Building Motion and Noise Detector Networks from Mobile Phones

Péter Ekler, Dr. Hassan Charaf

Budapest University of Technology and Economics
Magyar tudósok krt. 2, H-1111 Budapest, Hungary
peter.ekler@aut.bme.hu, hassan.charaf@aut.bme.hu

Abstract: As the capabilities of mobile phones are increasing, developers are able to create more complex applications on them. Nowadays, mobile phones support several networking technologies which allowes to create cooperative networks from them; this way mobile phones can work together in order to reach a common goal. We have created an application, called MobSensor, which basically turns the mobile phone into a motion and noise detector. MobSensor has also networking functions and it allowes to connect multiple mobile phones to each other. With the help of our solution we can create a cooperative sensor network from mobile devices running MobSensor. The paper discusses the architecture, the algorithms and the performance of MobSensor and it also examines its requirements.

Keywords: mobile phones, sensor network, motion detector, noise detector

1 Introduction

The hardware and software capabilities of mobile phones are increasing rapidly. They have high processing power and they can use relatively large amount of memory, thus they are able to run even complex applications with multiple threads. In this paper we would like to examine wheather a mobile phone is able to behave as a motion and noise detector and if it is possible then are we able to create a sensor network from mobile phones. To answer these questions first we have investigated which mobile platforms allow to implement such functions, after it we have implemented our solution, called MobSensor and we have examined its behavior in real environment.

Nowadays, we can realize the evolution of opearting systems for mobile phones. One of the oldest operating system for smartphones is the Symbian OS [1]. Symbian OS is an operating system aimed at wireless devices to provide enhanced usability and features on mobile devices. Symbian is also an independent company which started in 1998 partly owned by Ericsson, Nokia, Motorola and Psion. It was not until two years later the first symbian phone, the Ericsson R380 handset went on sale to the general public. Nowadays the Symbian OS is becomming more and more powerful, it supports all kind of hardwares and even touch screen. In 2001 the first GPRS Symbian Handset phone was released called the Nokia 7650.

Besides Symbian OS other mobile platforms are also developing like the Windows Mobile [2], where the goal is to create a platform for mobile phones which is easy to use and similar to the popular Windows operation system on personal computers. However if we look around in the market we can realize that new mobile platforms have also borned. One of the newest is the Android [3] which is developed by Google. Basically the Android platform is a software stack for mobile devices, which includes an operating system, a middleware and also key applications. The applications on the Android platform are written in the popular Java language. On Android platform developers have full access to the same framework APIs used by the core applications which is one of the key advantage of the platform. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities.

There are several platforms for mobile phones which makes the application developement harder, because it is always hard to decide which platform do we want to support. If the goal is to support different type of devices we can choose the Java ME platform [4] which enables to run the same application on different type of mobile phones with the help of a Java Virtual Machine (JVM) implementation. However developing applications on Java ME has some disadvandages like the capabilities of the JVM are limited and it does not allow to reach the low-level functions of the operating system. Another disadvantage is that the performance of these applications is lower because of the overhead of the JVM. Still the Java ME is the best solution if we want create an application, which is able to run on different type of mobile phones. In order to investigate wheather mobile phones are able to run a motion and noise detector application we have choosed the Java ME platform because we considered the platform independency as an important issue.

We can see that the software platforms of mobile phones are becomming more and more powerful and the hardware components of these devices are also developing well. The devices support different type of network connections like Bluetooth and even the WLAN technology. The mobile devices contains also several hardwares related to multimedia functions like camera, microphone, large display, stereo sound speakers, etc.

In order to implement a motion and noise detector we can use the built-in camera and microphone of the device and basically we have to implement an algorithm which compares the images and sound samples to detect differences. However it is not trivial to implement such algorithms on mobile phones, because of their limited resources compared to a personal computer.

The rest of the paper is organized as follows. Section 2 describes related work in the area of mobile phone programming and cooperative networks. Section 3 describes our application, called MobSensor, which basically turns the mobile phone into a network capable motion and noise detector. Section 4 highlights the requirements of MobSensor from the Java ME point of view. Section 5 proposes the architecture of our solution and it describes the motion and noise detecting algorithms. Finally Section 6 discusses about the performance and the user interface of MobSensor.

