UTILIZATION OF MOBILE PHONE AS POINTING DEVICE

Michal Németh

Bachelor's Thesis
May 2011

Degree Programme in Information Technology
School of Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
DESCRIPTION JAMK UNIVERSITY OF APPLIED SCIENCES

Author(s) NÉMETH, Michal	Type of publication Bachelor´s Thesis	Date 16.05.2011
	Pages 49	Language English
	Confidential ( ) Until	Permission for web publication (X)
Title UTILIZATION OF MOBILE PHONE AS POINTING DEVICE		
Degree Programme Information Technology		
Tutor(s) MIESKOLAINEN, Matti		
Assigned by Ixonos Finland Ltd.		

Abstract

Android platform along with smartphones running it is positioned on the third place in the statistics of the most used mobile operating system with 16% worldwide. (April 2011) The high popularity among users due to user-friendly GUI and fast responsibility as well as easy application development have affected this achievement.

This project aimed at utilizing provided hardware sensors's features from smartphone devices together with various useful libraries from Android API. As a result, an application combining different pointing devices was created. The application includes a touchpad simulator, a simulator of a pointing stick, a detection of few gestures done by rolling and pitching of a phone plus controlling a mouse cursor with tilting of the phone. The connection with the controlled PC is established wirelessly via Wi-Fi. This project provides possibility to easily extend it to a version which could also simulate a keyboard. The outcome is a handy, easy-to-use application capable of substituting common pointing devices.

The implemented application consists of two parts. The first one is an application for Android platform smartphones and the second one is a server application executing commands selected by user's application. The server application is implemented in Java programming language.

Keywords

Android, sensor, mouse, gesture, touchpad, pointing stick, Java, Wi-Fi

Miscellaneous

CONTENTS

ACRONYMS....3

1 INTRODUCTION....4

2 THEORETICAL BASIS....5
2.1 Pointing devices....5
2.2 Smartphones with Android....7
2.3 Sensors....9
2.4 Eclipse and Java....11
2.5 Ixonos and Scrum....14

3 REQUIREMENTS....16

4 IMPLEMENTATION....17
4.1 Background....17
4.2 Server application....19
4.2.1 Description....19
4.2.2 Communication interface....20
4.2.3 Implementation and data flow....22
4.3 Client application....28
4.3.1 Description....28
4.3.2 Application usage and implementation....29
4.3.2.1 Database....30
4.3.2.2 Connection....32
4.3.2.3 Main Activity....32
4.3.2.4 Compass....33
4.3.2.5 Pointer....35
4.3.2.6 Gestures....36

4.3.2.7 Touchpad....38
4.3.2.8 Pointing stick....39
4.4 Testing....41

5 CONCLUSION....42

6 DISCUSSION....43

REFERENCES....46

FIGURES

FIGURE 1. Most used pointing devices....6
FIGURE 2. HTC Hero smartphone with Android OS....8
FIGURE 3. Logo of Eclipse project....12
FIGURE 4. Logo of Java....13
FIGURE 5. Logo of Ixonos company....14
FIGURE 6. Hardware architecture of the system....17
FIGURE 7. Interaction model of components of the system....18
FIGURE 8. Software architecture of the server application....20
FIGURE 9. List of PCs....29
FIGURE 10. Item's context menu....29
FIGURE 11. Data model....30
FIGURE 12. GUI of the client application....33
FIGURE 13. Orientation of axis in the Android device....34

ACRONYMS

API – Application Programming Interface

GUI – Graphical User Interface

HTML – Hypertext markup language

ICT – Information and Communication Technologies

IrDA – Infrared Data Association

IDE – Integrated Development Environment

Java SE, ME, EE – Java Standard Edition, Micro Edition, Enterprise Edition

JDK – Java Development Kit

NDA – Non-Disclosure Agreement

OHA – Open Handset Alliance

OS – Operating System

PC – Personal Computer

PHP – Hypertext Preprocessor

SDK – Software Development Kit

TCP / IP – Transmission Control Protocol / Internet Protocol

UML – Unified Modeling Language

Wi-Fi – Wireless Fidelity – Wireless network

XML – Extensible Markup Language

1 INTRODUCTION

These days computers and all the electronic gadgets are an inseparable part of our everyday life. Firstly mentioned personal computers are not any longer meant only for working purposes, but more and more used for entertainment in people's spare time. This is also applicable to the mobile phones, which have transformed into multifunctional devices with almost same features as computers have. That provides developers of applications a possibility to combine them together and thus create functionality, which was even few years ago completely unimaginable to once become true. In the authors opinion, this project can be considered as one of that type of features. Why? Because a person can remotely control his PC when he forgets a mouse at home, or she/he just wants to control it while lying in a cozy bed, or in other various situations. With this project the author of this thesis wanted to achieve creation of an application capable of controlling mouse cursor with movements of the mobile phone and additionally, some other functions.

The topic was chosen because like I mentioned before, mobile phones (at present mainly smartphones) have great potential for applications developers to do almost everything. In this particular application Android platform phone was used which is rising in popularity. That is obviously an opportunity for developers to spread their products to more users and also users have wider variety of different types of applications. Furthermore, modern smartphones have more than one sensor for detecting changes in location and position of the apparatus.

The project presented in this thesis is derived from a similar one, which the author was working on during 3 months internship period in Ixonos Finland Ltd. in Jyväskylä. The original one is company confidential and NDA does not allow it to be presented in public.

2 THEORETICAL BASIS

2.1 Pointing devices

Mouse

Undoubtedly, a **mouse** is the most commonly used pointing device allowing partly actuating a personal computer. A mouse or a computer mouse is an input device of a computer, used to control the cursor position on the screen and perform operations by pressing its buttons. The computer mouse is generally a small object that fits in users palm and is freely lying on a pad. At the bottom of the mouse is a device that detects two dimensional movements of the mouse relatively to the pad underneath and then it is transferred to a computer screen. The mouse got its name because the older models with a cable leading to the computer reminded of this rodent. (Mouse SK Wiki)

The following figure shows the most widespread devices. In the upper row joystick, optical mouse, trackball are illustrated; in the lower row: touchpad and pointing stick.

