Implementing Monitoring Systems in Mobile Applications – a Case Study

Octavian DOSPINESCU, Roxana-Marina STRAINU
Faculty of Economics and Business Administration, AL.I.Cuza University, Iasi
doctav@uaic.ro, roxana.strainu@gmail.com

During the last years, the evolution of mobile applications allowed the developers to become more and more creative. In this way, we can imagine many applications models with a real impact in the real life. In this article we are trying to present a case study for a monitoring system using the Android platform and the benefits of computer networks. We use the power of mobile sockets and mobile threads, integrating them in a complex architecture in order to obtain a real monitoring system. As an immediate application, we propose a baby monitoring systems so that the children could be remotely supervised by their parents. The case study is based on an Android mobile client-server architecture and also uses the capabilities offered by the phone's speaker and microphone. We intend to have a robust application and that's why we initially preferred Network Service Discovery and Android P2P, but these functionalities are implemented starting with Android 4.1. So, we emulated all these functionalities by using a model based on sockets and server sockets.

Keywords: *Mobile Monitoring Systems, Android Implementation, Mobile Sockets, Mobile Architectures*

1 Introduction

In this article we present a case study of a mobile monitoring application made on Android platform. Our goal is to implement an application which can be used by parents in order to supervise their children; this vision could be used to also improve the proposals about the intelligent tutoring systems described by [1] or to implement innovative systems described by [2]. We propose the following scenario: the baby is in a room and the parent is in other room; in both rooms they have smart-phones with our application. Without using the GSM provider, the parent can hear the activity of the baby. Eventually, if the baby cries, the system will warn the parent about the situation.

The application is designed to run in some conditions and rules:

- the application does not use the GSM capabilities and it can also run on a tablet;
- the minimum required platform is Android 2.0;
- the application uses only a Wi-Fi connection and doesn't need any external remote server;
- the application uses the speakers and the microphones of the involved smart-phones.

Initially, we intended to use Network Service Discovery [3] and Android P2P, but these functionalities are implemented starting with Android 4.1. For these reasons, we preferred to emulate these capabilities by using the socket's technology. As a conclusion, the only restrictions are the following: smart-phones with a Wi-Fi connection, speaker and microphone. As we know, these resources are already available on almost all the smart-phones, so the proposed application could be installed on a significant number of mobile devices. Although the new HTML5 standard could represent a solid enterprise mobile computing foundation and a feasible path towards the proliferation of the applications as described in [4], we consider that it is extremely important that an application could be used by a large number of clients with "normal" smart-phones. We are also conscious that, according to [5], Mobile Application Development is a popular topic these days, when tons of mobile apps are released and available tools and resources are expanding.

Nowadays, the wireless networks have many sensors and protocols used to capture and transmit data [6], but one of the oldest methods of network communication provided by Java is by using Transfer Control Protocol

DOI: 10.12948/issn14531305/18.4.2014.11

(TCP) sockets. For this, we use two classes: Socket and Server Socket, the latter being the one to trace the connections that are made. For the audio streaming, it is used the device's microphone which is operated by Android through the objects of type Audio Recorder.
The information received by the microphone is encoded (using Audio Format class) through predefined constants (ENCODING_PCM_16BIT) and it is allocated a broadcast channel (in our case CHANNEL_IN_MONO). The audio coding is done using PCM (Pulse Code Modulation). It converts audio signals (represented by sound waves) to a digital signal (strings of 0 and 1) with or without compression.
For the audio reception it is used the smartphone speaker. The speaker is handled in Android using AudioTrack class. The signal emitted by the speaker uses the same type of encryption like that is received by the microphone and transmitted over the network. This time, the communication channel is CHANNEL_OUT_DEFAULT and the speaker chosen to issue the sound is operated by Audio Manager class. The predefined constant VOICE_CALL will direct the sound to receiver, and STREAM_MUSIC will direct the information to the speaker.

