International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:15, No:4, 2021

Design, Development by Functional Analysis in UML and Static Test of a Multimedia Voice and Video Communication Platform on IP for a Use Adapted to the Context of Local Businesses in Lubumbashi

Blaise Fyama, Elie Museng, Grace Mukoma

***Abstract*—**In this article we present a java implementation of video telephony using the SIP protocol (Session Initiation Protocol). After a functional analysis of the SIP protocol, we relied on the work of Italian researchers of University of Parma-Italy to acquire adequate libraries for the development of our own communication tool. In order to optimize the code and improve the prototype, we used, in an incremental approach, test techniques based on a static analysis based on the evaluation of the complexity of the software with the application of metrics and the number cyclomatic of Mccabe. The objective is to promote the emergence of local start-ups producing IP video in a well understood local context. We have arrived at the creation of a video telephony tool whose code is optimized.

***Keywords*—**Static analysis, coding, complexity, mccabe metrics, Sip, uml.

I. Introduction

IN this article we present a java implementation of video telephony using the SIP protocol. This article aims, in particular, the study of the protocols used in VoIP, the different architectures proposed, and the development of a VoIP application carrying out signaling using the SIP protocol and facilitating the transport of voice and video. Indeed, there are software such as Skype, and Lynk from Microsoft that do multimedia communication. However, our objective is part of the use and manipulation of standards developed by telecommunications organizations such as the ITU-T and the IETF. This will help facilitate the interoperability of equipment and applications in this area. This application is implemented with an object oriented approach using the Java programming language for reasons of portability and reuse. It is therefore a question of developing an application based on the client-server model, of analyzing and processing the messages exchanged between different entities. Another part of our application is to use specific protocols that allow us to route voice and video over IP.

Congolese companies, benefiting from our solution, will be able to set up a fairly flexible, inexpensive, and secure VoIP platform.

Basically our methodology is as follows:

1. Analysis and comparison of multimedia communication standards (H.323 and SIP) [1].
2. Vulnerability study VoIP and good security practices
3. Application architecture modeling
4. Object-oriented design (UML representation, class diagram) [2].
5. Object oriented implementation using the Java programming language [2].
6. Application of metrics from McCabe for source code evaluation and optimization [3].
7. Application test and results. We also evaluated the complexity of the software with the application of metrics and the numbercyclomatic of Mccabe.

Ultimately, our approach gives Congolese companies the opportunity to obtain an inexpensive messaging and VoIP solution with local maintenance.

II. Overview of the SIP Protocol

SIP is a signaling protocol defined by the IETF (Internet Engineering Task Force) allowing the establishment, release and modification of multimedia sessions (RFC 3261) [4]. It inherits certain functionalities from the HTTP (Hyper Text Transport Protocol) protocols used to navigate the WEB, and SMTP (Simple Mail Transport Protocol) used to transmit electronic messages (e-mails) [5].

SIP is based on a client/server transactional model like HTTP. Addressing uses the concept of Uniform Resource Locator (SIP) which resembles an e-mail address. Each participant in a SIP network can therefore be addressed by a SIP URL [4]. In addition, SIP requests are acknowledged by responses identified by a digital code. In fact, most of the SIP response codes have been borrowed from the HTTP protocol. For example, when the recipient is not located, a response code "404 Not Found" is returned.

A SIP request consists of headers like an SMTP command. Finally SIP as SMTP is a textual protocol.

SIP has been extended to support many services such as presence, instant messaging (similar to SMS service in mobile networks), call transfer, conference, complementary telephony services, etc.

The SIP protocol is only a signaling protocol. Once the session is established, the participants of the session directly exchange their audio/video traffic through the RTP protocol

Blaise Fyama Mwepu is with the UPL Université Protestante de Lubumbashi, Congo, The Democratic Republic Of The (e-mail: bfyama@gmail.com).

(Real-Time Transport Protocol). It is a call control protocol, not media control. SIP is also not a file transfer protocol such as HTTP, used to transport large volumes of data. It has been designed to transmit short signaling messages in order to establish, maintain and release multimedia sessions. Note that SIP has the advantage of not being attached to a particular medium and is supposed to be independent of the transport protocol [5]. Fig. 1 shows the standard structure of the SIP protocol.