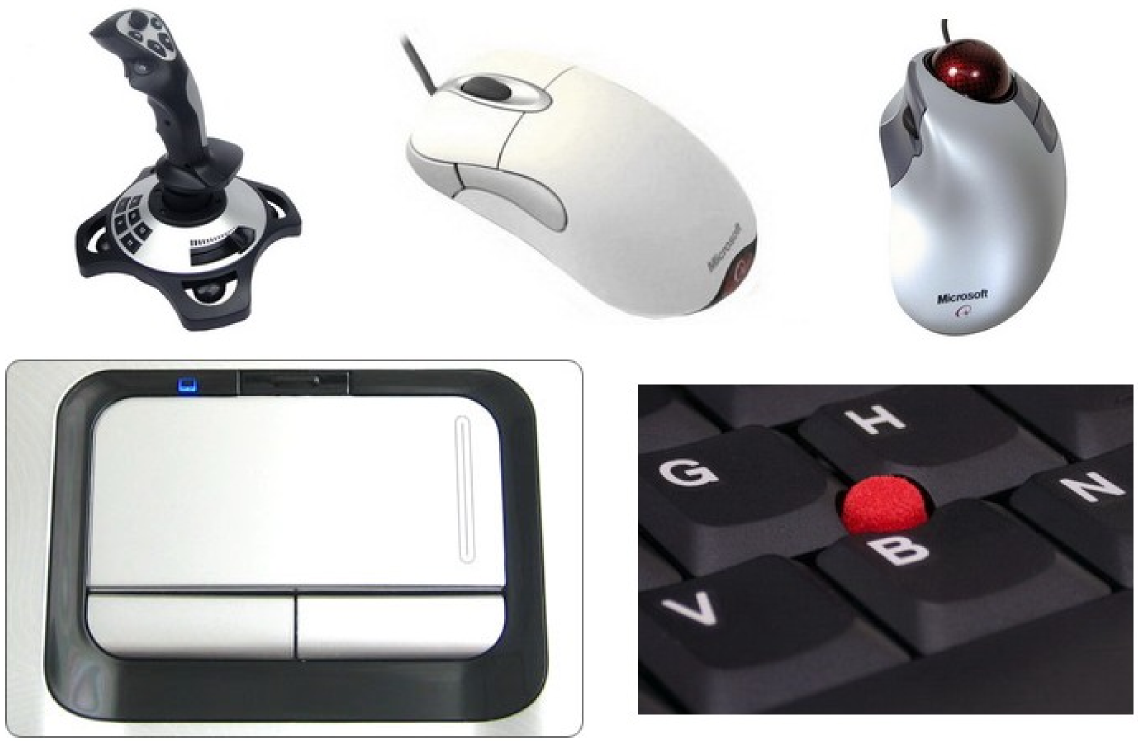


FIGURE 1. Most used pointing devices.

The cable is one way how to connect mouse with computer. Even though this concept is old-fashioned, most models still have a wired connection, because wireless requires batteries as a power supply. On the other hand, that independence presents more comfortable usage anywhere.

Touchpad

Another widely used way of controlling the cursor, mainly integrated in laptops, is **touchpad** (or trackpad). Touchpad is usually a small, rectangle shaped, resistive touchscreen. To control it, a user has to slightly press its surface with the finger and then slide it on the surface. Since it is almost always built-in, no external mouse is needed; therefore it reduces the amount of necessary equipment, but requires a "within reach" distance from laptop. (Touchpad SK Wiki)

Miscellaneous

Except these popular ones, manufacturers have come up with many different inventions and combinations, namely **trackball**, **joystick**, **pointing stick** (TrackPoint) and many others. There even exists a **gyroscopic mouse**, thus something really close to this project, but the user still needs extra equipment.

2.2 Smartphones with Android

Smartphone is a mobile phone that offers more advanced computing ability than older types of phones. According to this ability, it is capable of running its own, fully functional, mobile operating system.

One of those operating systems is **Android OS**. Android is a Linux based software platform developed by American company **Google Inc.** giving the entire platform with source code to the association of the companies (OHA) also a member. Google not only initiated the creation of Android platform, but also the emergence of OHA and financial rewards in the Android Developer Challenge competition, which emerged from the first applications for this platform. Android SDK allows developers to write applications in Java using libraries developed by Google.

FIGURE 2. HTC Hero smartphone with Android OS.

Android platform was announced on 5th November 2007 simultaneously with the foundation of OHA consortium, which currently has 34 hardware manufacturers, software and telecommunications companies involved in promoting open standards in the world of mobile devices. Since the beginning of 2008, when the very first publicly available version of this platform was released, its components are all available to anyone under the Apache license-free software and open-source license. Android from its launch has undergone a great change to the final **Android 2.3** version **Gingerbread** (12th June 2010). **Android 3.0** is designed only for tablets. (Android Operating System SK Wiki)

Below is a list of the latest Android versions for smarphones:

- Android 1.5 (Cupcake)
- Android 1.6 (Donut)
- Android 2.1 (Eclair)
- Android 2.2 (Froyo)
- Android 2.3 (Gingerbread).

(Android Operating System EN Wiki)

Few manufacturers supporting Android OS are listed below as follows:

- Acer Inc.
- Cherry Mobile
- HTC
- LG Group
- Motorola
- Samsung
- Sony Ericsson.

(List of Android devices EN Wiki)

2.3 Sensors

"In general, a sensor is a device that measures some physical quantity and converts it into the form readable for an observing person or other device / instrument." (Sensor EN Wiki) Smartphones currently dispose with a couple of different sensors for measuring various values. In terms of Android platform phones, developers have possibility to utilize data from following sensors:

- Accelerometer
- Compass
- Gravity sensor
- Gyroscope
- Light sensor
- Linear acceleration sensor

- Magnetic field sensor
- Rotation vector sensor.

However, not every device has all of them. Some of listed receptors are not relevant to the presented thesis; therefore the author does not introduce them onward in the more detailed scope.

Accelerometer

"An accelerometer is a device that measures the proper ***acceleration*** *of the device. This is not necessarily the same as the coordinate acceleration (change of velocity of the device in space), but is rather the type of acceleration associated with the phenomenon of weight experienced by a test mass that resides in the frame of reference of the accelerometer device."* (Accelerometer EN Wiki) "*Sensor's values are in* ***meters/second^2*** *units. A sensor measures the acceleration applied to the device. For this reason, when the device is sitting on a table (and obviously not accelerating), the accelerometer reads a magnitude of* ***g*** *= 9.81 m/s^2. Similarly, when the device is in free-fall and therefore dangerously accelerating towards to ground at 9.81 m/s^2, its accelerometer reads a magnitude of 0 m/s^2."* (Android Developers – sensors)

Compass

"A compass is a navigational instrument for determining ***direction*** *relative to the Earth's* ***magnetic poles****. It consists of a magnetized pointer (usually marked on the North end) free to align itself with* ***Earth's magnetic field****."* (Compass EN Wiki)

In Android's terminology it is called **Orientation sensor**. More details about a Compass sensor are elucidated in the chapter 4.3.2.4 .

Gyroscope

"A gyroscope is an instrument consisting of a rapidly ***spinning wheel*** *so mounted as to use the tendency of such a wheel to maintain a fixed position in space, and to resist any force which tries to change it. The way it will move if a twisting force is applied depends on the extent and orientation of the force and the way the gyroscope is mounted. A free vertically spinning gyroscope remains vertical as the carrying vehicle tilts, so providing an artificial horizon. A horizontal gyroscope will maintain a certain bearing, and therefore indicate a vessel's heading as it turns. Modern gyroscopes (including those built-in in smartphones) no longer have a spinning wheel."* (Gyroscope Cambridge Encyclopedia)

"All values are in ***radians/second*** *and measure the rate of rotation around the* ***X*****,** ***Y*** *and* ***Z*** *axis. The coordinate system is the same as is used for the acceleration sensor."* (Android Developers – sensors) Rotation is positive in the **counter-clockwise** direction.

2.4 Eclipse and Java

Eclipse

Eclipse is an **open-source development platform** that is known to most people as an **IDE for Java** programming. Flexible design allows the platform to extend the list of supported programming languages with the help of **plug-ins**, such as C++, Python or PHP. It allows plug-ins that extend the development environment such as the UML design, and writing HTML or XML.

FIGURE 3. Logo of
Eclipse project.

