC.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
```

Cuando este hilo se notifica, notifica a otro hilo que es el que va a enviar la imagen de la cámara, este hilo también recibe la imagen de otros usuarios que envía el servidor, y actualiza continuamente los paneles, con la imagen proveniente de la cámara. Cuando la cámara se apaga, el hilo vuelve a esperar.

D. EnviarImagenCamara

Este hilo envía la imagen de la cámara del usuario, este mediante el uso de una api llamada Webcam Capture, mediante esta api, obtenemos la imagen de la cámara, para así poder enviarla al servidor como arreglo de bytes, este hilo está separado del anterior ya que al activar la cámara este debe estar siempre activada, por lo que enviarla y recibirla en el mismo hilo se vuelve un poco tedioso, tambien verifica si la camara no esta siendo utilizada por otra aplicación, ya que una cámara lo puede utilizar una aplicación a la vez.

```java
@Override
public void run() {
    try {
        BufferedImage image;
        String name = null;
        int camaras = 0;
        Webcam cam = null;

        for (Webcam w : Webcam.getWebcams()) {
            if (!w.isOpen()) {
                name = w.getName();
                break;
            }
        }

        synchronized (this) {
            this.wait();
            int i = 0;
            while (true) {
                while (cameraOn) {

                    try {
                        cam = Webcam.getWebcamByName(name);
                        cam.open();
                    } catch (WebcamException e) {
                        for (Webcam w : Webcam.getWebcams()) {
                            if (!w.isOpen()) {
                                name = w.getName();
                                break;
                            }
                        }
                        cam = Webcam.getWebcamByName(name);
                        cam.open();
                    }
                    image = cam.getImage();
                    byte[] bytes = WebcamUtils.getImageBytes(cam, "png");
                    socket.getOutputStream().write(bytes);
                    socket.getOutputStream().flush();
```

E. CanalAudio

Este hilo reproduce el sonido que proviene del canal de audio de los

usuarios conectados en el servidor.

```
public void run() {
    try {
        AudioFormat af = SoundPacket.defaultFormat;
        DataLine.Info info = new DataLine.Info(SourceDataLine.class, af);
        speaker = (SourceDataLine) AudioSystem.getLine(info);
        speaker.open(af);
        speaker.start();
        //si hay paquetes de sonido nuevos los reproduce
        while (true) {
            if (queue.isEmpty()) {
                sleep(10);
                continue;
            } else {
                lastPacketTime = System.nanoTime();
                Message in = queue.get(0);
                queue.remove(in);
                if (in.getData() instanceof SoundPacket) {
                    SoundPacket m = (SoundPacket) (in.getData());
                    if (m.getData() == null) {
                        byte[] noise = new byte[lastSoundPacketLen];
                        for (int i = 0; i < noise.length; i++) {
                            noise[i] = (byte) ((Math.random() * 3) - 1);
                        }
                        speaker.write(noise, 0, noise.length);
                    } else {
                        GZIPInputStream gis = new GZIPInputStream(new ByteArrayInputStream(m.getData()));
                        ByteArrayOutputStream baos = new ByteArrayOutputStream();
                        while (true) {
                            int b = gis.read();
                            if (b == -1) {
                                break;
                            } else {
                                baos.write((byte) b);
                            }
                        }
                        byte[] toPlay = baos.toByteArray();
                        speaker.write(toPlay, 0, toPlay.length);
                        lastSoundPacketLen = m.getData().length;
```

El método run mira si hay nuevos paquetes de audio para ser reproducidos, obteniendo los datos y descomprimiendolos para así poder reproducirlos.

F. Voz

Este hilo recibe los datos del servidor, crea el hilo de micrófono, y envía los datos del micrófono hacia el servidor.

```java
@Override
public void run() {
    try {
        ObjectInputStream fromServer = new ObjectInputStream(s.getInputStream());
        ObjectOutputStream toServer = new ObjectOutputStream(s.getOutputStream());
        try {
            sleep(100);
            st = new MicrofonoThread(toServer);
            st.start();
        } catch (Exception e) {
            System.out.println("mic unavailable " + e);
        }
        //envia los datos del server al canal de audio especifico
        while (true) {

            if (s.getInputStream().available() > 0) {
                Message in = (Message) (fromServer.readObject());
                CanalAudioThread sendTo = null;
                for (CanalAudioThread ch : chs) {
                    if (ch.getChId() == in.getChId()) {
                        sendTo = ch;
                    }
                }
                if (sendTo != null) {
                    sendTo.addToQueue(in);
                } else {
                    CanalAudioThread ch = new CanalAudioThread(in.getChId());
                    ch.addToQueue(in);
                    ch.start();
                    chs.add(ch);
                }
            } else {
                ArrayList<CanalAudioThread> killMe = new ArrayList<CanalAudioThread>();
                for (CanalAudioThread c : chs) {
                    if (c.canKill()) {
                        killMe.add(c);
                    }
                }
```

También crea los canales de audio para cada cliente y también los elimina si se ha perdido la conexión.

F. Micrófono

Este hilo lee los datos del micrófono del usuario y envía dichos datos al servidor.

```java
@Override
public void run() {
    while (true) {
        if (mic.available() >= SoundPacket.defaultDataLenght) {
            byte[] buff = new byte[SoundPacket.defaultDataLenght];
            while (mic.available() >= SoundPacket.defaultDataLenght) {
                mic.read(buff, 0, buff.length);
            }
            try {
                long tot = 0;
                for (int i = 0; i < buff.length; i++) {
                    buff[i] *= amplification;
                    tot += Math.abs(buff[i]);
                }
                tot *= 2.5;
                tot /= buff.length;
                //mandar los paquetes
                Message m = null;
                if (tot == 0) {
                    m = new Message(-1, -1, new SoundPacket(null));
                } else {
                    //comprimir los paquetes de sonido con GZIP
                    ByteArrayOutputStream baos = new ByteArrayOutputStream();
                    GZIPOutputStream go = new GZIPOutputStream(baos);
                    go.write(buff);
                    go.flush();
                    go.close();
                    baos.flush();
                    baos.close();
                    m = new Message(-1, -1, new SoundPacket(baos.toByteArray()));
                }
                toServer.writeObject(m);
            } catch (IOException ex) {
                stop();
            }
```

Estos datos los envía por la clase de utilidad Message comprimiendo los datos del micrófono con gzip y enviando los datos como objeto.