2 Scenarios and architectures for data and sound transmission in the monitor system

The network audio transmission is made by data packets with the rules described in [7]. They can be created and transmitted using Datagram Packet and Datagram Socket which work together. From the sender's side, the Datagram Socket object transmits an object of type Datagram Packet (which contains data from a buffer) to the IP connected to that device. The IP is then passed as a parameter when creating objects of type AudioSender (the class that will manage the sound recording operation). The receiver's object of Datagram Socket type receives a Datagram Packet and the data from the packet is introduced in a buffer which is then played by the speaker with the Audio Receiver class. This emission-reception process is presented in Figure 1.

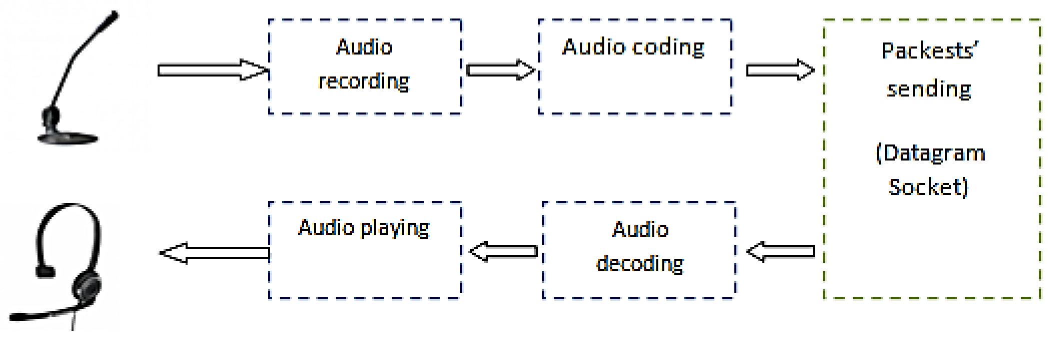

Fig. 1. The emission-reception process using Datagram Socket in a client-server architecture

The application uses threads and they are obtained by implementing the Runnable interface or by inheriting the Thread class which allows to override the run() method.

Scenario A:
In this scenario, the flow of execution is as follows:

- on both devices the application starts and awaits the user action;
- when the user selects a type of device, the graphical interface will change according to its type.

The complete description of this scenario is presented in Figure 2.

DOI: 10.12948/issn14531305/18.4.2014.11

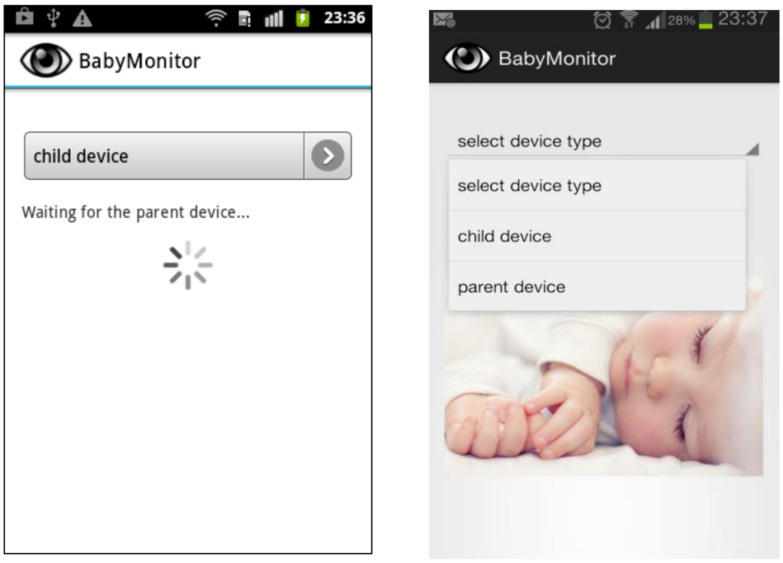

Fig. 2. Devices waiting for the user's action

The child device waits for parent device to connect. When the socket server object from the child device detects the connection, it runs the thread of the Audio Sender object which will switch on the microphone and it will start to deliver data packets to the child device. This will start the thread of AudioReceiver in order to receive packets from the server and to deliver them as a sound. The messages are sent by the server and the client to the GUI using a special handler.
The connected devices will show like in Figure 3 and in terms of hearing, the child device transmits the sounds to the parent device.

