Conception, développement par analyse fonctionnelle en UML et test statique d'une plateforme de communication multimédia voix et vidéo sur IP pour un usage adapté au contexte des entreprises locales à Lubumbashi.

Blaise Fyama, Elie Museng, Grace Mukoma

Conception, développement par analyse fonctionnelle en UML et test statique d'une plateforme de communication multimédia voix et vidéo sur IP pour un usage adapté au contexte des entreprises locales à Lubumbashi.

Development, conception by UML functional analysis and static analysis of multimedia communication platform over IP for a usage in local companies in Lubumbashi

Blaise FYAMA

Faculté des Sciences Informatiques, Université Liberté, 2179, Avenue du 30 juin, Lubumbashi, RD Congo

bfyama@gmail.com

Elie MUSENG

Faculté des Sciences Informatiques, Université Liberté, 2179, Avenue du 30 juin, Lubumbashi, RD Congo

musengkayij@gmail.com

Grace MUKOMA

Faculté des Sciences Informatiques, Université Liberté, 2179, Avenue du 30 juin, Lubumbashi, RD Congo

Mukomalagrace80 @gmail.com

Résumé

Dans cet article nous présentons une implémentation en java de la vidéo téléphonie grâce au protocole SIP (Session Initiation Protocol). Après une analyse fonctionnelle du protocole SIP et en se basant sur des travaux des chercheurs d'une université italienne (**University of parma-Italy)** pour se doter des librairies adéquates pour un développement de notre outil de communication. Afin d'optimiser le code et d'améliorer le prototype nous nous sommes servis, dans une démarche incrémentale, des techniques de test basées sur une analyse statique reposant sur l'évaluation de la complexité du logiciel avec l'application de métriques et le nombre cyclomatique de Mccabe.

Abstract

In this paper we present an implementation in the Session Initiation Protocol (SIP). After a functional analysis of the SIP protocol and based on the work of researchers from an Italian university **(University of parma-Italy**) to acquire adequate libraries for the development of our communication tool. To optimize the code and improve the prototype with us, in an incremental approach, test techniques on a static analysis based on the evaluation of the complexity of the software with the application of metrics and the cyclomatic number of Mccabe.

Mots-clés

Sip, uml, codage, analyse statique, complexité, metriques mccabe

Keywords

sip, uml, development, static analysis, complexity, mccabe metrics

1. Introduction

Dans cet article nous présentons une implémentation en java de la vidéo téléphonie grâce au protocole SIP (Session Initiation Protocol). Cet article vise notamment, l'étude des protocoles utilisés dans la VoIP, des différentes architectures proposées, et le développement d'une application de VoIP effectuant la signalisation grâce au protocole SIP et facilitant le transport de la voix et de la vidéo. En effet, il existe des logiciels tels que Skype, et lynk de Microsoft qui font de la communication multimédia. Cependant notre objectif s'inscrit dans l'utilisation et la manipulation des standards développés par les organismes de télécommunications tels que l'ITU-T et l'IETF. Ce qui aidera à faciliter l'interopérabilité des équipements et des applications dans ce domaine.

Cette application est implémentée avec une approche orientée objet en utilisant le langage de programmation Java pour des raisons de portabilité et de réutilisation.

Il s'agit donc de développer une application basée sur le modèle client-serveur, d'analyser et traiter les messages échangés entre différentes entités. Un autre volet de notre application consiste à utiliser les protocoles spécifiques qui nous permettent d'acheminer la voix et de la vidéo sur IP.

Les entreprises, bénéficiant de notre solution, seront capables de mettre en place une plateforme de VoIP assez flexible, peu couteuse, et sécurisée. En gros notre méthodologie se présente comme suite :

1. Analyse et comparaison des standards de la communication multimédia (H.323 et SIP).
2. Étude sur les vulnérabilités de la VoIP et les bonnes pratiques de sécurisation
3. Modélisation de l'architecture de l'application
4. Conception orientée objet (représentation UML, diagramme des classes).
5. Implantation orientée objet en utilisant le langage de programmation Java.
6. Application des métriques de McCabe pour l'évaluation et l'optimisation du code source.
7. Test et résultats de l'application. Nous avons aussi fait l'évaluation de la complexité du logiciel avec l'application de métriques et le nombre cyclomatique de Mccabe.

2. Aperçu du protocole SIP

SIP est un protocole de signalisation défini par l'IETF (Internet Engineering Task Force) permettant l'établissement, la libération et la modification de sessions multimédias (RFC 3261). Il hérite de certaines fonctionnalités des protocoles HTTP (Hyper Text Transport Protocol) utilisé pour naviguer sur le WEB, et SMTP (Simple Mail Transport Protocol) utilisé pour transmettre des messages électroniques (E-mails). (The Internet Society, 2002)

SIP s'appuie sur un modèle transactionnel client/serveur comme HTTP. L'adressage utilise le concept d'URL SIP (Uniform Resource Locator) qui ressemble à une adresse E-mail. Chaque participant dans un réseau SIP est donc adressable par une URL SIP.

Par ailleurs, les requêtes SIP sont acquittées par des réponses identifiées par un code numérique. D'ailleurs, la plupart des codes de réponses SIP ont été empruntés au protocole HTTP. Par exemple, lorsque le destinataire n'est pas localisé, un code de réponse «404 Not Found » est retourné.

Une requête SIP est constituée de headers comme une commande SMTP. Enfin SIP comme SMTP est un protocole textuel.

SIP a été étendu afin de supporter de nombreux services tels que la présence, la messagerie instantanée (similaire au service SMS dans les réseaux mobiles), le transfert d'appel, la conférence, les services complémentaires de téléphonie, etc.

Le protocole SIP n'est qu'un protocole de signalisation. Une fois la session établie, les participants de la session s'échangent directement leur trafic audio/vidéo à travers le protocole RTP (Real-Time Transport Protocol).

Il s'agit d'un protocole de contrôle d'appel et non de contrôle du média. SIP n'est pas non plus un protocole de transfert de fichier tel que HTTP, utilisé afin de transporter de grands volumes de données. Il a été conçu pour transmettre des messages de signalisation courts afin d'établir, maintenir et libérer des sessions multimédia. Notons que SIP possède l'avantage de ne pas être attaché à un médium particulier et est censé être indépendant du protocole de transport. (The Internet Society, 2002).