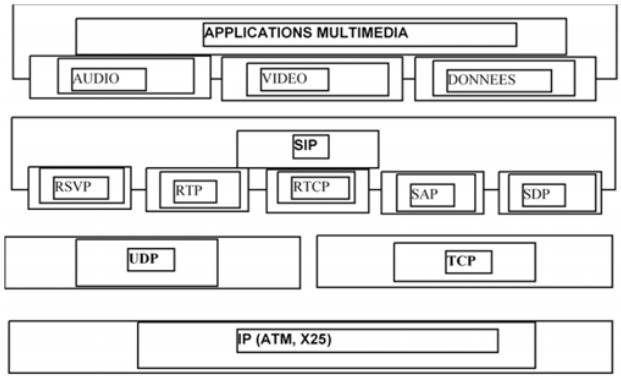

Fig. 1 Standard structure of the SIP Protocol

Fig. 1 is the SIP protocol stack, according to which:

1. SIP initiates the multimedia session (audio, video).
2. RSVP: Internet resource reservation protocol for the deployment of the real-time application.
3. SDP: Multimedia session description protocol.
4. RTP, RTCP participate in the transport of real-time data exchanged between participants in an SIP session.
5. UDP, TCP, transport multimedia data.
6. IP represents the communication network used.

III. How a SIP Messaging System Works

Fig. 2 represents the SIP protocol dialogue system and the exchange of RTP/RTCP packets of the application between two client-server user agents (UAC/UAS) through a proxy server.

Fig. 2 represents: The initiation of SIP sessions, the exchange of RTP/RTCP packets as well as the termination of the application session between two client-server user agents (UAC/UAS) through a proxy server

Succession of stages:

1. Client A who wishes to initiate a communication issues an INVITE request containing the From, To and Via URLs. This is sent to the proxy server.
2. The proxy receives the INVITE request from the client, it extracts the URL “To”, consults its database to check if the recipient exists and based on this verification it sends a status message to the source client notifying it that the recipient is not found in case this one does not exist. Otherwise, it forwards the request to the recipient at the IP address contained in the URL “To” to notify the recipient of an invitation to communicate.
3. At the same time that the proxy routes the request to the recipient, it also sends a status message to the client to notify it that it is trying to establish contact with the recipient.