2 Related Work

Today wireless communication between mobile devices are increasingly becoming more and more important. Cognition, a continuous process involving sensing, reasoning, understanding and reacting, can be applied to wireless networks in order to adapt the system to the highly dynamic wireless ecosystem. The ultimate goals are to enhance the efficiency in the use of radio resources as well as to improve both link and network performance. Fitzek et al [5] present a detailed overview of a rapidly emerging topic in modern communications: cognitive wireless networks.

In [6] the authors collected examples about solutions for mobile phones based on cooperation. Each chapter contains examples and source code to rapidly make developers familiar with the most important concepts. Examples cover peer to peer networks, cooperative networking, cross layer protocol design, key challenges such as power consumption.

In a previous paper [7] we have examinded the cooperation between mainstream mobile phones by creating a BitTorrent client. In this peer-to-peer protocol the cooperatin between peers is extreamly important and the results were encouring; we managed to connect with mobile phones to the existing peer-to-peer community and participate as a full peer.

The key difference between this paper and the previous researches is that our goal is to create a sensor network from mobile phones by utilizing the special “sensors” of the device like camera and microphone.

3 MobSensor Functions

The main idea behind creating MobSensor application was that mobile phones are basically small computers with different types of multimedia capabilities and network handling technologies. MobSensor is able to sense its environment; it listens for changes via a motion and noise detector and it fires an alert if it detects anything.

The motion detector unit uses the camera of the phone and detects differences in the environment. If the difference measure calculated by the comparison algorithm is higher than a specific value, the application alerts. The noise detector works the same way; it uses the microphone of the mobile device and detects differences between sound samples. In both cases users are able to set the sensibility of the sensor algorithms to fit to the current environment.

MobSensor also has networking functionality; users are able to establish a network of mobile devices with ad-hoc WLAN technology or with the help of a WLAN router. In order to establish the sensor network in the current implementation of MobSensor, one device has to become a central 'boss' device that will behave as a server so that other phones can connect to it. The 'boss' device has a special ability: it can temporarily disable other sensors. This way, we can walk with the 'boss' device in the network without triggering any alerts. Figure 1 shows how mobile phones can establish an ad-hoc WLAN network with MobSensor.

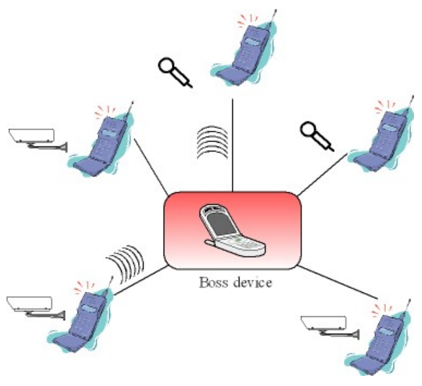


Figure 1
Ad-hoc WLAN network established by cooperating MobSensor applications

4 Requirements of MobSensor

The first step in implementing a motion and noise detector application on mobile phones is to choose the target platform. For this purpose, it is necessary to examine the capabilities of the perspective platform to decide wheather it is able to run such an application. Since platform independency is a key issue we have choosen the Java ME platform to support several different types of mobile phones. Following, we describe the platform requirements that are necessary for implementing a motion and noise detector and how Java ME can fulfil these requirements.

The application requires moderate processing power to compare images and sound samples in order to determine the difference between them. Besides that the applications requires also reasonabla amount of memory to store the image and sound samples for the comparison algorithm. Each time the application needs to store two images or sound samples in the memory in order to calculate a difference number between the two samples. If this number is higher than a specific one, then a motion or noise is detected and the application can alert.

From the Java ME point of view we have to clarify which Java ME versions and optional packages do we need. Java ME is based on three main elements: configurations, profiles and optional packages.

1 Configurations [8] describe the capabilities of the virtual machine and provide the basic set of libraries for a broad range of devices. The configuration targeting resource-constraint devices like mobile phones is called Connected Limited Device Configuration (CLDC). MobSensor requires the CLDC 1.1 configuration, because it allows applications to use more memory than the 1.0 version, which only allows 160-512 kB of memory. However this requirement does not mean a big limitation since CLDC 1.1 is supported by almost all type of mobile phones nowadays.