Structure standard du protocole SIP

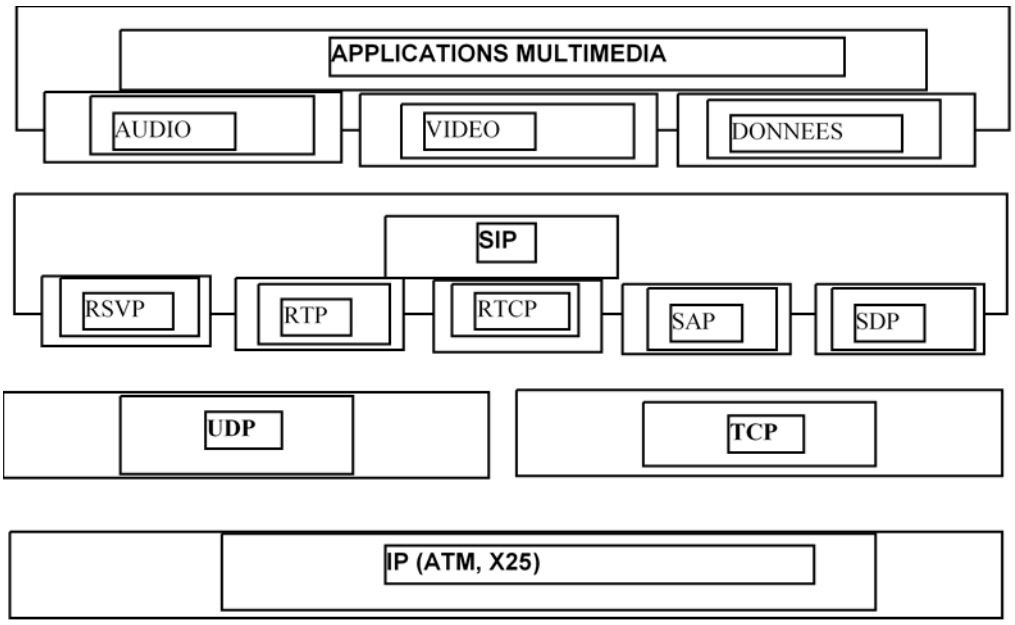


Figure 1 : Structure standard du Protocole SIP

3. Principe de fonctionnement d'un système de messagerie SIP

Le schéma ci-dessous(figure …) représente le système de dialogue du protocole SIP et l'échange des paquets RTP/RTCP de l'application entre deux agents utilisateurs client-serveur (*UAC/UAS*) à travers un serveur proxy :

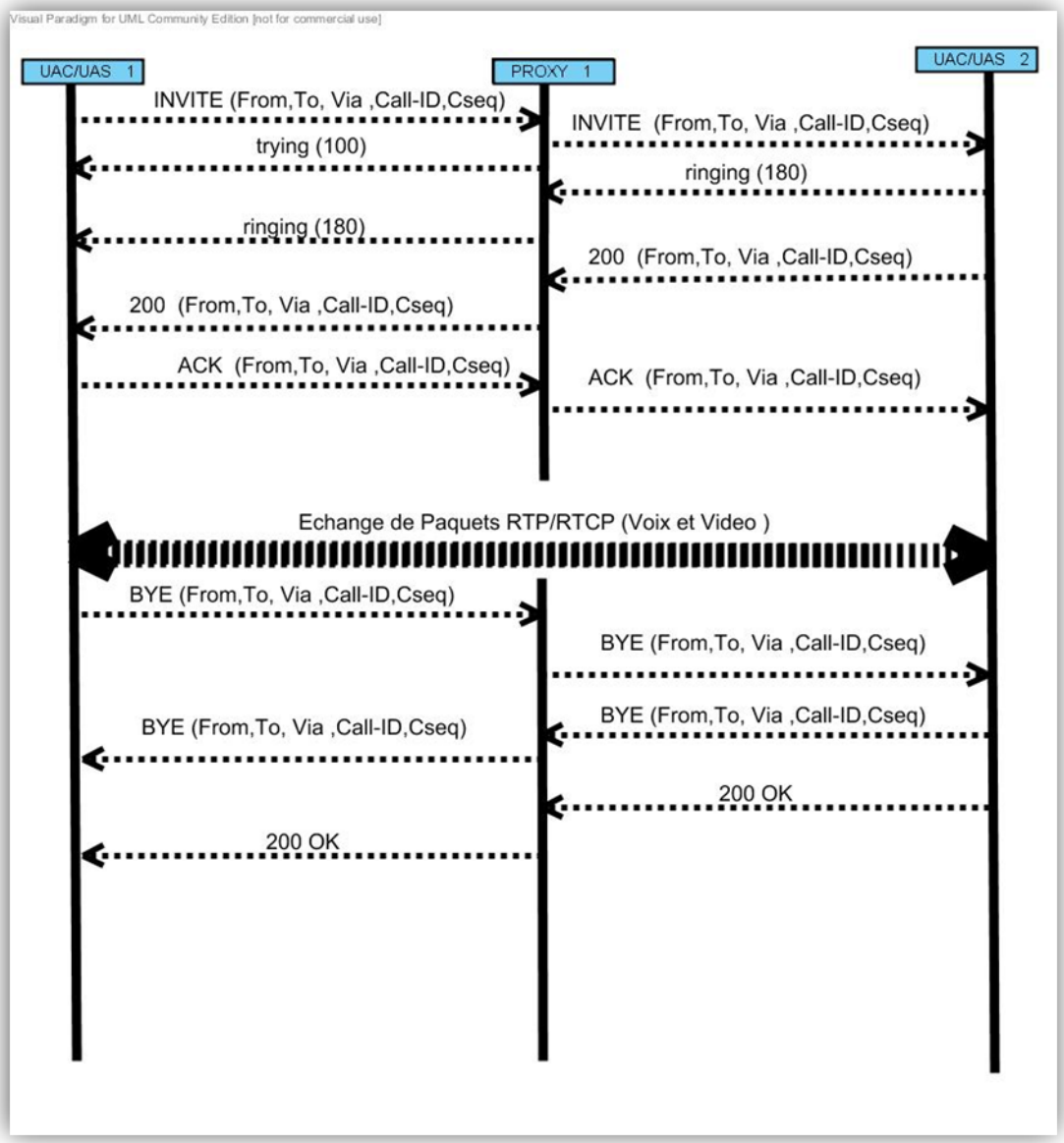


Figure 2 : Principe de fonctionnement d'un système de messagerie SIP

Succession des étapes :

1. Le client A qui désire initier une communication émet une requête INVITE contenant les URL From, To et Via. Celle-ci est envoyée au serveur proxy.