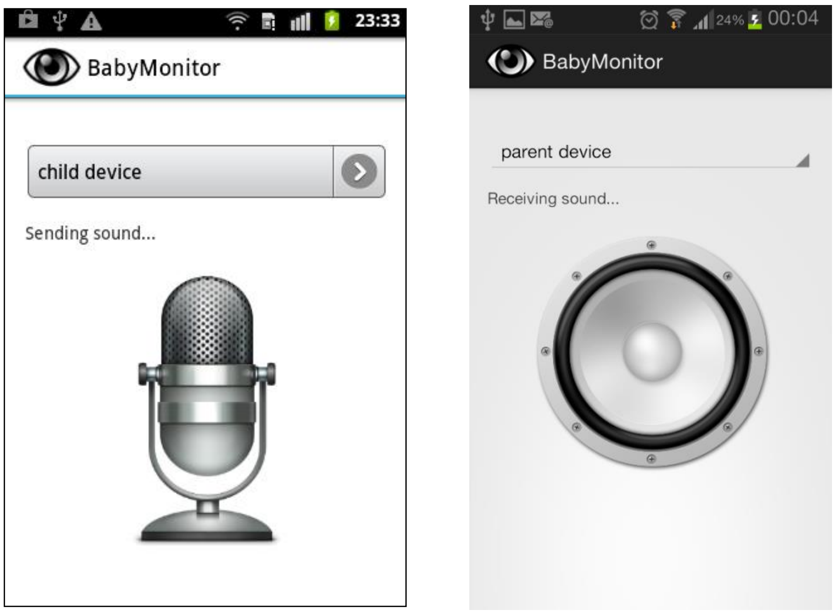

Fig. 3. Connected devices in scenario A

In order to better illustrate the operation of this scenario, we offer the detailed diagram presented in Figure 4.