Unlike other Java development environments such as NetBeans, Eclipse has a philosophy of being closely tied to the scalability of using plug-ins. In the basic version the Eclipse includes only integrated development tools for Java such as a standard compiler, debugger, etc., but does not include a tool for visual design of GUI, desktop applications or application server - all such extensions are needed to be delivered by the form of plug-ins. For this reason is Eclipse currently the most popular IDE for Java. (Eclipse SK Wiki)

Below is a list of Eclipse versions as follows:

- Callisto
- Europa
- Ganymede
- Galileo
- Helios.

(Eclipse EN Wiki)

Java

Java is an **object-oriented programming language** developed by **Sun Microsystems** and launched on 23rd May 1995. It is one of the most widely used programming languages in the world. Thanks to its portability it is being used for programs that are working on various systems from smart cards (JavaCard platform), via mobile phones and various embedded devices (**Java ME** platform), applications for desktop computers (**Java SE**) to large distributed operating systems cooperating on a number of computers spreading around the world (**Java EE**). These technologies as a whole are called the Java platform. On 8th May 2007 Sun released the Java source code (about 2.5 million lines of code) and Java will be further developed as open source. (Java CS Wiki)

FIGURE 4. Logo of Java.

There are several theories about the origin of the name of this language, one of them speaks of the inspiration slang for coffee.

2.5 Ixonos and Scrum

Ixonos

“Ixonos is an ***ICT services company*** *creating innovative solutions for* ***mobility****,* ***social media*** *and* ***digital services****. Together with customers develops products and services which let people enjoy inspiring digital experience. Ixonos’s clientele comprises leading mobile and smartphone manufacturers operating on global markets, mobile network suppliers and teleoperators as well as leading Finnish finance companies and public administration organizations. Ixonos has its headquarters in* ***Helsinki****, Finland, and other local offices in Tampere, Turku, Salo, Jyväskylä, Oulu and Kemi. Besides that, Ixonos has subsidiaries in Košice, the Slovak Republic, and in Germany, Ixonos Testhouse has office in Tallinn, Estonia and Ixonos Beijing in China.”* (Ixonos's internal brochure)

FIGURE 5. Logo of Ixonos company.

Projects in the company are developed with the use of Scrum methodology and this particular one was no exception.

Scrum

Scrum is one of many **agile software development** methodologies. It is a process skeleton that contains sets of practices and roles.

The main roles are:

- **Scrum Master** – not team leader, but ensures that development runs as smooth as possible and searches for needed resources
- **Product Owner** – represents customer's needs
- **Team** – a group of people responsible for the whole development.

The process is divided into stages called "**Sprints**", from which every one lasts few weeks (from two up to four) and each has a shippable version at the end of a sprint. Each sprint begins with Spring planning meeting, where **Sprint backlog** is prepared by selecting tasks from Product backlog. After that, every day is held a **Daily Scrum**. At Daily Scrum the members review, what they have done the day before, what are they planning for upcoming day and if they are facing any serious obstacles. The meeting takes place every day at the same place and the same time. The maximal duration should not exceed 15 minutes. At the end of every sprint a functional version is presented along with **Sprint Review Meeting**. (Scrum EN Wiki)

3 REQUIREMENTS

The supervisor set the following requirements for the project:

The project has to consist of two parts:

- Client mobile application
- Server application executing client's commands.

Requirements for the client are listed below as follows:

- Implementation for Android platform
- Connection via TCP/IP protocol
- Usage of Wi-Fi
- Possibility to generate keyboard presses on the remote PC
- Possibility to control mouse cursor on the remote PC with changes in orientation of the mobile device
- Possibility to detect easy to use gestures and assign them some special function executed in the remote PC
- Simulate touchpad with usage of the screen of the mobile phone to control the cursor on the remote PC
- Simulate pointing stick with usage of the screen of the mobile phone to control the cursor on the remote PC

Requirements for the server are given below:

- Implementation in the Java
- Usage of java.awt.Robot to simulate keyboard and mouse events
- Usage of sockets for receiving orders from client

4 IMPLEMENTATION

4.1 Background

As previously stated, this project is not a copy of the original one, which the author was working on during his internship, just a derivation of it. For implementation **Eclipse Galileo** was used – IDE for Java developers including **Java SE Development Kit 6 Update 24** with **Android SDK Tools revision 8**. The application was tested in early phases on a virtual device with Android 1.5, later on real **HTC Hero with Android version 1.5**. The phone was connected with a server machine via **Wi-Fi router**. The server application is written in Java language in the same version of Eclipse and JDK. For transmission commands from client to cursor movements **Robot** class from **java.awt** library was used.

From all these, it can be seen that the project consists of two main parts:

- **Client application** in a mobile phone
- **Server application**.

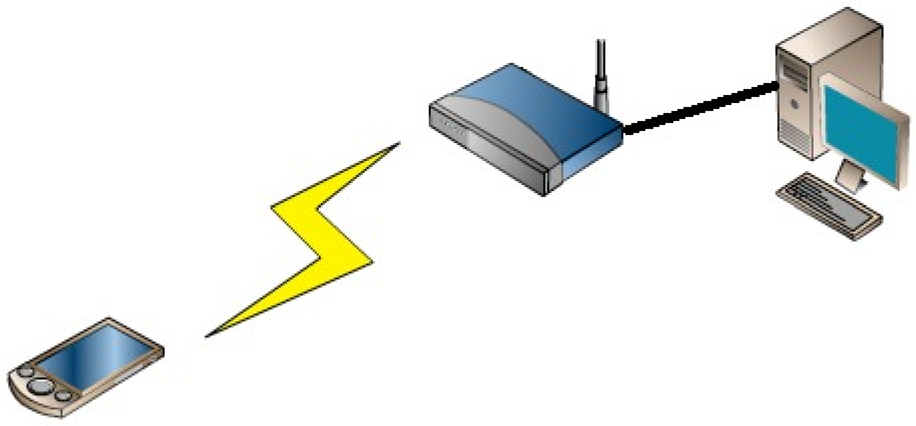


FIGURE 6. Hardware architecture of the system.