2. Le proxy reçoit la requête INVITE du client, il en extrait l'URL To, consulte sa base de données pour vérifier si le destinataire existe et en fonction de cette vérification

il renvoie un message d'état au client source le notifiant que le destinataire n'est pas trouvé dans le cas où celui-ci n'existe pas. Dans le cas contraire il achemine la requête au destinataire à l'adresse IP contenue dans l'URL To pour signaler au destinataire une invitation à une communication.

3. En même temps que le proxy achemine la requête au destinataire, il envoie aussi un message d'état au client pour le notifier qu'il essaie d'établir un contact avec le destinataire.

4. Dès la réception de la requête INVITE par le client destinataire, un message d'invitation s'affiche à son écran lui indiquant le nom du client qui désire établir une communication. Un message d'état est envoyé de manière automatique au proxy qui l'acheminera vers le client source pour l'informer que le destinataire a reçu son invitation avec succès.

5. Le proxy reçoit la réponse du destinataire, il extrait l'URL *From* du message et de la même manière qu'à l'étape 2, il achemine celle-ci vers le client source. Un message *ringing* est affiché à l'écran de ce dernier.

6. Le client destinataire répond par un message *Ok* au client source et accepte ainsi son invitation à communiquer. Cette réponse est envoyée au proxy.

7. Le proxy reçoit la réponse Ok, il l'achemine au client source de la même manière qu'à l'étape 5.

8. À la réception de la réponse Ok, un message d'acquittement est envoyé au proxy automatiquement contenant les mêmes URL From et To que les messages échangés précédemment.

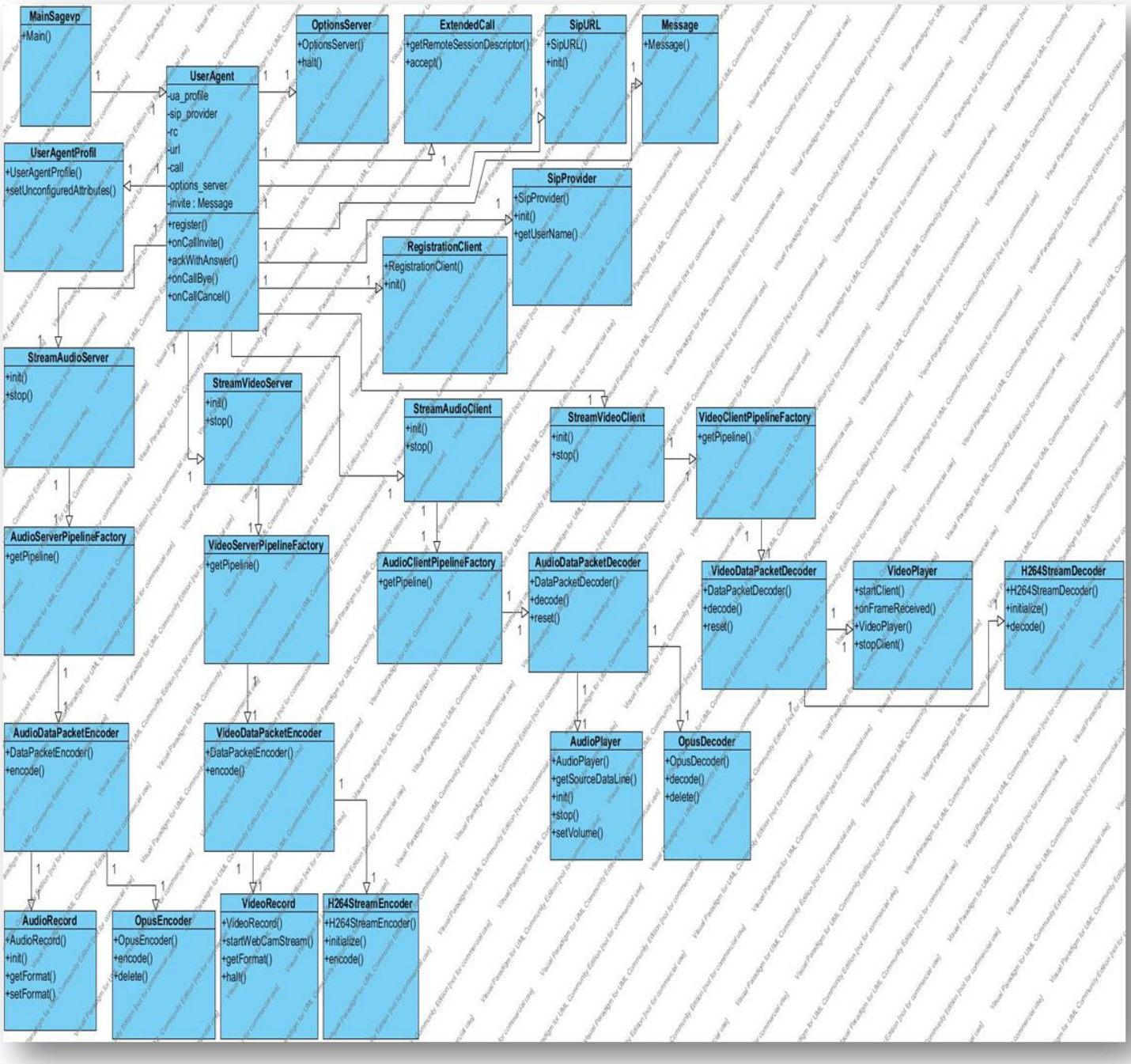
9. De la même manière qu'à l'étape 2, le proxy achemine ce message d'acquittement vers le destinataire.

10. La communication est établie dès la réception des messages d'acquittement *Ack.* a cette étape les paquets RTP et RTCP sont échangés entre les clients en multicast. Les ports RTP et RTCP sont attribués et gérés par le serveur proxy. Chaque fois qu'une communication est établie, le serveur proxy attribue à chaque client un port RTP pair et un port RTCP impair immédiatement supérieur.