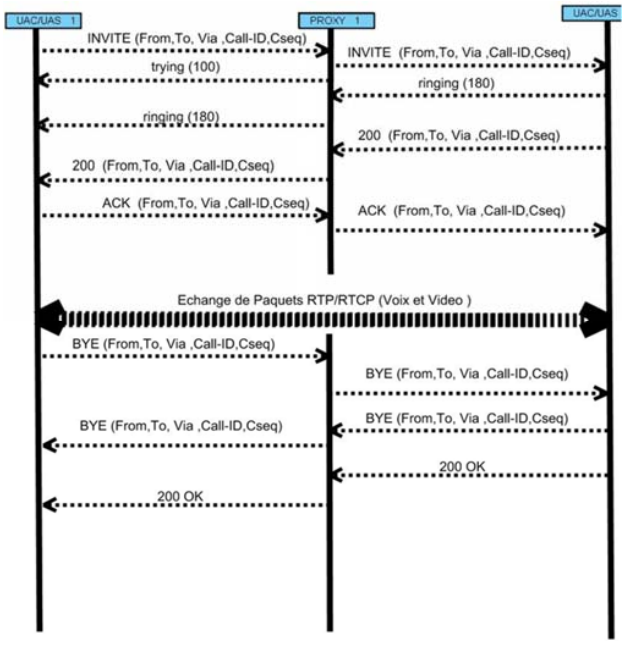

Fig. 2 Operating principle of a SIP messaging system

4. Upon receipt of the INVITE request by the recipient client, an invitation message is displayed on its screen indicating the name of the client who wishes to establish a communication [4]. A status message is automatically sent to the proxy which will route it to the source client to inform them that the recipient has successfully received their invitation.
5. The proxy receives the response from the recipient, it extracts the From URL from the message and in the same way as in step 2, it routes it to the source client. A ringing message is displayed on the latter's screen.
6. The recipient client responds with an Ok message to the source client and accepts their invitation to communicate. This response is sent to the proxy.
7. The proxy receives the response Ok, it routes it to the source client in the same way as in step 5.
8. Upon receipt of the Ok response, an acknowledgment message is sent to the proxy automatically containing the same URLs “From” and “To” as the messages exchanged previously.
9. In the same way as in step 2, the proxy routes this acknowledgment message to the recipient.
10. Communication is established upon receipt of Ack acknowledgment messages. At this stage the RTP and RTCP packets are exchanged between the clients in multicast. RTP and RTCP ports are assigned and managed by the proxy server [6]. Whenever communication is established, the proxy server assigns each client an even RTP port and an immediately higher odd RTCP port.

11. If a client wishes to end his participation in the multimedia conference, he sends a Bye request to the proxy, which in turn routes it to the other clients to inform them of his departure.
12. Clients respond with an Ok message to the client who wishes to terminate his connection via the proxy. This frees up the RTP and RTCP ports that he assigned to this client [7], [5].

IV. Diagrams of the Design Classes of Our Application

Fig. 3 represents the fundamental class diagram of our SIP application in which we present the relations between these which make the system functional the key component of this model is the UserAgent class, whose extract from the source code is shown in Fig. 4.

The class in Fig. 4 represents the implementation of methods according to the following import attributes:

1. The userAgent component profile
2. The SIP Provider
3. SIP client registration
4. SIP call

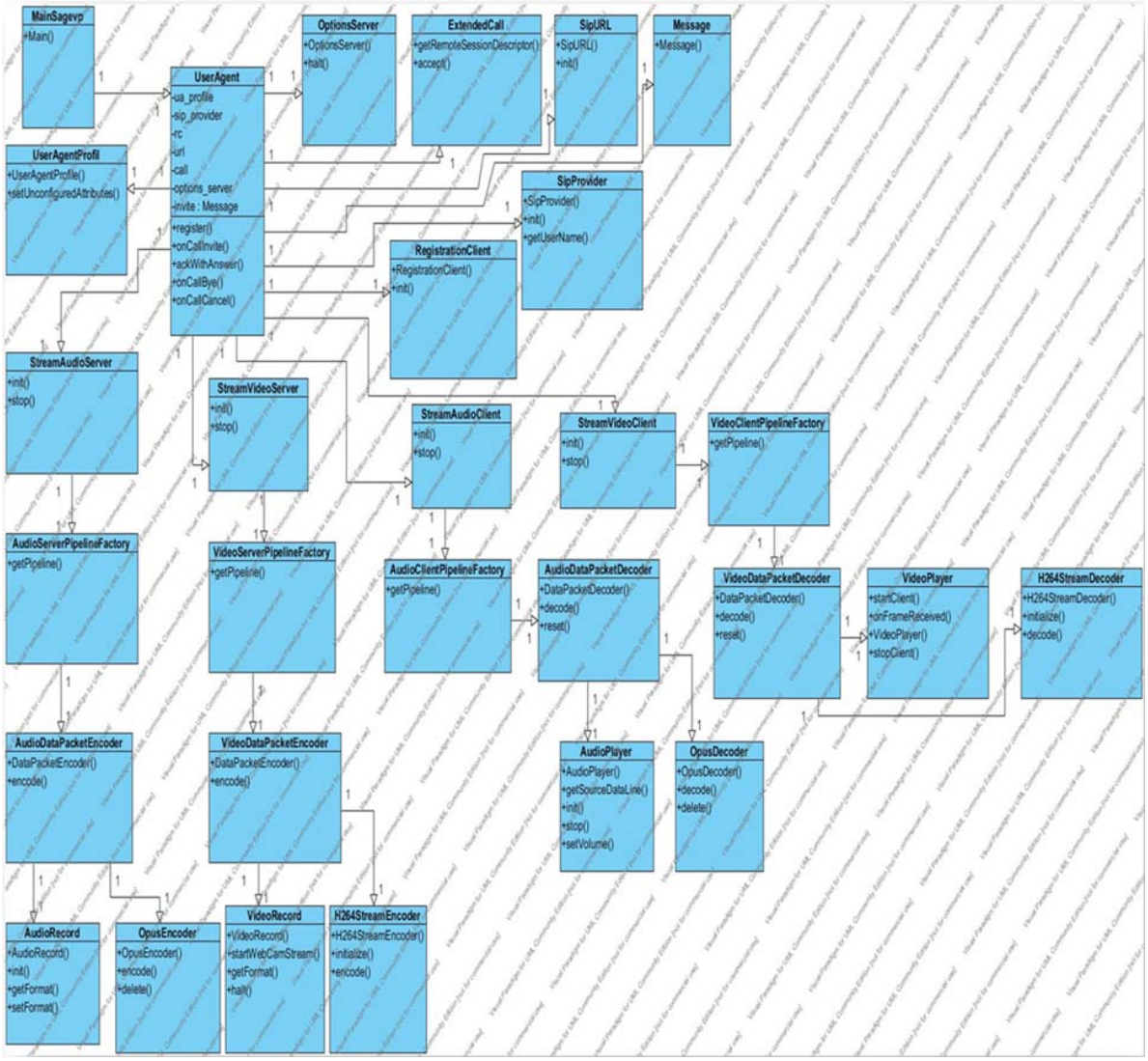

Fig. 3 Fundamental class diagram

```
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);
   }
```

Fig. 4 Implementation source code extract of the main class 'UserAgent'

V. Application Architecture

Fig. 5 shows the architecture of our proposed application and the interactions between the different components during the process of exchanging SIP messages and multimedia data (voice and video).

A. Description of the Components of the Architecture

1. Client-Server User Agent

The SIP Client is the component designed to initiate a conversation session over IP. It is responsible for the creation and termination of a dialogue. And this class takes care of the generation of all client requests by highlighting all the methods of requests via the proxy.

This client application is a server at the same time, which processes the SIP responses that come from the proxy, and generates other messages such as message ringing, and acknowledgments [16], [17].

2. Multimedia Data Processing (Voice and Video)

This is the module responsible for capturing multimedia data (voice and video), from sources such as the MIC (for audio) and the Webcam (for images), and then the data are processed for either sending over the network either for hearing and viewing.