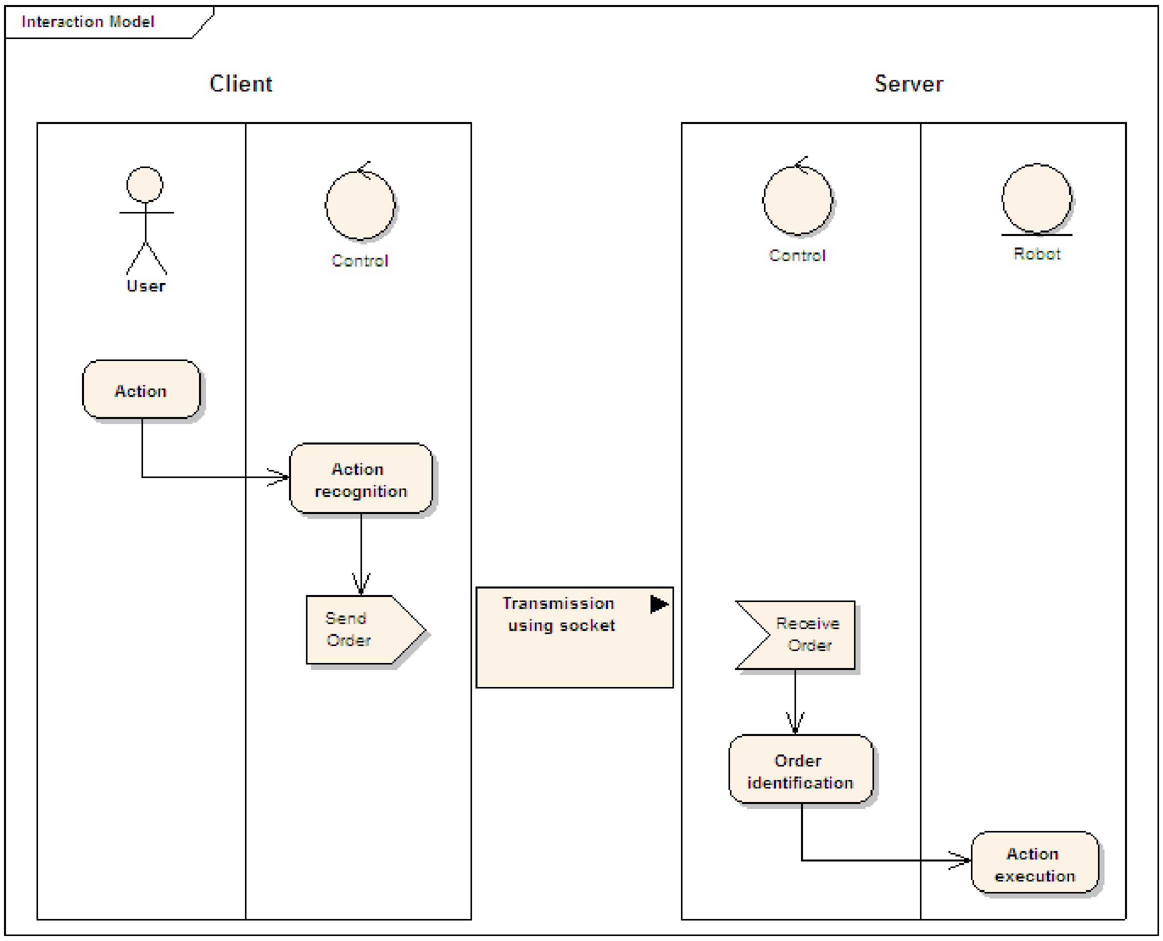


FIGURE 7. Interaction model of components of the system.

The more detailed software architectures of both parts are described in the following chapters.

4.2 Server application

4.2.1 Description

Basically, the main function of the server is to execute commands entered by the user with use of his client application. More specifically, if a user presses some type of button on his user interface layout, for instance left mouse button click, the server has to generate a system call, as if a real left mouse button was clicked. Java contains dozens of different types of libraries and **java.awt** was chosen that includes the **Robot** class. This class is capable of generating fake button presses, releases and mouse cursor movements and many other various functions, such as capturing a screen, getting color of selected pixel, and further. The server does not know, what evoked sending the command to the server, if it was simple button on layout or special gesture. It can recognize only 2 types of event, either **keyboard** or **pointer**.

The server receives orders from client via socket connection, which is in this particular case connected to wireless router attached to the server's PC. For case in point see figure 6.

The following figure displays a software structure of the server consisting of two packages: **data** and **server**. Section 4.2.2 contains closer look at the internal structure and its functionality.