11. Si un client désire mettre fin à sa participation à la conférence multimédia, il envoie une requête *Bye* au proxy qui l'achemine à son tour aux autres clients pour les informer du départ de celui-ci.

12. Les clients répondent par un message *Ok* au client qui désire terminer sa connexion via le proxy. Celui-ci libère les ports RTP et RTCP qu'il a attribué à ce client. (The Internet Society, 2002)

4. Diagrammes des classes de conception de notre application



Les composant clé de ce modèle est la classe **UserAgent,** dont l'extrait du code source et montré ci-dessous :

```
public class UserAgent extends CallListenerAdapter implements
CallWatcherListener, RegistrationClientListener
{


   // ***************************** attributes ****************************

   /** UserAgentProfile */
   protected UserAgentProfile ua_profile;

   /** SipProvider */
   protected SipProvider sip_provider;

   /** RegistrationClient */
   protected RegistrationClient rc=null;

   /** Call */
   protected ExtendedCall call;


   /** Register with the registrar server*/
   public void register()
   {  if (rc.isRegistering()) rc.halt();
     rc.register();
   }

   /** Register with the registrar server
     * @param expire_time expiration time in seconds */
   public void register(int expire_time)
   {  if (rc.isRegistering()) rc.halt();
      rc.register(expire_time);
   }
/** From CallListener. Callback function called when arriving a new INVITE
method (incoming call) */
   public void onCallInvite(Call call, NameAddress callee, NameAddress
caller, String sdp, Message invite)
   {  printLog("onCallInvite()",Log.LEVEL_LOW);
  // System.out.println("INVITE :"+call.getRemoteSessionDescriptor());
      if (call!=this.call) {  printLog("NOT the current call",Log.LEVEL_LOW);
return;  }
      // never called (the method onNewInocomingCall() is called instead): do
nothing.
   }
  /** From CallListener. Callback function called when arriving a CANCEL
request */
   public void onCallCancel(Call call, Message cancel)
   {  printLog("onCallCancel()",Log.LEVEL_LOW);
      if (call!=this.call) {  printLog("NOT the current call",Log.LEVEL_LOW);
return;  }
      printLog("CANCEL",Log.LEVEL_HIGH);
      this.call=null;
      // sound
      if (clip_ring!=null) clip_ring.stop();
      if (clip_off!=null) clip_off.play();
      // response timeout
      if (response_to!=null) response_to.halt();

      if (listener!=null) listener.onUaCallCancelled(this);
   }
```

Figure 3 : Extrait de code source d'implémentation de la classe principale 'UserAgent'

5. Architecture de l'Application

La figure suivante montre l'architecture de notre application proposée et les interactions entre les différentes composantes durant le processus d'échange des messages SIP et de données multimédias (voix et vidéo).

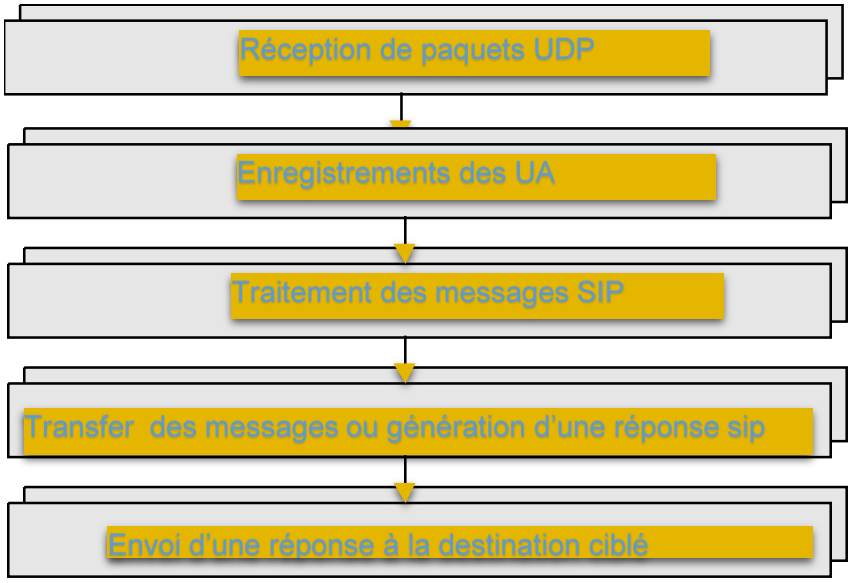


Figure 4 : Architecture de notre application

5.1 Description des composantes de l'architecture

5.1.1 Agent utilisateur client-serveur

Le Client SIP est la composante conçu pour initier une session de conversation sur IP.il se charge de la création et de la terminaison d'un dialogue.

Et ce elle qui s'occupe de la génération de toutes les demandes clientes en mettant en évidence toute les méthodes de demandes via le proxy.

Cette application client est un serveur en même temps, celui-ci traite les réponses SIP qui proviennent du proxy, et génère d'autres messages tels que le message ringing, et les acquittements.

5.1.2 Traitement de données multimédia (voix et vidéo)

C'est le module charger de la capture de données multimédia (voix et vidéo), à partir de sources tel que le **MIC (pour l'audio)** et la **Webcam (pour les images)**, et ensuite les données sont traitaient soit pour l'envoi sur le réseau soit pour l'audition et la visualisation.

5.1.3 Gestion du fichier de Contacts SIP

Il s'agit d'un fichier qui doit permettre la persistance de contacts SIP pour les utilisateurs, en quelque sorte un système d'annuaire.

5.1.4 Interface Graphique

L'interface graphique comme le nom l'indique, permet de cacher à l'utilisateur la complexité de l'implémentation, en lui fournissant l'accès au client SIP à un niveau d'abstraction plus élevé.

5.1.5 Proxy

Ce serveur a été conçu pour procurer un service au client, pour analyser et interpréter des invitations de sessions basées sur le protocole SIP et pour générer des réponses aux clients. Il permet aussi aux utilisateurs de s'enregistrer dans sa liste. La figure suivante montre les tâches successives du proxy dans un processus de signalisation SIP qui consistent en la réception des paquets UDP, l'enregistrement du client dans le cas où celui-ci se connecte pour la première fois, ensuite les paquets sont traités par le proxy pour décider de la réponse à renvoyer au client source ou leurs acheminements vers le destinataire.