2 For defining a higher-level API the Java ME platform specifies profiles on top of the different configurations. The combination of Mobile Information Device Profile (MIDP) [9] with CLDC is widely used to provide a complete Java application environment for mobile phones and similar devices. MobSensor requires MIDP 2.0, since it supports different type of networking protocols like TCP/IP which we need to establish an ad-hoc sensor network from multiple mobile phones which are running MobSensor. MIDP 2.0 is also supported by almost all of the mobile phones which are not older than 3 or 4 years.

3 If we want to use other technology specific APIs in our application, we can import different kinds of optional packages which can be found in different JSRs [10] (Java Specification Request). MobSensor requires JSR 135 (Mobile Multimedia API) in order to reach the microphone and the camera of the phone.

To sum up from the MobSensor point of view, the Java environment on the device has to support CLDC 1.1, MIDP 2.0 and JSR 135. We can check on the manufacturers website wheather the selected phone supports these versions but nowadays they are common in mainstream mobile phones.

After we managed to choose the target platform and the necessary package versions two questions still remain. Will the mobile phones be able to constantly read the images from the camera and calculate a difference between two image in order to detect motions? Can these detectors work together in a network? Of course the same questions are valid in the case of noise detector. Following we describe our solution and we discuss about its performance.

5 MobSensor Engine

5.1 High Level Architecture of MobSensor

The application is built-up from four main units (Figure 2), each of which is responsible for a different functionality. The Motion and Noise Detector units implement the sensor functions, the User Interface unit is responsible for showing the most important information on the phone screen and the Network Manager implements the networking functions.

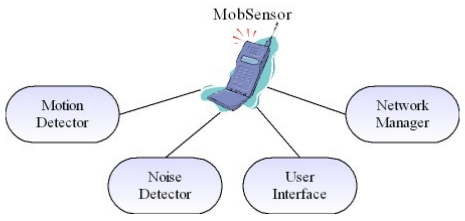


Figure 2
High level architecture of MobSensor

In this paper we do not discuss the networking function in more details since it does not contain any difficulties.

5.2 Motion and Noise Detecting Algorithm

The motion detector algorithm is based on continuous image recording and comparison. Figure 3 illustrates the algorithm on a flow diagram. We have not shown any exit points on the diagram, because user can stop the motion detector at any time.

The flow diagram of the noise detecting algoirthm is the same, thus we do not show it on a separate figure.

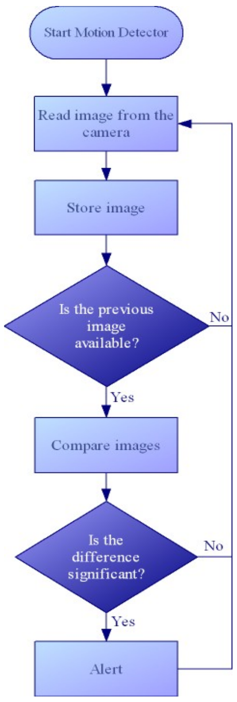

Start Motion Detector
Read image from the camera
Store image
Is the previous image available?
No
Yes
Compare images
Is the difference significant?
No
Yes
Alert

Figure 3
Flow diagram of the motion detector

MobSensor uses JSR 135 [11] for capturing images and recording sounds. The core of the motion detector is basically the following thread:

```
class MotionDetecting extends Thread {
  public void run() {
    byte[] imageBinary; // image from the camera
    Image image;
    int diffValue;

    // while motiondetecting is enabled
    while (motionDetecting_Enabled) {
      try {
        //--- reading the snapshot
        imageBinary = null;
        try {
          imageBinary = videoControl.
            getSnapshot("encoding=jpeg");
          // resize image if needed
        }
        catch (MediaException ex) {
          // handle exception
        }
        //--- comparing images
        if (imageBinary!=null) {
          // create image from the binary data
          image = Image.createImage(
            imageBinary, 0, imageBinary.length);
          // if there was no previous image
          if (previousImage == null) {
            previousImage = image;
          }
          //else compare images
          else {
            diffValue = pixelDifference_BetweenImages(
              previousImage,image,sensibility);
            if (!alert_disabled &&
              diffValue>diff_limit) {
              // make some noise
              alert();
              previousImage = null;
            }
            else {
              previousImage = image;
            }
          }
          imageBinary = null;
          System.gc();
        }
      } catch (Throwable t) {
        // handle exception
      }

      try {
        sleep(100);
      } catch (InterruptedException ex) {
        // handle exception
      }
    }
  }
}
```