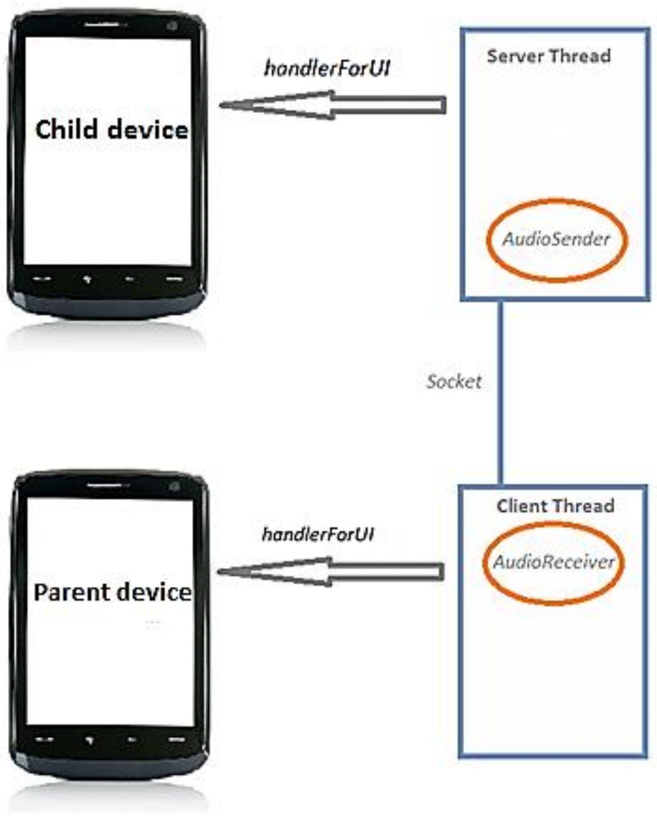

Fig. 4. The execution flow and the architecture of scenario A

Scenario B:
This scenario assumes that the parent device is able to emit sounds to the child device. This means that the parent device will have a button that will handle events of sound emission to and from the child device. As we can observe, this scenario is more complicated. Based on the scenario A described in Figure 4, it is not enough a communication only from client and GUI, but also in a reverse sense so that the client thread knows how to manage the transmission and reception (by pressing and releasing the button).
Also, the communication between client and server is no longer enough by using just sockets. The client must warn the server that it intends to transmit sound to it and the server should prepare for receiving it. In the same scenario, both client and server threads will have to manage how to communicate with the threads reserved for the sound. There will be not enough just to call the start() method of the sender and receiver's thread, but also their swap between them depending on user's action. The performances of the threading technique are very well described in [8].
Figure 5 reveals the way of working for this scenario and the internal structure of the application that is based only on threads.

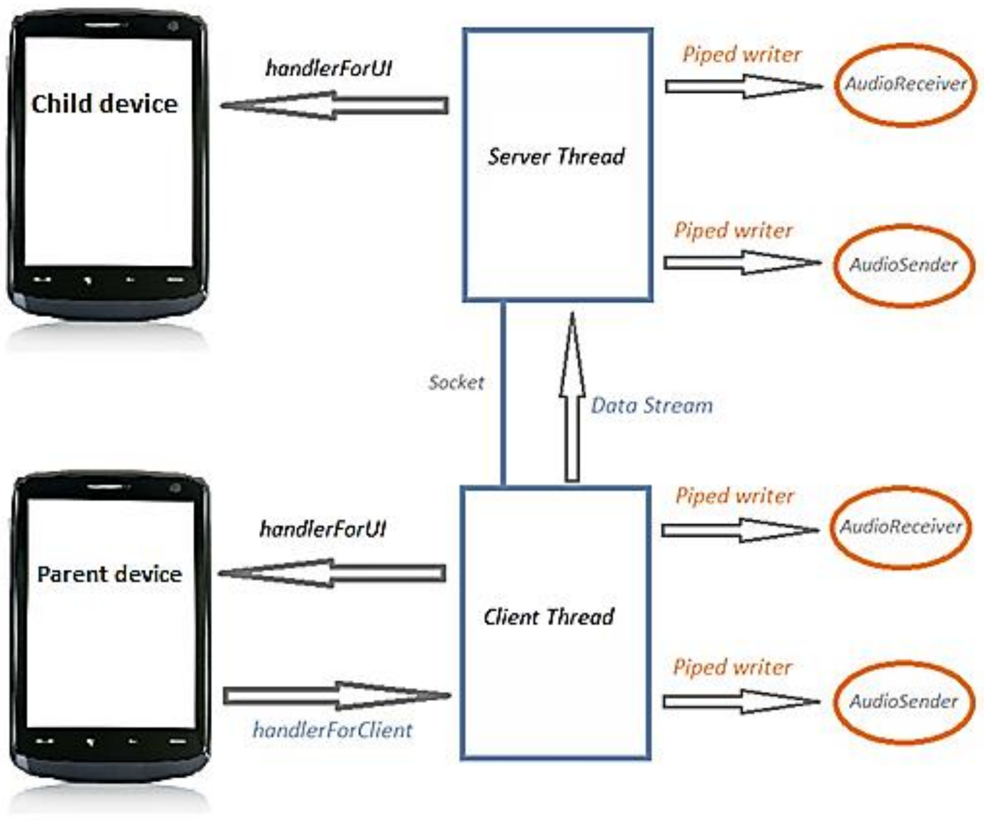

Fig. 5. The execution flow and the architecture of scenario B

The threads communicate each other using socket data streams and we had in mind the rules and principles presented in [9] and [10]. In this scenario, when the client receives the signal from the graphical interface by pressing "Talk to the baby" button to send sound to the server, it will communicate to the AudioSender thread, through a Piped Writer object, how to behave:

- "1" means that it should start recording and sending packets to the receiver;
- "0" means that it should release the microphone because the audio streaming is not necessary anymore.