Le serveur Proxy a possibilité de gérer aussi les ports RTP/RTCP dans une communication audio/visuelle. Dans le but de pouvoir réaliser ce modèle, nous avons implémentés notre application avec une approche orientée objet en utilisant le langage Java. De plus, grâce au Framework **javax.sound.sampled** a développé par Sun Microsystems qui inclut les classes telles que **(TargetDataLine)** qui facilite l'implémentation de la partie récupération des Stream (flux de données voix) sur le **MIC.** Pour cela, nous avons utilisé un kit de développement JDK1.8 de java.

6. Implémentation et test de l'application

Dans cette section nous décrivons de manière générale le modèle client-serveur en Java qui constitue la base de notre structure SIP. Nous présentons aussi les différentes classes java qui composent les différents programmes de notre application.

6.1.1 Les différentes API et FRAMEWORK du langage JAVA utilisés

6.1.1.1 MJSIP :

MjSip inclut toutes les classes et méthodes pour créer une application SIP. Il implémente la pile en couches complète, L'architecture telle que définie dans le RFC 3261, et est entièrement conforme à la norme. En outre, il comprend des interfaces de niveau supérieur pour le contrôle d'appel.

Il est formé par plusieurs paquets incluant:

- objets SIP standard tels que les messages SIP, les transactions, les dialogues, etc.,
- diverses extensions SIP déjà définies dans l'IETF,
- API de contrôle d'appel,
- une implémentation de référence de certains systèmes SIP (serveurs et UA). (mjsip, 2012)

6.1.1.1.1 Quelques raisons d'utiliser MjSip

Il existe plusieurs implémentations disponibles de SIP, dans les langages de programmation Java et C ++; MjSip est juste un autre.

Les principales caractéristiques de MjSip sont:

- Il est basé sur Java, donc il est multiplate-forme,

- Ce n'est pas seulement une API, mais inclut l'implémentation complète de la pile SIP,

- Il est très puissant et conforme à la norme IETF RFC 3261 et aux extensions,

- il est simple à utiliser et très simple à étendre (eh bien, c'est ce que nous pensons ... ;-)),

- Il est très léger et peut être utilisé simultanément pour les implémentations des serveurs et des terminaux légers. (mjsip, 2012)

6.1.1.1.2 Architecture MjSip

Selon l'architecture SIP définie dans RFC 3261, le MjSip principal est structuré en trois couches de base: Transport, Transaction et boîte de dialogue. En plus de ces couches, MjSip fournit également des niveaux de contrôle d'appel et de niveau d'application, avec le API correspondantes.

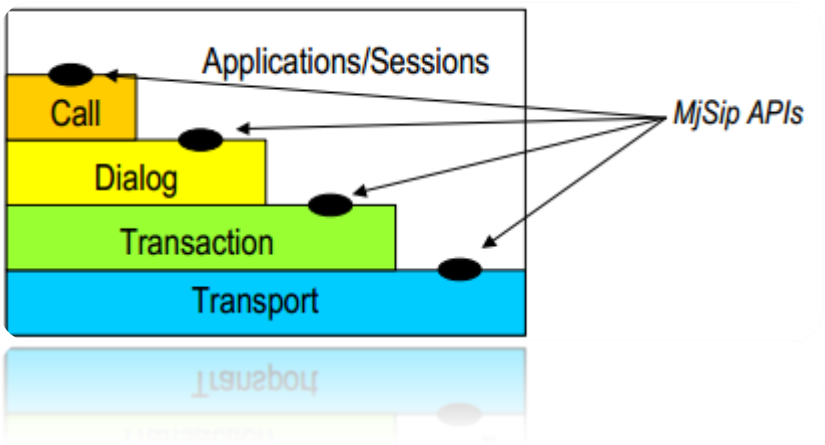


Figure 5 : Les couches de Base de MjSip

La couche la plus basse est le **Transportlayer** qui fournit le transport de messages SIP. Il est responsable d'envoyer et de recevoir des messages SIP via différents protocoles de transport de couche inférieure et **démultiplexer** les messages entrants vers la couche supérieure appropriée.
La deuxième couche est la **Transactionlayer**. Les transactions sont une composante fondamentale du SIP. Dans SIP, une transaction est une demande envoyée par un client (un client de transaction) à un serveur de transaction avec toutes les réponses à cette demande envoyée à partir du serveur de transaction retourne au client. La couche de transaction gère les retransmissions de couches supérieures, la correspondance des réponses aux demandes et aux délais. La couche de transaction envoie et reçoit des messages via la couche de transport.

La troisième couche (au-dessus de la couche de transaction) est la Dialog qui lie les différentes transactions au sein de la même "session". UNE
Dialogue est une relation SIP pair-à-pair entre deux agents utilisateurs qui persiste pendant un certain temps. Le dialogue facilite le Séquençage des messages et bon déroulement des demandes entre les agents utilisateurs.
INVITE est la méthode qui établit une boîte de dialogue (appelée boîte de dialogue d'invitation). Une boîte de dialogue invite est implémentée dans MjSip par la classe **InviteDialog**.
La couche SIP supérieure est le contrôleur d'appel qui implémente un appel SIP complet. La couche Contrôle d'appel est Implémentée par CallAPI, qui offre une interface simple à utiliser pour gérer les appels SIP entrants et à venir. Un appel
Peut comporter plus d'une boîte de dialogue.
Sur ces quatre couches, il y a les sessions SIP qui relient deux ou plusieurs entités d'application (participants) à Différents systèmes. (mjsip, 2012)

6.1.1.2 JAVAX.SOUND :

Est un API de bas niveau pour la manipulation de flux Audio d'entrée/sortie. Elle permet la capture, la lecture des flux audio et la manipulation de ceux-ci (effet mixage etc.), contrairement à l'api JMF, il permet un contrôle total sur celui-ci. (Antoine, 2006)

Il est composé de deux sous Package ayant rapport aux domaines de java sound :

- **javax.dound.Sampled** qui comme son l'indique est dédié au traitement des sons échantillonnés de sources analogique.

- **javax.dound.midi** qui est lui dédié à tout ce qui touche à la synthèse de son.