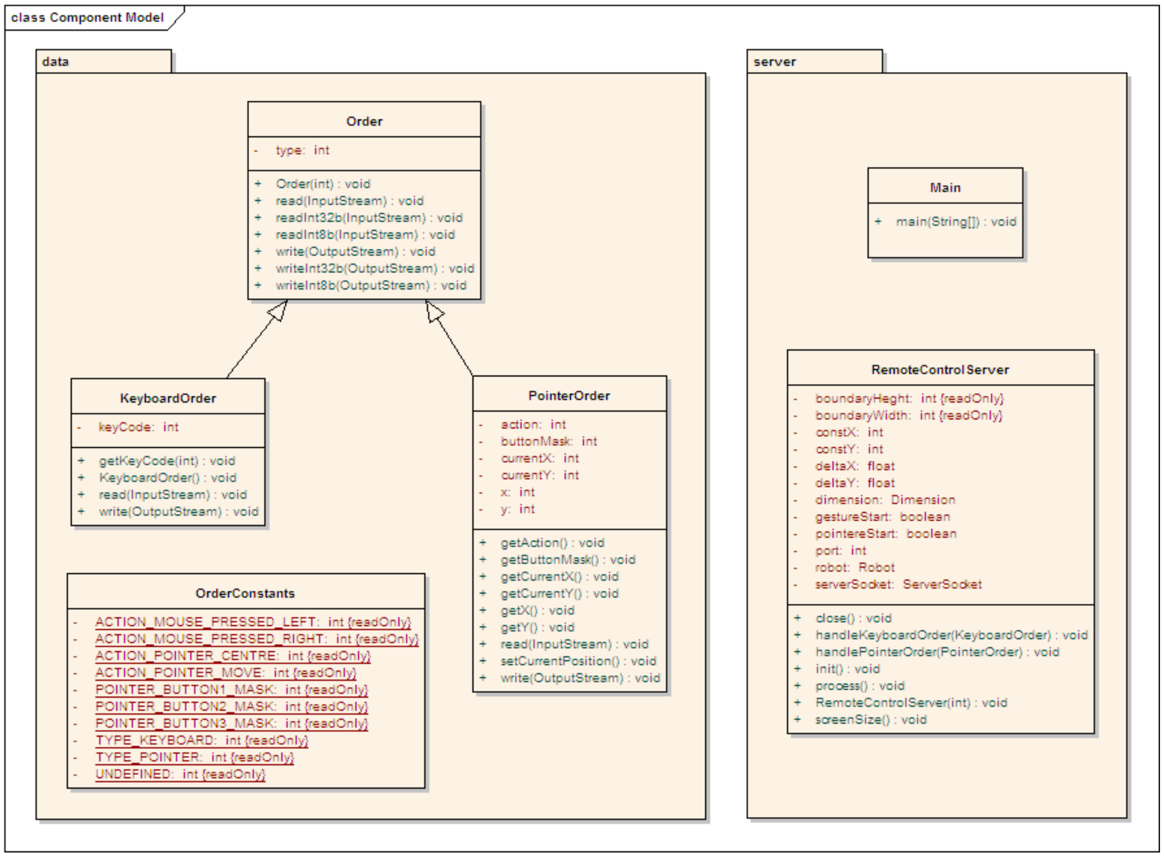

class Component Model
data
Order
- type: int
+ Order(int) : void
+ read(InputStream) : void
+ readInt32b(InputStream) : void
+ readInt8b(InputStream) : void
+ write(OutputStream) : void
+ writeInt32b(OutputStream) : void
+ writeInt8b(OutputStream) : void
KeyboardOrder
- keyCode: int
+ getKeyCode(int) : void
+ KeyboardOrder() : void
+ read(InputStream) : void
+ write(OutputStream) : void
PointerOrder
- action: int
- buttonMask: int
- currentX: int
- currentY: int
- x: int
- y: int
+ getAction() : void
+ getButtonMask() : void
+ getCurrentX() : void
+ getCurrentY() : void
+ getX() : void
+ getY() : void
+ read(InputStream) : void
+ setCurrentPosition() : void
+ write(OutputStream) : void
OrderConstants
- ACTION_MOUSE_PRESSED_LEFT: int {readOnly}
- ACTION_MOUSE_PRESSED_RIGHT: int {readOnly}
- ACTION_POINTER_CENTRE: int {readOnly}
- ACTION_POINTER_MOVE: int {readOnly}
- POINTER_BUTTON1_MASK: int {readOnly}
- POINTER_BUTTON2_MASK: int {readOnly}
- POINTER_BUTTON3_MASK: int {readOnly}
- TYPE_KEYBOARD: int {readOnly}
- TYPE_POINTER: int {readOnly}
- UNDEFINED: int {readOnly}
server
Main
+ main(String[]) : void
RemoteControlServer
- boundaryHeght: int {readOnly}
- boundaryWidth: int {readOnly}
- constX: int
- constY: int
- deltaX: float
- deltaY: float
- dimension: Dimension
- gestureStart: boolean
- pointereStart: boolean
- port: int
- robot: Robot
- serverSocket: ServerSocket
+ close() : void
+ handleKeyboardOrder(KeyboardOrder) : void
+ handlePointerOrder(PointerOrder) : void
+ init() : void
+ process() : void
+ RemoteControlServer(int) : void
+ screenSize() : void

FIGURE 8. Software architecture of the server application.

4.2.2 Communication interface

Like in every type of communication, also computers or programs inside have to understand each other and it is accomplished by a means called **interface**. Interface in the project consists of few events, called **Order**. These orders, represented by implemented class, have to be exactly the same on the client side. Without the same “language”, it is impossible for the server to understand clients commands. Basically, the computer communication is only bits, zeros and ones, but merged to the bytes, then to integers, can be a full-valued way of interaction. All interactions are based only on the integer values.

Class Order is a **parent class** (superclass) providing to all its child classes (subclasses) core functions: reading and writing 32 bit Integers from, and into a socket. It is needed due to **read()** method from **InputStream** class and **write()** method from **OutputStream**, since they read 8 bit, but Java's Integer is 32 bit. Here a reader can see bitwise conversions in both essential methods:

```
public static int readInt32b(InputStream in) throws IOException {

      /* reading 4 bytes from the socket for conversion */

      int a = in.read();

      int b = in.read();

      int c = in.read();

      int d = in.read();

      /* creating 4 bytes Integer with bite shift operators */

      if ((a < 0) || (b < 0) || (c < 0) || (d < 0)) // returns UNDEFINED

            return -1;

      return (a << 24) | (b << 16) | (c << 8) | (d);

}


public static void writeInt32b(OutputStream out, int value) throws
IOException {

      /* splitting Integer into four one-Byte chunks and sending */

      out.write((value >>> 24) & 0xFF);

      out.write((value >>> 16) & 0xFF);

      out.write((value >>> 8) & 0xFF);

      out.write((value) & 0xFF);

}
```

The Order also has one attribute called **type** and it is part of every event. Actually, it determines what type of command it is, whether keyboard or pointer one. Every type is an integer value defined in **OrderConstants** interface class along with other constants used to interact.

4.2.3 Implementation and data flow

In this section “the life” of the server is described from its start till executing a received command. In the **main** method instance of **RemoteControlServer** class is created and in the new thread its **process()** method is run. That simply means that process() method runs in the infinite loop, until it is not shut down or some unexpected error occurs.

```
public void process() throws IOException {
	if (serverSocket == null)
		serverSocket = new ServerSocket(this.port);
	System.out.println("RemoteServer: Waiting for connection.");
	Socket socket = serverSocket.accept();
	System.out.println("RemoteServer: Accepted connection.");

	InputStream socketIn = socket.getInputStream();
	OutputStream socketOut = socket.getOutputStream();
	...
```

Process()'s task is to create a new **socket** connection for the client and when clients tries to connect, the server automatically accepts it. A user is then connected and the next part is reading data sent by the client from the socket. For this purpose infinite **while** loop reads data from the socket. It is terminated by disconnection of the user or sending mismatch data.

```
disp_loop:while (true) {
	int type = Order.readInt32b(socketIn); // reads type
	switch (type) {
	case OrderConstants.TYPE_POINTER: // if pointer
		PointerOrder po = new PointerOrder();
		po.read(socketIn); // reads rest of order
```

```
            handlePointerOrder(po); // handles received data
            break;
      case Order Constants.TYPE_KEYBOARD:
            KeyboardOrder ko = new KeyboardOrder();
            ko.read(socketIn);
            handleKeyboardOrder(ko);
            break;
      default:
            System.out.println("Unknown order type!");
            break disp_loop;
      }
}
```

As a first value in every single order, there has to be **type**. According to type, switch statement differs, what type of command should be executed. Firstly, instance of **PointerOrder** or **KeyboardOrder** is created and the rest of the data is read. Secondly, handle method is called to process data received into PointerOrder's or KeyboardOrder's private attributes. In case of unknown type, **break** command ends up while loop.

```
private void handleKeyboardOrder(KeyboardOrder ko) {
      System.out.println("Key pressed: " + ke.getKeyCode());
      robot.keyPress(ke.getKeyCode());
      robot.keyRelease(ke.getKeyCode());
}
```

HandleKeyboardOrder method is really simple. Based on the second given value besides type, it performs keyboard button press through the **robot.keyPress**(KeyEvent.VK_*something*) with virtual key code parameter. All the virtual codes can be found on this constants' page http://download.oracle.com/javase/1.4.2/docs/api/constant-values.html#java.awt.event.KeyEvent.CHAR_UNDEFINED.

Right after key press is automatically executed **robot.keyRelease**(KeyEvent.VK_*something*) is called with an identical value as a key press. Of course, it could be sent by the user at the moment when he/she in fact releases the button, but for this project's purposes it was not needed. Computer game or some more complex application could have a feature of sending the same event while the button is held.

HandlePointerOrder method has much more logic and algorithmization inside.

```
private void handlePointerOrder(PointerOrder po) {
      po.setCurrentPosition();
      sreenSize();
      constX = dimensions.width / boundaryWidth;
      constY = dimensions.height / boundaryHeight;

      switch (po.getAction()) {
      case OrderConstants.ACTION_POINTER_CENTRE:
            ...
      case OrderConstants.ACTION_POINTER_MOVE:
            ...
      case OrderConstants.ACTION_MOUSE_PRESSED_LEFT:
            ...
      case OrderConstants.ACTION_MOUSE_PRESSED_RIGHT:
            ...
      default:
      }
}
```

The first step is to find out the current position of the cursor to have starting point, since the robot uses **absolute positioning**. A savoir, the robot needs for instance values 800 and 600 to position the cursor to the point that is distant 800 screen pixels to the right and 600 screen pixels lower from the left top corner. The client sends only changes of his movement, therefore the server has to modify the current location in compliance with these changes and pass the final absolute values to the robot. This approach puts fewer computing operations on the mobile application and also different computers have different screen resolutions – the user does not need to adjust to diverse PCs.

SetCurrentPosition() assigns actual position into private attributes (absolute position) and **screenSize()** puts screen resolution into dimensions variable. **ConstX**'s and **constY**'s are just multiplying constants because received values are too small to cover the whole screen size with a moving phone round a slight amount. All these steps are only preparation before the actual calculations of received values.