Meanwhile, the server is alerted via a text message (using objects of type Data Stream) that is to receive a sound in order to know to stop broadcasting and start reception. The server in turn sends messages to Audio Receiver and Audio Sender to stop or start, then to the graphical interface to know to handle the corresponding event: it will replace the image of the microphone with the image of the speaker. So it will also behave the client thread announcing GUI about the sound transmission to another device, and it will change the speaker with the microphone.
The result of this process is presented in Figure 6.

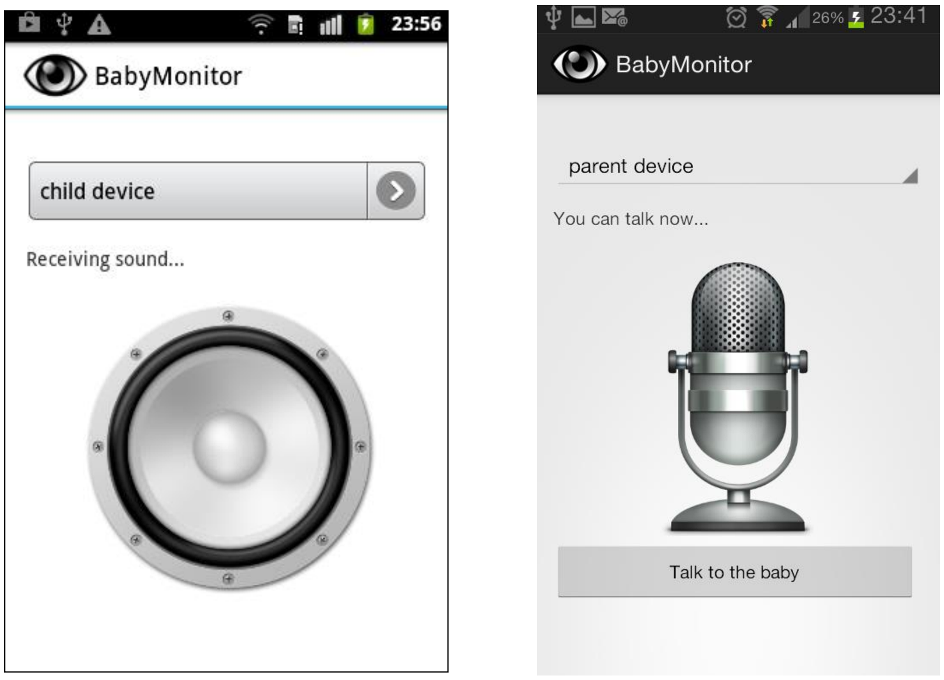

Fig. 6. The behavior of parent and child devices when they transmit sound in reverse

When the "Talk to the baby" button is released, the images on the screen and the corresponding text will change as shown in Figure 7.