Extrait du code qui s'occupe de l'initialisation des paramètres d'échantillonnages :

```
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.TargetDataLine;

public class AudioRecord {

      private int sampleRate = 8000;
      private int channels = 1;
      private int sampleSizeInBits = 16;
      private boolean signed = true;
      private boolean bigEndian = false;
      private TargetDataLine targetLine = null;

      private AudioFormat format = null;
      private boolean isOpen = false;

      public AudioRecord(int sampleRate, int sampleSizeInBits,
int channels, boolean signed, boolean bigEndian) {
            super();
            this.sampleRate = sampleRate;
            this.channels = channels;
            this.sampleSizeInBits = sampleSizeInBits;
            this.signed = signed;
            this.bigEndian = bigEndian;

            init(sampleRate, sampleSizeInBits, channels, signed,
bigEndian);
      }
```

Figure 6 : Extrait de code d'initialisation des paramètres d'échantillonnage de son.

6.1.1.3 Librairie jna

Pour le codec OPUS nous avons fait usage d'un interfaçage entre son implémentation native et le langage java en passant par la **librairie jna** qui est supporté par la communauté open source sur **github**. Dans le code source la méthode qui effectue l'encodage est opus_encode(). (Github, Java native access, 2012)

Voici un extrait du code effectuant les imports :

```
import com.sun.jna.Library;

import com.sun.jna.Native;

import com.sun.jna.NativeLibrary;

import com.sun.jna.Pointer;

import com.sun.jna.PointerType;

import com.sun.jna.ptr.FloatByReference;

import com.sun.jna.ptr.IntByReference;

import com.sun.jna.ptr.PointerByReference;

import com.sun.jna.ptr.ShortByReference;
```

Figure 7 : Importation des éléments de la librairie jna pour l'interfaçage du langage Java et le codec OPUS.

6.1.1.4 Netty

Netty est un framework java permettant de développer des applications réseaux asynchrones, maintenables et performantes. Il contient un API bien conçu qui déploie une couche d'abstraction au-dessus de java. Elle est simple à utiliser et découpe proprement le code en morceaux logique appelés « Handler », ce qui permet de développer des applications réseaux simple à maintenir. (project, 2003)

Cette abstraction n'entrave pas les performances de l'application car les possibilités de configuration sont très importantes.

Netty dispose également de nouveaux buffers optimisés et simple d'utilisation.

Il propose de nombreuses fonctionnalités utiles au quotidien pour les développeurs :

- La gestion de plusieurs protocoles de la couche transport, comme UDP et TCP.
- La gestion HTTP : construction des entêtes, transformation des messages en objets java.
- La gestion de websockets.
- La gestion de SSL
- La gestion de la compression.

Ainsi dans notre application l'avons utilisé jusque-là au niveau de la gestion de protocoles de transports.

6.1.1.5 Webcam Capture API

C'est une bibliothèque qui permet d'utiliser la **webcam** intégrée ou externe directement depuis Java. Il est conçu pour abstraire les fonctions de caméra couramment utilisées et prendre en charge diverses fonctionnalités de capture. (Github, Webcam capture api, 2016)

6.1.1.6 Xuggler

Xuggler est une bibliothèque Java qui permet de décoder et Codé une variété de formats de fichiers multimédias directement à partir de Java. Il est construit sur le FFMPEG, mais est conçu avec les objectifs suivants:

- Facilité d'utilisation ;
- Sécurité ;
- Portabilité.

Contrairement à la plupart des bibliothèques Java, Xuggle possède un code natif (par exemple, une bibliothèque partagée par Windows DLL ou Linux) composant qui doit être installé avec lui. (Xuggle, 2010)

6.1.1.7 Extrait des codes sources d'implémentation de certaines entités de l'Application

Il faut retenir à ce niveau que nous avons effectué des tests assez concluant lors de l'implémentation de l'application. Cependant pour le module de signalisation, nous avons choisi d'utiliser le protocole de transport UDP et le port 5060. Ce dernier est le port par défaut utilisé dans le protocole SIP, bien qu'il nous était possible de choisir tout autre port public.

Voici maintenant quelques extraits de code source pour certaines entités de l'architecture globale de 'architecture de l'application.

6.1.1.7.1 Transmission audio:

La Méthode run() contenue dans ServerChannelHandler qui permet d'écrire les données audio sur le réseau :

```
public void run() {
            try {
                  ShortBuffer shortBuffer = recordFromMic();
                  Object msg = opusEncoder.encode(shortBuffer);
                  ByteBuffer test = (ByteBuffer)msg;
                  DataPacket packet = new DataPacket();
                  packet.setData(test.array());
                  packet.setPayloadType(8);
                  packet.setSequenceNumber(seq);
                  packet.setTimestamp(System.currentTimeMillis());

                  if (msg != null) {
                        channelGroup.write(packet);// ecriture le canal
                  }
                  seq ++;
            } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
               }
```

Figure 8 : Extrait du code de la méthode run() pour l'écriture des données sur le réseau

6.1.1.7.2 Transmission vidéo :

La méthode **encode()** est contenue dans la classe **H264Encoder** qui permet la compression des données récupérées via la webcam.

```
public Object encode(Object msg) throws Exception {

            if (msg == null) {
                  return null;
            }
            if (!(msg instanceof BufferedImage)) {
                  throw new IllegalArgumentException("your need to pass
into an bufferedimage");
            }
            logger.info("encode the frame");
            BufferedImage bufferedImage = (BufferedImage)msg;
            //here is the encode
            //convert the image
            BufferedImage convetedImage =
ImageUtils.convertToType(bufferedImage, BufferedImage.TYPE_3BYTE_BGR);
            IConverter converter =
ConverterFactory.createConverter(convetedImage, Type.YUV420P);
            //to frame
            long now = System.currentTimeMillis();
            if (startTime == 0) {
                        startTime = now;
            }
            IVideoPicture pFrame = converter.toPicture(convetedImage,
(now - startTime)*1000);

            iStreamCoder.encodeVideo(iPacket, pFrame, 0) ;
            //free the MEM
            pFrame.delete();
            converter.delete();
            //write to the container
            if (iPacket.isComplete()) {

                  //iPacket.delete();
                  //here we send the package to the remote peer
                  try{
                        ByteBuffer byteBuffer = iPacket.getByteBuffer();
                        if (iPacket.isKeyPacket()) {
                              logger.info("key frame");
                        }
                        ChannelBuffer channelBuffe =
ChannelBuffers.copiedBuffer(byteBuffer.order(ByteOrder.BIG_ENDIAN));
                        if (frameEncoder != null) {
                              return frameEncoder.encode(channelBuffe);
                        }
                        return channelBuffe;

                  }finally{
                        iPacket.reset();
                  }
            }else{
                  return null;
            }
        }
```

Figure 9 : Extrait de code de la compression des données récuoérées via la webcam.