The loop in the thread reads an image into the `imageBinary` byte array. After this, an `Image` object is created from the byte array, which will be compared to the previous image. If there is no previous image then we store this new `Image` object and start the whole loop again. Otherwise the comparison algorithm calculates a difference value (`diffValue`) from the current and the previous image with a specific sensitivity parameter (set by the user). If this value is higher than a specific number then the sensor alerts.

The algorithm that calculates the difference value from the images is implemented in the following two functions:

```
public static int pixelDifferencesBetweenImages(
  Image a, Image b, int aSensibility) {
  int[] aImagePixels = getPixels(a);
  int[] bImagePixels = getPixels(b);
  int pixelDifference = 0;

  for (int i=0; i<aImagePixels.length; i++) {
      if (getDifferenceFromARGB(
        aImagePixels[i],bImagePixels[i])>
        aSensibility) {
        pixelDifference ++;}
  }
  return pixelDifference;
}

public static int getDifferenceFromARGB(int firstPixel,
  int secoundPixel) {
    int red1 = (firstPixel >> 16) & 0xff;
    int green1 = (firstPixel >> 8) & 0xff;
    int blue1 = firstPixel & 0xff;

    int red2 = (secoundPixel >> 16) & 0xff;
    int green2 = (secoundPixel >> 8) & 0xff;
    int blue2 = secoundPixel & 0xff;

    int result = 0;

    result+=Math.abs(red1-red2);
    result+=Math.abs(green1-green2);
    result+=Math.abs(blue1-blue2);

    return result;
}
```

This current initial implementation is rather simple as it checks every pixel of the image and calculates a `pixelDifference` value based on the color of the pixels. Note that this comparison algorithm has a weak point: if we put the sensor in a dark room and we just turn on the light it will also alert. To avoid this false alert we have to extend the algorithm with an extra condition: if the color change for all pixels is rather large then the motion detector should not alert.

The noise detector works the same way as the motion detector (we will not discuss it in more details): it basically records half second long sound pieces and compares their power.

6 MobSensor Application

After the implementation of MobSensor we have tested it on several devices and the performance was reasonably good. Following we discuss about the performance of MobSensor, after we introduce the user interface of the implemented solution.

6.1 Performance of MobSensor

The required time for comparing an image to the previous one ($T_{performance}$) depends from three main components (1).

$$T_{performance} = T_{read} + T_{resize} + T_{compare} \quad (1)$$

T_{read} means how long does it take to read an image from the camera. T_{resize} is the required time for resizing the image if it is needed, because comparing large images can take too much time. $T_{compare}$ means how long does it take to determine the difference between the two image. In the algorithm we can set weather we want to resize the images or not. We can realize that if we do not resize the image then T_{resize} is 0, but the value of $T_{compare}$ will increase, because the resolution of the images is higher.

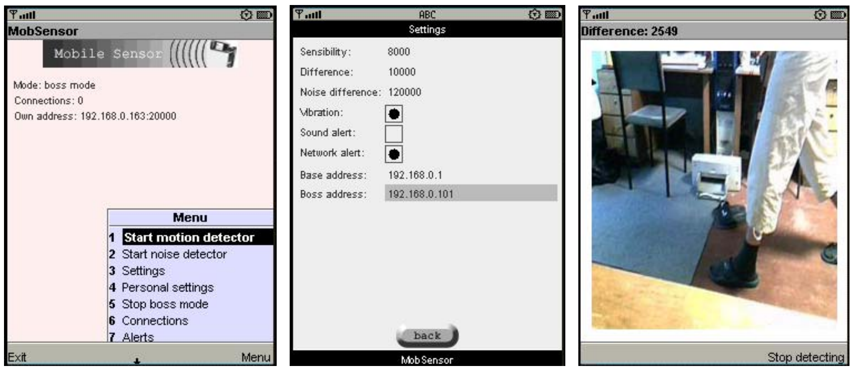
In general, the performance of the motion detector depends on the camera type and the JVM implementation of the device, because if the device has for example a five megapixel camera and the JVM does not allow to read images from the camera with low resolution than the image processing and image comparison can take much time. In order to decrease this processing time we can resize the images but it has also overhead (T_{resize}) as we have mentioned above. For example on Nokia N91 devices the process of comparing an image to the previous one without resizing the images took half secound, which is enough for detecting motions. On other devices the performance was also reasonable but sometimes the JVM implementation of the device does not allow to read low resolution images from the camera and resizing the images manually (T_{resize}) can take much time.