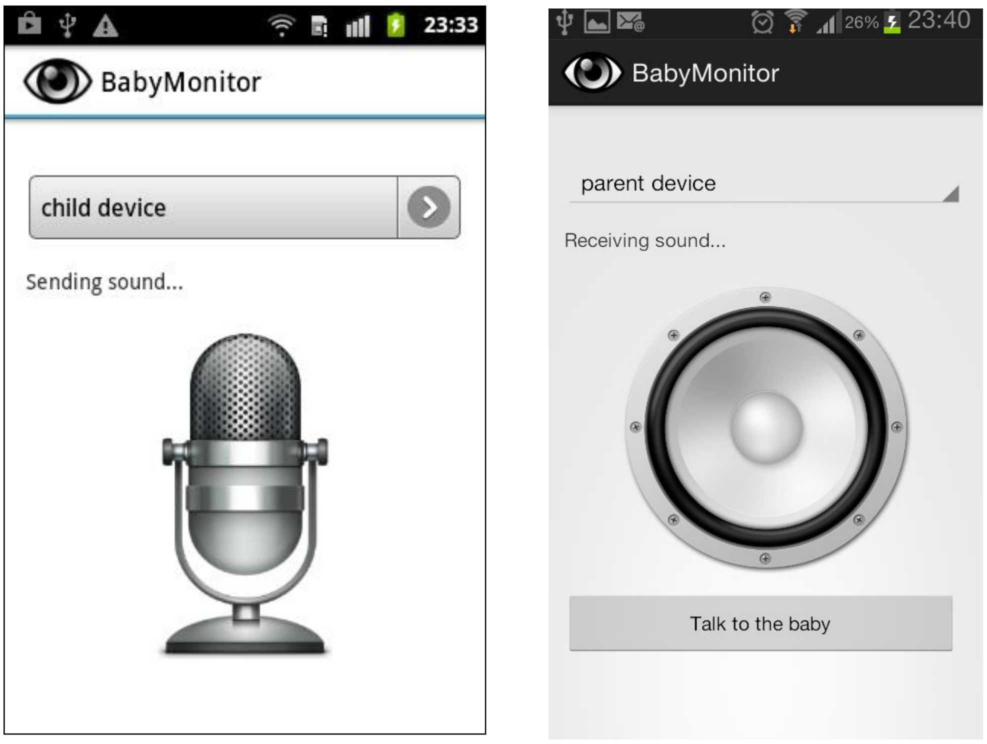

Fig. 7. Connected devices according to the scenario B

As we could see, the most interesting and realistic scenario is scenario B, which will be implemented in the following section.

3 The implementation of the monitoring system using Android platform

In order to implement the scenarios presented in the section above, we used many specific classes: AudioReceiver, AudioSender, ClientThread, ServerThread, CommunicationThread. All these classes are used then in MainActivity class. We consider that the most important and relevant classes are **AudioReceiver** and **AudioSender**.

AudioReceiver class is inherited from the Thread class. It receives sounds and the plays them for the user.

```
public class AudioReceiver extends Thread{
	public static DatagramSocket socket;
	private AudioTrack speaker;

	//Audio Configuration.
	private int sampleRate = 16000;      //How much will be ideal?
	private int channelConfig = AudioFormat.CHANNEL_OUT_DEFAULT;
	private int audioFormat = AudioFormat.ENCODING_PCM_16BIT;
	private boolean status = true;

	public AudioReceiver() {
		// TODO Auto-generated constructor stub
		Log.d("Audio Receiver","Creating receiver...");

	}

	public int getPort(){
		return socket.getLocalPort();
	}

	public InetAddress getIP(){
		return socket.getLocalAddress();
	}
	@Override
	public void run() {
		try {
			//socket-ul severului
			socket = new DatagramSocket(50005);
			Log.d("VR", "Socket was Created");
			Log.d("VR", "IP server: "+socket.getInetAddress());
			/*int minBufSize = AudioRecord.getMinBufferSize(sampleRate,
					channelConfig, audioFormat);*/
			//the same minim buffer size at sender and receiver
			int minBufSize = 4096;
			byte[] buffer = new byte[minBufSize];
			//the first parameter says where the sound will go.
			//AudioManager.VOICE_CALL is for the special speaker, and
			//AudioManager.STREAM_MUSIC is for the speaker
			speaker = new
AudioTrack(AudioManager.STREAM_MUSIC,sampleRate,channelConfig,audioFormat,minBufSiz
e,AudioTrack.MODE_STREAM);

			DatagramPacket packet;
			//speaker.play();
			Log.d("VR", "Playing the sound...");

			while(!isInterrupted()) {
				try {
					packet = new DatagramPacket(buffer,buffer.length);
					socket.receive(packet);
					Log.d("VR", "Receive packet of dimension:
"+buffer.length);
					//reading content from packet
					buffer=packet.getData();
					Log.d("VR", "Put the packet in buffer...");
					//speaker.setPlaybackRate(16000);
					//send data to speaker
					speaker.write(buffer, 0, minBufSize);
					Log.d("VR", "Writing data in speaker...");
					speaker.play();
				} catch(IOException e) {
```