The second PointerOrder's attribute (after type) is **action** regarding to the pointer events and is differential value of the switch statement.

One possibility of action is centering (**ACTION_POINTER_CENTRE**). It is only positioning cursor to the center of the screen. This feature was added, because users want to “calibrate” their device at some point.

Move branch (**ACTION_POINTER_MOVE**) performs all necessary calculations and the robot places the cursor.

```
case OrderConstants.ACTION_POINTER_MOVE:

      float divideBy = 10.0f;

      po.setCurrentPosition();

      int curX = po.getCurrentX(); int curY = po.getCurrentY();

      deltaX = (float) po.getX() / divideBy;
```

```
    deltaY = (float) po.getY() / divideBy;


    for (int i = 1; i <= 10; i++) {

            robot.mouseMove((int) (curX + deltaX * i * constX),

                            (int) (curY + deltaY * i * constY));

            robot.delay(4);

    }

    break;
```

The client sends pointer events every **50 milliseconds** – 20 times per second. This interval was chosen, because there is no overload of data sent, but the movement is not even very abrupt. Despite all effort, the cursor is not as smooth as in the regular mouse, because of these 50 millisecond intervals and also the robot does not shuttle the cursor, but simply “jumps” to the new position. Moreover, the sensor in the phone generates values, even while lying on the table, hence averaging is also needed there that causes more inaccuracies. These issues concerning mobile device will be analyzed in a more detailed way in the next chapter. To avoid seeming blinking (caused by untender movements) of the cursor, time between orders is filled with moving manually the cursor to the new position, instead of popping it in the new location. For creating those transitions two different approaches were tried. The first one (shown above) feels more “real-time”, but is not as accurate as the second, which has a notable delay. After some testing by colleagues, the first one looked more user-friendly and was chosen as a better one.

The whole transition process is divided into the ten pieces, so between two commands the cursor moves **10 times**. The received movement change values are divided by ten and assigned to **delta** variables. Then, in the **for** loop, is a cursor step by step moved to the new position. This significantly improves the perceived smoothness; however it is still not perfect. The loop has to have 4 milliseconds delay, because it is executed faster than 50 milliseconds. Naturally, human eye cannot record those ten steps without slowdown.

In order to make movements even smoother, **divideBy** could be changed to higher values. The cursor then moves evidently much more fluently, but on the other hand it is not clever to put a higher number than 50, since the delay gap grows even more. In addition, the whole application is busier looping more times. Consequently, the robot is active nearly for the whole 50 milliseconds, therefore any delay is odd.

The shown move branch is applicable only for Pointer part of the client application. Touchpad and pointing stick have different processing of received values. The main difference between pointer and touchpad / pointing stick is that touchpad's data are not actually processed at all, thus dividing process and multiplying values with constX and constY are not needed. What the client sends, the server just copies. In brief, instead of

```
robot.mouseMove((int) (curX + deltaX * i * constX),

                 (int) (curY + deltaY * i * constY));
```

the server simply runs

```
robot.mouseMove((int) (curX + actualPosX), (int) (curY + actualPosY));
```

To summarize, the second **mouseMove** method only adds given values to the current absolute position of the cursor.

This change significantly improves the accuracy of the touchpad to be almost identical with real ones in laptops, for instance. The fact that even with the real touchpad it is impossible to cover the whole screen, one finger scroll allows reducing all calculations causing inaccuracy as much as possible. Unfortunately, it cannot be said about pointer function.

The last two branches in action switch statement only perform mouse buttons presses and releases using **robot.mousePress**(InputEvent.BUTTON*X*_MASK).

- **BUTTON1_MASK** – left mouse button
- **BUTTON2_MASK** – middle mouse button
- **BUTTON3_MASK** – right mouse button.

More constants can be found here:
http://download.oracle.com/javase/1.4.2/docs/api/constant-values.html#java.awt.event.InputEvent.BUTTON1_MASK

4.3 Client application

4.3.1 Description

As for client side, here is the main logic and idea. Being short, the application's main functionality is to send simple orders to the server, or identify movements or gestures performed by a user and then send them. The core functions are as follows:

- **controlling a mouse cursor** with changing phone's position
- recognizing few movement **gestures**
- **touchpad** simulator
- **pointing stick** simulator.

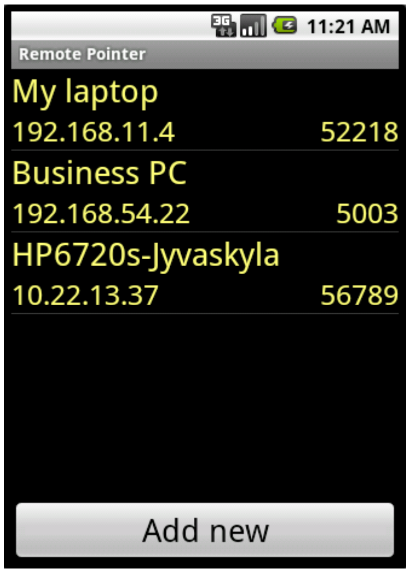
A built-in compass sensor was chosen to read the data it produces and utilize them in algorithms. To mention some other minor features, the application is able of **adding**, **deleting** and **editing PC's IP** addresses and storing them into the **database**.

To run this application users have to own mobile phones with Android OS with a minimum 1.5 version. The next requirement is built-in compass sensor.

4.3.2 Application usage and implementation

A prerequisite before starting up the application is to manually connect to the **Wi-Fi** a user is going to use. The application itself cannot connect to the router, only to the socket, if available.

After the launch is called **onCreate** method of the **MainActivity** class that sets up initial necessaries such as reading records with PCs from the database (if that does not exist, then it creates a new one), putting them into the list and finally showing them in the layout. “**Add**” button is situated in the bottom part of the screen. On a click, a new layout is loaded with a text field for entering a new PC that can be stored in the database for the next usage.



My laptop	
192.168.11.4	52218
Business PC	
192.168.54.22	5003
HP6720s-Jyvaskyla	
10.22.13.37	56789

FIGURE 9. List of PCs.

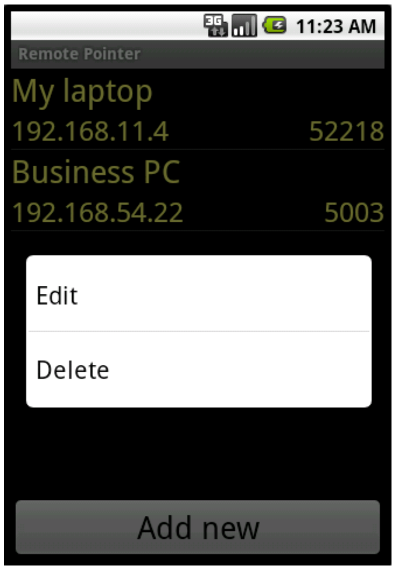


FIGURE 10. Item's context menu.

If there are entries in the database, a user can press and hold the selected item to launch the context menu with two options: either to “**Edit**” or to “**Delete**” chosen entry. Apparently, the delete function deletes a record from the database and the edit function offers a possibility to change details of the saved items. The layout of the Edit form is the same as when adding a new entry, except that the text fields contain old details for update.

4.3.2.1 Database

A database has a minor mission in the project. Its only purpose is the user's comfort – no need to insert the same data every time. Accordingly, the structure is very simple.