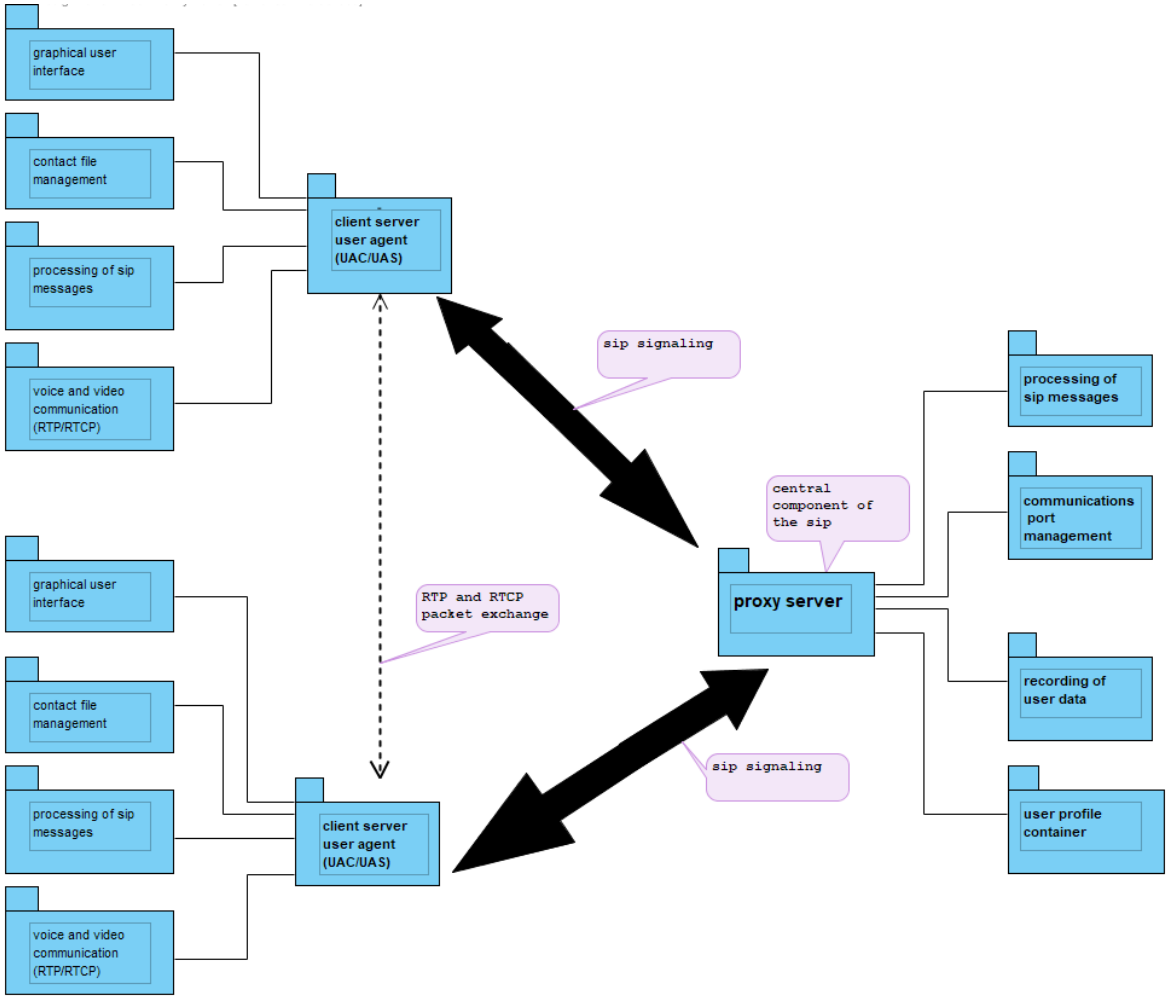

Fig. 5 Architecture of our application: This architecture is separated into two logical parts, on the one hand the functions included in the SIP application module and on the other hand by those of the Proxy server

3. Managing the SIP Contacts File

It is a file which should allow the persistence of SIP contacts for users, in a way a directory system.

4. Graphic Interface

The graphical interface as the name suggests, allows hiding the complexity of the implementation from the user, by providing access to the SIP client at a higher level of abstraction.

5. Proxy

This server was designed to provide service to the client, to analyze and interpret session invitations based on the SIP protocol and to generate responses to clients. It also allows users to be registered. The successive tasks of the proxy in a SIP signaling process consist in receiving UDP packets, registering the client in the event that the latter connects for the first time, then the packets are processed by the proxy to decide on the response to be returned to the source client or their routes to the recipient [5].

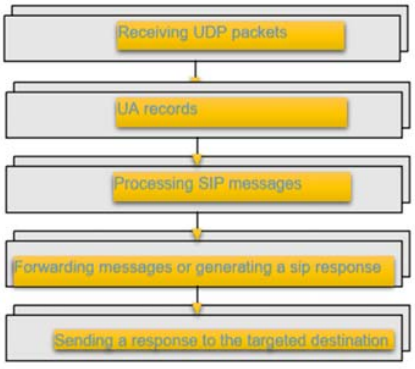

Fig. 6 Proxy server services, UDP packet reception, Registration of UAs, Processing of SIP messages.

The Proxy server can also manage the RTP/RTCP ports in audio/visual communication. In order to be able to realize this model, we implemented our application with an object-oriented approach using the Java language. We also used the

framework javax.sound.sampled developed by Sun Microsystems which includes classes such as (TargetDataLine) which facilitates the implementation of the recovery part of Stream (voice data flow) on the MIC.

VI. Implementation and Testing of the Application

In this section we describe in general the client-server model in Java which forms the basis of our SIP structure. We also present the different java classes that make up the different programs of our application.

A. The Different APIs and Framework of the JAVA Language Used

1. MJSIP

MjSip includes all the classes and methods for creating a SIP application. It implements the full layered stack, architecture as defined in RFC 3261, and is fully compliant with the standard. In addition, it includes top-level interfaces for call control [8]. It is made up of several packages including:

- standard SIP objects such as SIP messages, transactions, dialogs, etc.,
- various SIP extensions already defined in the IETF,
- call control API,
- a reference implementation of certain SIP systems (servers and UA) [8].