DOI: 10.12948/issn14531305/18.4.2014.11

```
                              Log.e("VR","IOException");
                        }
                  }
            } catch (SocketException e) {
                  Log.e("VR", "SocketException");
                  //socket.close();
            }
      }

      public void stopReceiving(){
            speaker.release();
            //socket.close();
      }

      public String getIpAddress() {
            String ip = "";
            try {
                  Enumeration<NetworkInterface> enumNetworkInterfaces = 
NetworkInterface.getNetworkInterfaces();
                  while (enumNetworkInterfaces.hasMoreElements()) {
                        NetworkInterface networkInterface = 
enumNetworkInterfaces.nextElement();
                        Enumeration<InetAddress> enumInetAddress = 
networkInterface.getInetAddresses();
                        while (enumInetAddress.hasMoreElements()) {
                              InetAddress inetAddress = 
enumInetAddress.nextElement();
                              if (inetAddress.isSiteLocalAddress()) {
                                    ip += "SiteLocalAddress: "+ 
inetAddress.getHostAddress() + "\n";
                              }
                        }
                  }
            } catch (SocketException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  ip += "Something Wrong! " + e.toString() + "\n";
            }
            return ip;
       }
}
```

AudioSender class is inherited from the Thread class. In fact, it is a sound sender that reads data from the smart-phone's microphone. The most important implemented methods are run(), stopSending() and resumeSending(), as we can see in the code below.

```
public class AudioSender extends Thread{

      public AudioSender(InetAddress destIP) {
            // TODO Auto-generated constructor stub
            Log.d("Audio Sender", "Creating the sender...");
            destinationIP=destIP;
            buffer = new byte[4096];
            recorder = new 
AudioRecord(MediaRecorder.AudioSource.MIC,sampleRate,channelConfig,audioFormat,buff
er.length);
      }

      @Override
      public void run() {

            try {
                  socket = new DatagramSocket();
                  Log.d("VS", "It was created the socket for Audio Sender");
      //android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_URG
ENT_AUDIO);
                    //Log.d("VS","Buffer has " + minBufSize +" bytes!");
                    Log.d("VS", "The recipient is: "+destinationIP.toString());
                    //recorder.startRecording();
                    Log.d("VS", "Start recording...");
                    if(recorder.getState()==AudioRecord.STATE_INITIALIZED){
                          recorder.startRecording();
                          while(status){
                                String action=getMessage();
                                Log.d(TAG,"The action for microphone is:
"+action);
                                //resumeSending();
                                while(result.equals("start")){
                                      resumeSending();
                                }
                                while(result.equals("true")){
                                      recorder.release();
                                      Log.d(TAG, "Pausing microphone!...");

                                }
                          }
                    }else{
                          Log.d(TAG, "Warning!The microphone was not initialized");
                    }

              }catch (IOException e) {
                    Log.e("VS", "IOException");
              }
       }

       public void stopSending(){
              recorder.release();
              Log.d(TAG, " Pausing microphone!...");
       }

       public void resumeSending(){
         //reading data from MIC into buffer
         recorder.read(buffer, 0, buffer.length);
         Log.d("VS", "The buffer has "+buffer.length);
         //we put the buffer in a packet
         Log.d("VS", "Packet created...");
         DatagramPacket packet = new DatagramPacket
(buffer,buffer.length,destinationIP,port);
         Log.d("VS", "sending the packet...");
         try {
                    socket.send(packet);
              } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }
       }

}
```

4 Conclusions, advantages and future directions

As we noted in the previous sections, the application model that we propose has a number of real advantages:

- it can be used on smart-phones with Android 2.0;
- it uses limited resources and base classes (threads and classic graphical user interface);
- it uses features available on all phone (speaker and microphone);
- it does not require a remote dedicated server, because the transmission is made is a “peer-to-peer” way.