6.1.1.7.3 Invitation

Voici la méthode **call** contenu dans la classe **UserAgent** qui permet dans lancer l'appel avec la méthode **invite** de la spécification **sip**.

```
        /** Makes a new call (acting as UAC) with specific media
description (Vector of MediaDesc). */
   public void call(NameAddress callee, Vector media_descs)
   {
      // new media description
      if (media_descs==null) media_descs=ua_profile.media_descs;
      this.media_descs=media_descs;
      // new call

     printLog("DEBUG:
auth_user="+ua_profile.auth_user+"@"+ua_profile.auth_realm,Log.LEVE
L_HIGH);

      call=new
ExtendedCall(sip_provider,ua_profile.getUserURI(),ua_profile.auth_u
ser,ua_profile.auth_realm,ua_profile.auth_passwd,this);

      if (ua_profile.no_offer) call.call(callee);

      else
      {  SessionDescriptor
local_sdp=getSessionDescriptor(media_descs);
         call.call(callee,local_sdp.toString());
      }
      progress=false;
      ringing=false;
   }
```

Figure 10 : Extrait de code permettant d'initier une invitation dans un appel de la spécification SIP.

6.1.1.7.4 Terminaison d'appel SIP

Un client qui désire mettre fin à sa connexion envoie une requête BYE vers le serveur proxy. Le proxy achemine cette requête vers tous les participants, la transmission et la réception des paquets audio et vidéo sont arrêtés et le proxy dés enregistre le client, remet à jour sa liste de clients et libère tous les ports de transmission et réception multimédia. Voici la méthode **hangup** contenu dans la classe **UserAgent** qui permet de terminer l'appel avec la méthode **Bye** de la spécification SIP.

```
    /** Closes an ongoing, incoming, or pending call. */
 public void hangup()
 {  // sound
    if (clip_progress!=null) clip_progress.stop();
    if (clip_ring!=null) clip_ring.stop();
    // response timeout
    if (response_to!=null) response_to.halt();

    closeMediaSessions();
    if (call!=null) call.hangup();
    call=null;
 }
```

Figure 11 : Extrait de code pour la terminaison d'un appel SIP.

7. Évaluation de la complexité du logiciel (métriques et nombre cyclomatique de McCabe)

Pour quantifier la complexité d'un logiciel, les mesures les plus utilisées sont les lignes de code (LOC acronyme de « lines of code ») puisqu'elles sont simples, faciles à comprendre et à compter. Un essai simple à l'aide d'**Eclipse Metric** nous a permis d'évaluer par ailleurs la complexité cyclomatique du code en calculant notamment le nombre cyclomatique de McCabe. Tout ceci nous l'avons réalisé en utilisant le plugin **Metrics** d'eclipse que l'on a intégré dans **l'IDE eclipse** pour l'évaluation de différentes métriques de notre application. Une complexité cyclomatique trop élevée (supérieure à 30) indique qu'il faut refactoriser la méthode. Une complexité cyclomatique inférieure à 30 peut être acceptable si la méthode est suffisamment testée. La complexité cyclomatique est liée à la notion de "code coverage", c'est à dire la couverture du code par les tests. Dans l'idéal, une méthode devrait avoir un nombre de tests unitaires égal à sa complexité cyclomatique pour avoir un "code coverage" de 100%. Cela signifie que chaque chemin de la méthode a été testé. (verifysoft, 2016)

7.1 Eclipse Metric

Eclipse Metrics est un plugin eclipse qui analyse le code source pour calculer un certain nombre de métriques. Il est également capable de présenter un graphe des dépendances entre paquetages en 3D. Eclipse Metrics s'installe depuis le gestionnaire de mise à jour d'eclipse (URL : http://metrics.sourceforge.net/update). Après installation, il faut l'activer sur chaque projet à mesurer grâce au panneau de configuration des propriétés du projet (onglet Metrics). Ensuite, il suffit d'afficher la vue Metrics (Window/Show View/Metrics View) pour obtenir les statistiques de l'élément sélectionné dans l'explorateur d'Eclipse. La figure ci-dessous (figure 12) présente le résultat de l'exécution de Metrics sur le code "**AudioOutputStream.java**", présenté dans la section "Complexité Cyclomatique".

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Parameters (avg/max per method)		1	1	3	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	write
Number of Static Attributes (avg/max per type)	0	0	0	0	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Specialization Index (avg/max per type)		1	0	1	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Number of Classes	1					
Number of Attributes (avg/max per type)	2	2	0	2	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Number of Static Methods (avg/max per type)	0	0	0	0	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Number of Interfaces	0					
Total Lines of Code	35					
Weighted methods per Class (avg/max per type)	8	8	0	8	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Number of Methods (avg/max per type)	8	8	0	8	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Depth of Inheritance Tree (avg/max per type)		2	0	2	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
McCabe Cyclomatic Complexity (avg/max per method)		1	0	1	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	AudioOutputStream
Nested Block Depth (avg/max per method)		1	0	1	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	AudioOutputStream
Lack of Cohesion of Methods (avg/max per type)		0,5	0	0,5	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Method Lines of Code (avg/max per method)	10	1,25	0,433	2	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	AudioOutputStream
Number of Overridden Methods (avg/max per type)	4	4	0	4	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	
Number of Children (avg/max per type)	1	1	0	1	/SageVp/src/com/ul/sagevp/sip/zoolu/sound/Audio...	

Figure 12 : Capture du résultat du calcul par metrics des paramètres de complexité du code source "**AudioOutputStream.java**".