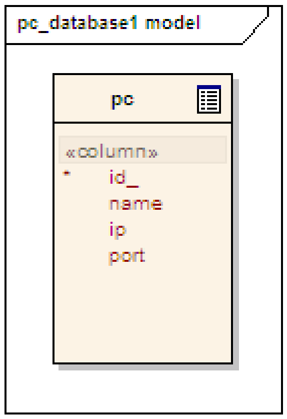


FIGURE 11. Data model.

Creation of the database:

```
super(context, DATABASE_NAME, null, 1);

db.execSQL("CREATE TABLE " + DATABASE_TABLE

            + " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " + NAME

            + " TEXT, " + IP + " TEXT, " + PORT + " TEXT);");
```

Super clause calls constructor of the **SQLiteOpenHelper**, because class handling database, extends SQLiteOpenHelper. The last parameter in the constructor indicates that the database is in version number one.

As previously written, after the application's launch, all entries are put into list a through the use of **SimpleCursorAdapter** class.

```
String[] resultColumns = new String[]{"_id", NAME, IP, PORT};
cursor = db.query(DATABASE_TABLE, resultColumns, null, null, null, null,
                 null);
ListAdapter adapter = new SimpleCursorAdapter(this,
                      R.layout.list_item, cursor,
                      new String[] {NAME, IP, PORT},
                      new int[] {R.id.namePC, R.id.ipPC, R.id.portPC});
setListAdapter(adapter);
```

Parameters of SimpleCursorAdapter's constructor are as follow:

- context, where the ListView is running
- resource identifier of a layout file that defines views for this list item wherein data will be shown
- database cursor containing all fetched data
- a list of column names representing the data to bind to the UI
- the views that should display column in the antecedent parameter.

4.3.2.2 Connection

In the list of predefined computers, the user can choose by one simple clicking on the wanted item. As a result, following method is called in order to connect to the selected PC.

```
private void connectToSocket() throws Exception {

      inetAddr = InetAddress.getByName(ip);

      socket = new Socket(inetAddr, port);

      PointerOrder po = new PointerOrder(ACTION_POINTER_CENTRE);

      out = socket.getOutputStream();

      po.write(out);

      out.flush();

}
```

Connection to the demanded server's IP and specific port is trying to be established. A command for centering the mouse cursor is immediately sent. If establishment was not successful, the user receives “Connection error occurred.” message via **Toast** widget.

4.3.2.3 Main Activity

When the result of connection is alright, a new activity named **Compass** starts. It holds the core client's program logic. The user interface has only few buttons for presentation of **keyboard commands** and **mouse buttons presses**. Besides handling buttons, it includes:

- initialization of the sensors
- reading the data from them
- processing of it and all needed calculations
- sending detected orientation changes / gestures.

All referred procedures are explained in the next chapter.

The following figure illustrates a user interface of the client application for a mouse simulator and gesture recognition.

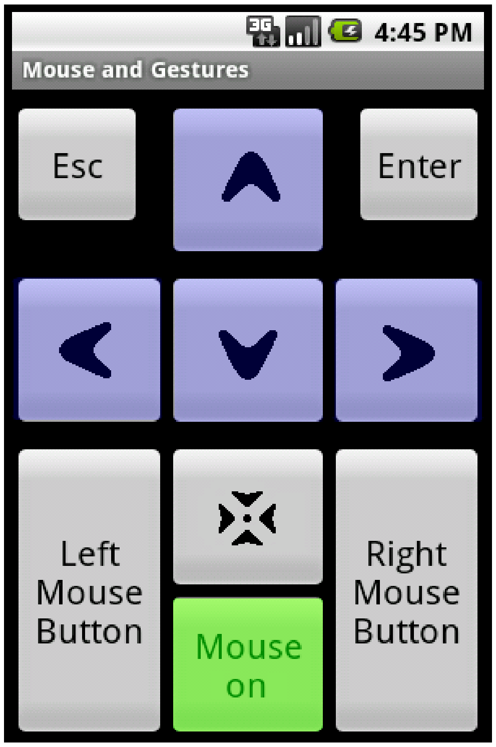


FIGURE 12. GUI of the client application.

4.3.2.4 *Compass*

A compass sensor has in Android API **TYPE_ORIENTATION** tag. Like in the real compass, also this one is highly sensitive for tiny changes. For this reason it generates new values even when it lies still on the table. Based on observation it happens approximately every **20-25 milliseconds**. These values are measured for all three pivot axes **X** (**value[1]**), **Y** (**value[2]**), **Z** (**value[0]**) and are presented in degrees. More specifically, the most used data for classic compass applications are differences in Z coordinate. Actually it is **azimuth**, the angle between the magnetic north direction and the Y axis, around the Z axis (0 to 359). North is represented by 0, 90 stands for east, 180 refers to south and 270 is west. A direction of the increase is clockwise. **Pitching** around X axis (-180 to

180) provides positive values when the Z axis moves toward the Y axis. **Rolling** around Y axis (-90 to 90) gives positive values when the X axis moves toward the Z axis.

"A reader should note that this definition is different from yaw, pitch and roll used in the aviation, where the X axis is along the long side of the plane (tail to nose)." (Android Developers – sensors)

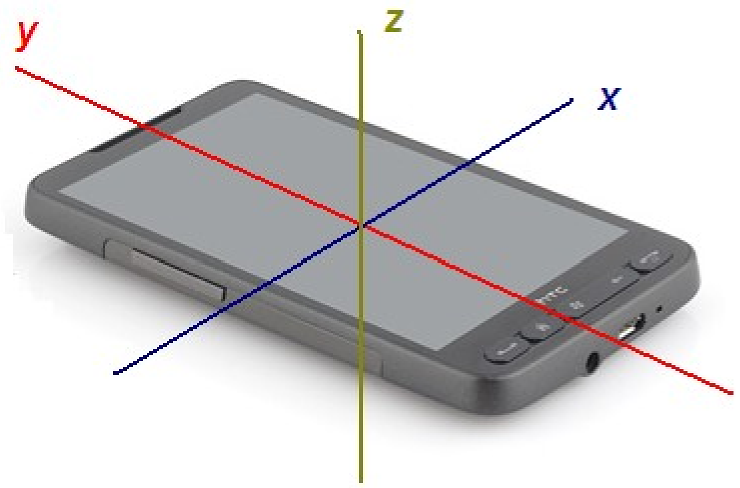


FIGURE 13. Orientation of axis in the Android device.

4.3.2.5 Pointer

This paragraph describes the processing sensor's data to achieve as good pointing results as possible. The shown source code is not the real one, because the actual code is too long. The algorithm below presents the main logic and how it works.

```
if (firstTime) {
      lastX = currentX;

      lastY = currentY;

      lastTime = currentTime;

} else {

      // every 50 milliseconds

      if (currentTime - lastTime > 50) {

            // if change of X or Y angle was bigger than 2 degrees

            if ((currentX - lastX > 2) || (currentY - lastY > 2)) {

                  move(currentX - lastX, currentY - lastY);

                  lastX = currentX;

                  lastY = currentY;

            }

            lastTime = currentTime;

      }

}
```

This algorithm is executed every time when the sensor registers any change. **CurrentX** and **currentY** contains current values from the sensor. In the first running of the code values are used only for something similar to calibration to have a starting point. Since this algorithm is executed really often (3-4 times in 100 milliseconds), the data loss is not harmful. **CurrentTime** is also provided by the sensor's function.