Some Reasons to Use MjSip

There are several implementations available of SIP, in the programming languages Java and C ++; MjSip is just another.

The main features of MjSip are:

- It is based on Java, so it is cross-platform,
- It is not just an API, but includes the full implementation of the SIP stack,
- It is very powerful and complies with the IETF RFC 3261 standard and extensions,
- It is simple to use and very simple to extend,
- It is very light and can be used simultaneously for server and light terminal implementations [8].

MjSip Architecture

According to the SIP architecture defined in RFC 3261, the main MjSip is structured in three basic layers: Transport, Transaction and dialog. In addition to these layers, MjSip also provides levels of call control and application level, with the corresponding API [8].

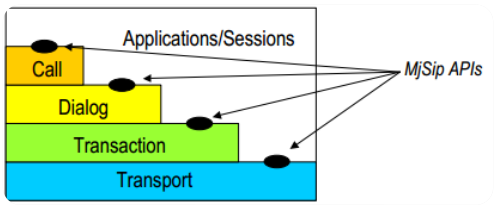

Fig. 7 The base layers of MjSip, Layers offer an easy-to-use interface to manage incoming and future SIP calls, dialogs, transactions and transport

The lowest layer is the Transportlayer which provides the transport of SIP messages. It is responsible for sending and receiving SIP messages via different lower layer transport protocols and demultiplexing incoming messages to the appropriate upper layer.

The second layer is the Transactionlayer. Transactions are a fundamental component of SIP. In SIP, a transaction is a request sent by a client (a transaction client) to a transaction server with all responses to this request sent from the transaction server returns to the client. This layer manages the retransmissions of upper layers, the correspondence of responses to requests and deadlines. It sends and receives messages via the transport layer.

The third layer (above the transaction layer) is the Dialog which links different transactions within the same "session". A dialogue is a peer-to-peer SIP relationship between two user agents that persists for some time. The dialog facilitates the sequencing of messages and the smooth running of requests between user agents.

INVITE () is the method that establishes a dialog (called an invitation dialog) and that is implemented in MjSip by the InviteDialog class.

The upper SIP layer is the call controller which implements a full SIP call. The Call Control layer is implemented by CallAPI, which provides an easy-to-use interface for managing incoming and future SIP calls. In addition, a call may have more than one dialog [8].

On these four layers, there are the SIP sessions which connect two or more application entities (participants) to different systems [8].

2. JAVAX.SOUND

It is a low-level API for manipulating audio input/output streams. It allows the capture, the reading of the audio streams and the manipulation of these (mixing effect etc.), unlike the JMF API, it allows total control over it. It is made up of two Packages related to the areas of java sound:

- javax.dound.Sampled which as its sound indicates is dedicated to the processing of sampled sounds from analog sources.
- javax.dound.midi which is dedicated to everything related to sound synthesis.

Fig. 8 shows a part of the code which initializes the parameters of sampling [11].

3. jna Bookstore

For the OPUS codec we made use of an interface between its native implementation and the java language via the jna library which is supported by the open source community on github. In the source code the method that performs the encoding is opus_encode () [9].

Fig. 9 is an extract from the code performing the imports.

```
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);
      }
```

Fig. 8 Code extract for initializing sound sampling parameters. This code describes the initialization process of the JAVAX.SOUND API sampling parameters

```
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;
```

Fig. 9 Importing elements from the jna library for interfacing the Java language and the OPUS codec. These are imports of the various modules via the JNA API for interaction with the OPUS codec

4. Netty

Netty is a java framework for developing asynchronous, maintainable and efficient network applications. It contains a well-designed API that deploys an abstraction layer on top of java. It is simple to use and neatly splits the code into logical pieces called "Handler", which makes it possible to develop network applications that are simple to maintain. This abstraction does not affect the performance of the application because the configuration possibilities are very important [12]. Netty also has new optimized and easy to use buffers. It offers many useful features on a daily basis for developers:

- Management of several transport layer protocols, such as UDP and TCP.
- HTTP management: construction of headers, transformation of messages into java objects.
- Websockets management.
- SSL management
- Compression management.

So in our application, we have used it up to now for the management of transport protocols.

5. Webcam Capture API

It is a library that allows using the integrated or external webcam directly from Java. It is designed to abstract commonly used camera functions and support various capture features [13].