Enfin, voici un exemple de graphe de dépendances en 3D. Metrics colore les cycles. Il est possible de ne voir que les dépendances d'un paquetage particulier en double-cliquant sur ce paquetage, comme le montre la figure 13 dans laquelle on a choisi de centrer la vue sur le paquetage **com.ul.sagevp.local.codec.audio** : Eclipse Metrics est un outil vraiment très utile pour faire ressortir les défauts architecturaux d'un projet.

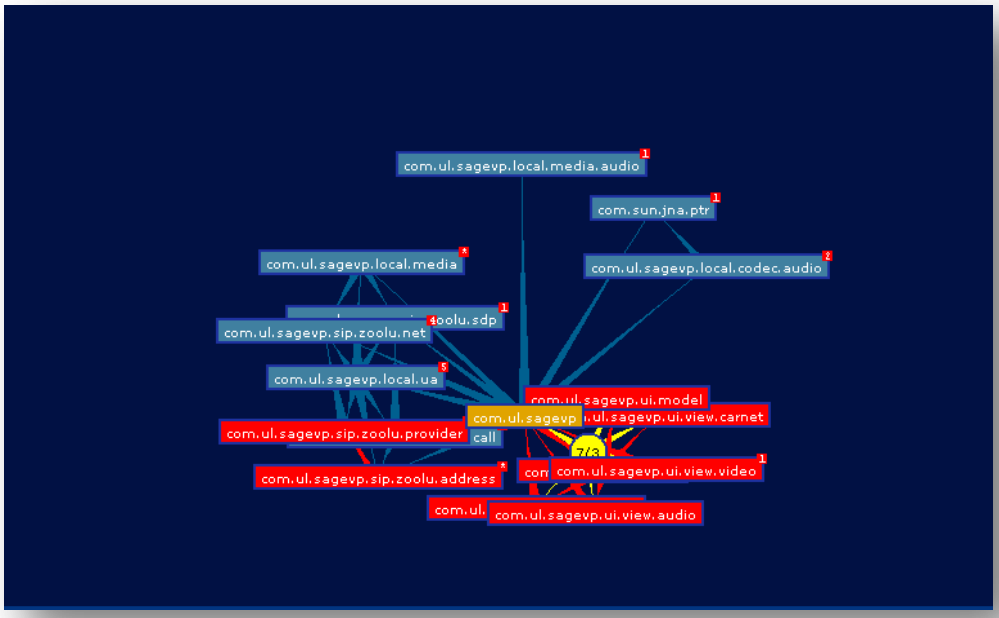


Figure 13 : Graphe de dépendance centré sur le paquetage **com.ul.sagevp.local.codec.audio**

7.2 Le nombre de lignes de codes du projet global

La figure ci-dessous montre les métriques de nombre de ligne de codes pour notre projet (SageVp) en général y comprit aussi les codes venant de l'API MGSIP. Au total **27070** ligne de codes dans lequel on soustrait **11395** lignes de codes venant de l'API MGSIP importé du projet **zoolu** d'où **15675** ligne de codes produit par notre équipe.

Ainsi pour évaluer la complexité par rapport à cette métrique nous nous intéressons au nombre de lignes de codes par fichier source .**java**, étant donné que un fichier ne doit pas dépasser 400 lignes de codes donc doit contenir au plus 10 méthodes, car chaque méthode doit avoir au plus 40 lignes de codes.

- Ligne de codes des fichiers du paquetage « **com.ul.sagevp.local.ua** » qui est le seul paquetage dans lequel il y a un fichier (**UserAgent**.**java**) qui contient plus de 400 lignes de codes soit **539**.

Etant donné que nous avons **39 Package et 10 classes en moyenne** par package et seule une classe sur un total de 314 a un dépassement de **139** de ligne de codes par rapport à la valeur maximum qu'un fichier doit avoir un maximum de 400 lignes. En général la **maintenance** et le **test** de l'application ne pose pas problème, et le **taux de défaut est négligeable**.

8. Conclusion

La complexité d'un logiciel est un **indicateur** de **performance** de ce dernier, ainsi nous avons mesurés les métriques de ligne de codes, le nombre cyclomatique de McCABE dans le but de déterminer jusqu'à ce niveau du code l'évolution de la complexité.

Etant donné que nous continuons avec le développement de certains aspects du projet, nous allons très prochainement mesurer aussi les métriques de complexité orientées objet de **Halstead** et **L'index de maintenabilité** pour parvenir à évaluer **le coup de la correction du logiciel** par rapport **à sa réécriture.**

9. Bibliographie

Antoine, R. (2006). l'api Java sound. Récupéré sur igm.univ-mlv: http://igm.univ-mlv.fr

Github. (2012). Java native access. Récupéré sur github: https://github.com

Github. (2016). Webcam capture api. Récupéré sur github: https://github.com/sarxos/webcam-capture

J-F. Rey, C. T. (2002). La technologie SIP dans l'entreprise. *Technology white paper, 4e Trimestre* , 3.

Kuhn, L. (2004). *SIP Session Initiation Protocol.* Paris.

Litzistorf, P. G. (2006). Best Practices for VoIP-SIP Security.

mjsip. (2012, avril 22). MjSip stack 1.5.4 and reference applications with source files. Parma. Récupéré sur MjSip: http://www.mjsip.org

Mohamed, T. B. (2009). transmission média sur les réseaux ip en utilisant les protocoles sip. Récupéré sur http://www.recentdocuments.com/doc/5-1/

Oussema, D. (2013). *Gestion des risques dans les infrastructures VoIP.* Lorraine .

project, T. N. (2003). Netty project. Récupéré sur netty.io: https://netty.io

Rohan, M. (2004). *Paquetage d'événement Résumé de message et Indication de message en instance pour le protocole d'initialisation de session (SIP).* USA.

The Internet Society, T. T. (2002). Protocole d'initialisation de session(SIP : RFC 3261).

Thomas, G. (2010). *Sécurité de la téléphonie sur IP.* Paris.

Touti, B. (2011). *Conception et implémentation d'un gestionnaire de flux SIP.*

verifysoft. (2016). McCabe Metrics. Germany. Récupéré sur verifysoft: http://www.verifysoft.com

Warodom, W. (2012). *Architectures de réseaux pour la délivrance de services à domicile.* Toulouse.

Xuggle. (2010). To encode, decode, and generally juggle audio and video files in any way that you want. Récupéré sur xuggle: http://www.xuggle.com