During all the tests we made, the application

worked fine and the quality of the transmission was excellent.
In the future, our model could be improved with some additional features:

- a warning system if the level of the sound exceeds a specified limit (for example, if the baby cries, then the parent should be warned in a specific way);
- a video system which could be activated in some specific conditions (for example, if the baby does not produce any sound for a while, then the video transmission should automatically activate on the parent side).

As a conclusion, we can say that the model we propose in this article is a realistic one and it has the potential to be extended with additional features.

References

[1] E. Pecheanu, D.Stefanescu, S.C. Buraga and A. Istrate, "Integratin Hypermedia Objects in an Intelligent Tutoring System", The Annals of "Dunarea de Jos" University of Galati, fascicle III, 2000, pp. 92-99. Available: http://www.ann.ugal.ro/eeai/archives/lf-01.pdf

[2] I. Smeureanu, N. Isaila, "New information technologies for an innovative education", World Journal for Educational Technology, vol. 3, issue 3 (2011), pp. 177-189. Available: http://www.world-education-center.org/index.php/wjet/article/view/252/pdf_68

[3] Android Developer, "Using Network Service Discovery". Technical documentation. Available: http://developer.android.com/training/connect-devices-wirelessly/nsd.html

[4] C. Strîmbei, "SOA Based Data Architecture for HTML5 Web Applications", Revista Informatica Economică, vol. 17, no.2/2013, pp. 84-95

[5] L.Hurbean and D. Fotache, "Mobile Technology: Binding Social and Cloud into a New Enterprise Applications Platform", Informatica Economica Review, vol. 17, no.2/2013, pp. 73-83

[6] R.S. Bal and A.K. Rath, "Clusterin Structure and Deployment of Node in Wireless Sensor Network", Journal of Information technology and computer science", 2014, 10, pp. 70-76. Available: http://www.mecs-press.org/ijitcs/ijitcs-v6-n10/IJITCS-V6-N10-10.pdf

[7] M. Mayilvaganan and S. Dhivya, "A Simple Packet Transmission Scheme For Wireless Data Over Routing Protocols – A Survey", International Journal of Research in Computer Applications and Robotics, august 2014, vol. 2, Issue 8, pp. 131-135. Available: http://www.ijrcar.com/Volume_2_Issue_8/v2i819.pdf

[8] A.Kika and S. Greca, "Multithreading Image Processing in Single-core and Multicore CPU using Java", International Journal of Advanced Computer Science and Applications, Vol. 4, No. 9, 2013. Available: http://thesai.org/Downloads/Volume4No9/Paper_26-Multithreading_Image_Processing_in_Single-core.pdf

[9] S.Sangar and M.L.Manickam, "Security and Privacy in Wireless Body Area Network", Indian Streams Research Journal, Vol. 4, Issue 8, 2014, pp. 1-7

[10] M.Georgescu and N.Suicimezov, "Issues Regarding Security Principles in Cloud Computing", USV Annals of Economics and Public Administration, Vol. 12, Issue 2(16), 2012, pp. 221-226

Octavian DOSPINESCU graduated the Faculty of Economics and Business Administration in 2000 and the Faculty of Informatics in 2001. He achieved the PhD in 2009 and he has published as author or co-author over 30 articles. He is author and co-author of 10 books and teaches as a lecturer in the Department of Information Systems of the Faculty of Economics and Business Administration, University Alexandru Ioan Cuza, Iasi. Since 2010 he has been a Microsoft Certified Professional, Dynamics Navision, Trade&Inventory Module. In 2014 he successfully completed the course "Programming Mobile Applications for Android Handheld Systems" authorized by Maryland University. He is interested in mobile devices software, computer programming and decision support systems.

Roxana-Marina STRAINU graduated in 2014 the Master of Business Information Systems at the Faculty of Economics and Business Administration, Alexandru Ioan Cuza University of Iasi. She also graduated the Faculty of Mathematics in the year 2005. She is interested in developing smart systems and mobile applications on Android platform. Now she is a PhD student in the business information systems area.

DOI: 10.12948/issn14531305/18.4.2014.11