6. Xuggler

Xuggler is a Java library that allows decoding and encoding a variety of multimedia file formats directly from Java [10]. It is built on the FFMPEG, but is designed with the following objectives:

- Ease of use;
- Security;
- Portability.

Unlike most Java libraries, Xuggle has native code (for example, a library shared by Windows DLL or Linux) component that must be installed with it [10].

7. Sample of the Source Code Implementation of Certain Entities

It should be remembered at this level that we carried out fairly conclusive tests during the implementation of the application. However, for the signaling module, we have chosen to use the UDP transport protocol and port 5060 which is the default port used by the SIP protocol, although it was possible to choose any other public port.

Here are some extracts of source code for some entities of the global architecture of the application architecture.

o Audio Transmission

The run () Method contained in ServerChannelHandler allows writing audio data on the network.

```
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();
          }
       }
```

Fig. 10 Extract of the code of the run () method for writing data on the network. In this method we present the writing on the network of real time data (audio) encoded by the OPUS codec

o Video Transmission

The encode () method is contained in the H264Encoder class which allows compression of the data retrieved via the webcam.

o Invitation

Here is the call method contained in the UserAgent class which is used to launch the call with the invite method of the sip specification.

```
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;
          }
       }
```

Fig. 11 Code extract of the compression of the data retrieved via the webcam. The function implemented here presents the process of compressing image data coming from the camera

```
        /** 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;
   }
```

Fig. 12 Code extract allowing to initiate an invitation in a call of the SIP specification. The call method implements the process of initiating an Apple via the two parameters including the address of the called party and the description of the media through which the data will pass.

- SIP Call Termination

A client who wants to end their connection sends a BYE request to the proxy server. The proxy routes this request to all participants, the transmission and reception of audio and video packets are stopped and the proxy registers the client, updates its client list and frees all multimedia transmission and reception ports. We present on Fig. 13 the hangup method contained in the UserAgent class which allows to end the call with the Bye method of the SIP specification.

```
     /** 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;
   }
```

Fig. 13 Code extract for the termination of a SIP call. The hangup () method implements the media closure and release procedure during a SIP session

VII. Software Complexity Assessment (Metrics and McCabe Cyclomatic Number)

To quantify the complexity of software, the most used measures are lines of code (LOC) since they are simple, easy to understand and count [3]. A simple test using Eclipse Metric allowed us to evaluate the cyclomatic complexity of the code by calculating in particular the cyclomatic number of McCabe. The metrics calculation was done by the eclipse Metrics plugin which we integrated into the eclipse IDE. Too high cyclomatic complexity (greater than 30) indicates that the method must be refactored. A cyclomatic complexity of less than 30 may be acceptable if the method is sufficiently tested. Cyclomatic complexity is linked to the notion of "code coverage", that is to say the coverage of the code by the tests. Ideally, a method should have a number of unit tests equal to its cyclomatic complexity to have a 100% "code coverage" [3].

B. Eclipse Metric

Eclipse Metrics is an eclipse plugin which analyzes the source code to calculate a certain number of metrics. It is also capable of presenting a graph of dependencies between packages in 3D [14], [15].

Eclipse Metrics is installed from the eclipse update manager (URL: http://metrics.sourceforge.net/update). After installation, it must be activated on each project to be measured using the project properties configuration panel (Metrics tab). Then, just display the Metrics view (Window/ Show View/Metrics View) to get the statistics of the selected element in the Eclipse explorer. Fig. 14 shows the result of running Metrics on the code "AudioOutputStream.java", presented in the section "Cyclomatic Complexity".

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...	

Fig. 14 Capture the result of the metrics calculation of the complexity parameters of the "AudioOutputStream.java" source code. These are statistics in terms of metrics from class AudioOutputStream

C. The Number of LOC for the Overall Project

Fig. 14 shows the code line number metrics for our project (SageVp) in general including also the codes coming from the MGSIP API. In total 27070 LOC in which we subtract 11395 LOC coming from the API MGSIP imported from the zoolu project from where 15675 LOC produced by our team. So to assess the complexity compared to this metric we are interested in the number of LOC per .java source file, given that a file must not exceed 400 LOC therefore must contain at most 10 methods, because each method must have at most 40 LOC.