The performance of the noise detector is almost the same but it does not depend on the device, since the data readed from the microphone has always the same characteristics. If we want to apply the (1) equation to the noise detector, then the value of T_{resize} is always 0.

6.2 User Interface of MobSensor

If we start the application, on the main screen we can see the most important information about its state. We can reach the main functions from the menu like starting the motion and noise detector, viewing the settings and reaching the networking functions. Figure 4 illustrates the user interface of the application.

After we started the 'boss' mode from the menu, the main screen shows what the network address of the boss is and how many phones have already connected to our sensor network.



Mode: boss mode
Connections: 0
Own address: 192.168.0.163:20000

Sensibility:	8000
Difference:	10000
Noise difference:	120000
Vibration:	●
Sound alert:	☐
Network alert:	●
Base address:	192.168.0.1
Boss address:	192.168.0.101

Difference: 2549

Figure 4
User interface of MobSensor

On the settings view, we can set the sensibility of the sensors and the difference values above which the detectors should alert. The last screenshot represents the motion detector when somebody is moving in front of the device. On the top of the screen we can see the difference number which comes from the previously described comparison algorithm.

Conclusions

In this paper we were investigating the possibility of using mobile phones as motion and noise detectors and to create a sensor network from mobile devices running this application. Nowadays the capabilities of mobile phones are increasing rapidly, which allowes to implement more resource intensive applications on them. Besides that mobile phones support several networking technologies which allowes them to share their resources between each other in order to reach a common goal.

We have implemented an application, called MobSensor, to analyze the behavior of mobile phones as motion and noise detectors. MobSensor is able to sense its environment and detect any kind of motions and noises. The motion and noise detection is based on continous image and sound sample comparison. We have analyzed the performance of MobSensor ans our experiments shows that it is reasonalby good.

Future plan will be to increase the performance of the application and to allow to create short-range sensor networks from MobSensor capable phones via Bluetooth connections. We also plan to extend MobSensor functions by allowing to upload those images to a website which caused the alert. This way we can see from a web browser what caused the alert and these images can be used for further analysis as well.

References

[1] History of the Symbian OS, Sept. 22, 2008 [Online], Available: http://www.symbiano.com/history-symbian.php

[2] Windows Mobile, Sept. 22, 2008 [Online], Available: http://www.microsoft.com/windowsmobile/en-us/default.mspx

[3] Android platform for mobile phones, Sept. 22, 2008 [Online], Available: http://code.google.com/android/

[4] Java ME platform, Sept. 23, 2008 [Online], Available: http://java.sun.com/javame/index.jsp

[5] H. P. Fitzek, M. D. Katz, "Cognitive Wireless Networks", ISBN: 978-1-4020-5978-0. Springer, 2007

[6] H. P. Fitzek, F. Reichert, "Mobile Phone Programming", ISBN: 978-1-4020-5968-1, Springer, 2007

[7] P. Ekler, J. K. Nurminen, A. J. Kiss "Experiences of Implementing BitTorrent on Java ME platform", CCNC'08. 1st IEEE International Peer-to-Peer for Handheld Devices Workshop 2008, Las Vegas, USA, to be published

[8] Connected Limited Device Configuration, Sept. 23, 2008 [Online], http://java.sun.com/products/cldc

[9] Mobile Information Device Profile 2 description, Sept. 24, 2008 [Online], Available: http://developers.sun.com/mobility/midp/articles/midp2network

[10] Java Specification Request overview, Sept. 24, 2008 [Online], Available: http://www.jcp.org/en/jsr/overview

[11] Java Specification Request 135, Sept. 24, 2008 [Online], Available: http://jcp.org/en/jsr/detail?id=135