Code line of the files in the "com.ul.sagevp.local.ua" package, which is the only package in which there is a file (UserAgent.java) which contains more than 400 LOC, ie 539.

Since we have 39 packages and 10 classes on average per package and only one class out of a total of 314 has 139 LOC overflowed from the maximum value that a file must have a maximum of 400 lines. In general, maintenance and testing of the application is not a problem, and the fault rate is negligible.

Fig. 15 Dependency graph centered on the com.ul.sagevp.local.codec.audio package. We present here the degree of dependence between the module of the application on the basis of inheritance between classes

Finally, Fig. 15 is an example of a 3D dependency graph. Metrics colors the cycles. It is possible to see only the dependencies of a particular package by double-clicking on that package, as shown in Fig. 13 in which we have chosen to center the view on the com.ul.sagevp.local.codec package. audio: Eclipse Metrics is a really useful tool for bringing out the architectural flaws of a project.

VIII. Conclusion

The objective of this work is part of the appropriation of new technologies in Africa by promoting the autonomous design of local tools capable of providing reliable IT solutions to companies in the DR Congo. We set a methodology that led to the design of our tool and this methodology was articulated first on the analysis and comparison of the standards of multimedia communication (H.323 and SIP) and our choice was based on the SIP protocol. We tackled the vulnerabilities of VoIP and identified good security practices. We proceeded to design the application architecture according to the UML formalism followed by coding in Java language.

References

[1] G. K., S. G. G Karopoulos, « Un cadre pour la confidentialité de l'identité dans SIP, » Un cadre pour la confidentialité de l'identité dans SIP, 2010.
[2] F. V. P Roques, « UML en action : de l'analyse des besoins à la conception en Java, » Eyrolles, 2014.
[3] verifysoft, «McCabe Metrics,» Germany, 2016.
[4] J. R. H. S. C. J. P. S. H. a. E. S. G. Camarillo, « SIP : Session Initiation Protocol, » RFC3261, 2018.
[5] R. R. V. P. H Agrawal, « Méthode et appareil pour SIP / H. 323 interfonctionnement, » IETF RFC 3261, 2011.
[6] J. C. E. S. J Ott, « Extensions RTCP (protocole de contrôle RTP) pour les sessions de multidiffusion à source unique avec commentaires de monodiffusion, » Extensions RTCP (protocole de contrôle RTP) pour les sessions de multidiffusion à source unique avec commentaires de monodiffusion, 2010.
[7] H. Schulzrinne, « RTP : un protocole de transport pour les applications temps réel, » RFC3550, p. 36, Novembre 2018.
[8] mjsip, «MjSip stack 1.5.4 and reference applications with source files, » Parma, 2012.
[9] U. Yamashita, « Évaluation de la capacité des odeurs de code à prendre en charge les évaluations de maintenabilité des logiciels : enquête empirique et approche méthodologique, » Évaluation de la capacité des odeurs de code à prendre en charge les évaluations de maintenabilité des logiciels : enquête empirique et approche méthodologique, 2012.
[10] Xuggle, «To encode, decode, and generally juggle audio and video files in any way that you want. » 2010.
[11] A. RINIE, «javaSound_arnie/javasound/presentation.html,» (En ligne). Available: http://www.igm.univ-mlv.fr. (Accès le 5 Juillet 2017).
[12] R. Cre, «Netty/introduction.html,» (En ligne). Available: http://www-

igm.univ-mlv.fr. [Accès le 10 juillet 2017).
[13] swarm, «image-capture,» (En ligne]. Available: http://www.it-swarm.dev. (Accès le 15 juillet 2017).
[14] A. Inc, «projects/sfnet_metrics/howto/install,» (En ligne). Available: http://fr.osdn.net. (Accès le 15 juillet 2017).
[15] D. Revuz, «eclipse-metrics.html,» (En ligne). Available: http://www-igm.univ-mlv.fr. (Accès le 20 juillet 2017).
[16] J. Rsenberg et H.Schulzrinne, Protocole d'initialisation de session, 2012.
[17] Z.Simon et Jean-Loui, Session intiation Protocol, 2010.