Data are gathered every 50 milliseconds and only if a change in any axis is higher than **2 degrees**. The first condition could be changed for some type of weighted mean or averaging, but there are not so much data for processing in that short time period. The second one is used to remove flickering of the cursor. They are caused by sensor's “inaccuracy” by calling **onSensorChanged**. Even, when the telephone is lying on the table and is not moving at all, the sensor generates small changes and those would worsen the wanted precision. As mentioned before, the sensor can at some point pass zero boundary and handling of these situations is missed in the source code, but has very important status. When it is found out to the which direction the user is moving, then the **move(X, Y)** method is called with Integer values (left – negative X, right – positive X, up – negative Y, down – positive Y) and those values are sent to the server. The last step is assigning the current position and time stamp to variables with “last” prefix, to have reference for comparing when onSensorChanged is called.

4.3.2.6 Gestures

The application recognizes two types of gestures. One is swinging (pitching) around X axis and the other is swinging (rolling) around Y axis. Axes are illustrated in figure 13. Every gesture has its assigned different keyboard button press. More specifically, pitching around X axis upward acts same as “Arrow Up”, opposite direction is “Arrow Down”. Rolling to the right side represents “Enter” key and swing to the left side is “Escape” key.

The described algorithm only demonstrates the main idea of the Y related axis' orientation changes.

```
if (firstTime) {
      lastTime = currentTime;
      lastGesture = currentTime;
} else {
      // every 50 milliseconds
      if (currentTime - lastTime > 50) {
            if ((Abs(currentZ) > 20) && (!startCount)) {
                  direction = (currentZ > 0) ? RIGHT : LEFT;
                  startCount = true;
            timerStarts = currentTime;
        }
        if (startCount)
            if (Abs(currentZ) > 55) {
                if ((currentTime - timerStarts) < 200)
                    if ((currentTime - lastGesture) > 500) {
                        if (direction == RIGHT)
                            gesture(RIGHT);
                        else
                            gesture(LEFT);
                        lastGesture = event.timestamp;
                    }
            }
    lastTime = currentTime;
    }
}
```

Initially, in the first run current values were just assigned as is done in the previous example. Afterwards, it is determined every 50 milliseconds, whether the angle of the phone is higher than **20 degrees**. This is considered a beginning of the gesture, in case it is not only part of a gesture that already has begun in the previous method call. At this point it had to be detected, to which side the phone is turned. Clockwise movement generates positive values and in the algorithm it means to the **RIGHT**. Anticlockwise has negative values and is labeled as **LEFT**. The next step is to reach **55 degrees** angle that is taken as accomplishing the gesture, however it has to be done in less than **200 milliseconds**. The time is measured from passing 20 degrees till reaching 55 degrees.

To sum up, a user has to rotate his phone by at least 35 degrees within less than 200 milliseconds. This combination, along with some others, seems to give good results according to testing by more users.

This is valid except that it had to be taken into account that the phone is also returning to the default position. Backward movement was at the early stages detected as movement to the opposite side. What is more, limitation about gesture frequency had to be introduced. In this setup, only one gesture per half second is accepted.

Algorithm for X axis is almost identical, except for the angle. In fact, swinging around X axis is more difficult, especially down; therefore there is only **45 degrees** angle to be reached. Even though the algorithm has some flaws, after few minutes everyone is capable of mastering predefined gestures.

4.3.2.7 Touchpad

Implementation of the touchpad was much easier in comparison with the Pointer or gestures. Android's **OnGestureListener** class has useful methods for identifying different touch gestures done with **finger**. Here is a list of the mentioned methods:

- onDown – when a simple tap occurs
- onFling – notifies of a fling event when it occurs
- onLongPress – notifies when a long press occurs
- onScroll – when user scrolls on the screen
- onShowPress – when the user has performed down event and not a move or up yet
- onSingleTapUp – opposite gesture than onDown.

OnScroll was chosen due to the values it generates. Float distance alteration of X and Y coordinate. This provides all necessary data, almost ready for sending to the server. Only few slight changes of raw values needed to be done. To avoid little unintentional and in most cases unwanted motions, every value lower than **0.2 pixel**, is simply ignored.

4.3.2.8 Pointing stick

As noted in the introduction, pointing stick (**TrackPoint** is IBM's trademark) is a small **red joystick** located between the “**G**”, “**H**”, “**B**” buttons in the middle of the keyboard in some models of laptops (mainly IBM, Dell, HP). Speed of a cursor movement depends on the quantity of power applied to the stick to the chosen direction. Hence a **red dot** positioned in the middle of the phone's screen to represent the reference point was created. After positioning a finger on the dot and moving it away from the dot, the cursor moves accordingly to the appropriate direction. The farther a user has his finger from the center, the faster the cursor is moving.

The source code below is a simplified version of the real one:

```
if (currentTime - lastTime > 30) {

      if (isNotInCircle(currentX, currentY)) {

            deltaX = (currentX > centreX) ? currentX + centreX :

                        centreX - currentX;

            deltaY = (currentY > centreY) ? currentY + centreY :

                        centreY - currentY;

            deltaX /= 5;

            deltaY /= 5;

            lastTime = currentTime;
            move(deltaX, deltaY);
      }
}
```

Firstly, time difference of 30 milliseconds between two onScroll events is checked. The result is assigned to the **deltaX** and **deltaY** variables. The next step is to ignore scrolls which happen inside the red point. **IsNotInCircle** method determines, if the fingertip lies in the red area. Next two lines mirror the differentiation to which position the scrolling happens. The server executes touchpad's and pointing stick's data in the same way and in this particular case the sent values are enormously huge, especially those far from the center dot, so there is need to divide them by five. Finally, the values are sent to the server side.

Even though the pointing stick utilization gives satisfying results, mobile's screen does not provide a reference point unlike real physical dot under a person's finger. The user then can lose a realization, where on the screen he/she is in case he focuses his sight mainly on the monitor. As a result of this, a real physical pointing stick is definitely much more convenient to use.

4.4 Testing

Since the main development process lasted barely two weeks, there was no room for special unity test or whatsoever. Moreover the project is not extensive to such degree that absence of them could markedly affect the final result. However, some kind of testing was carried to assure as the best results as possible. The highest importance was put on accuracy and a “feel” during the usage. Various angles to be reached in order to accept a gesture were tried, different frequencies for sending of the orders we tested, diverse changes in sensitivity and ranges of movement for the mouse simulator we experimented. To set it up as genuine as possible, few colleagues were included to the testing process.

In despite of every user has his own vision how it should behave and because of that it is hard to adjust it to everyone's complacence. Afterall the effort of creation of the most common behavior and settings were accomplished.

5 CONCLUSION

This chapter is aimed at concluding the results of the project called “Utilization of a mobile phone as a pointing device”. The actual version of the project meets the set requirements very well. However, there is still room for different optimization and precision related issues improvements. The application provides all the four main features needed – hardware mouse substitution, gesture recognition, touchpad and pointing stick simulator. The main area for a research and an algorithmization were sensors in the smartphone and generated values.

A utilization of gestures recognition and touchpad simulator provides very good usability as well as pointing stick and mouse simulator; however, the last two mentioned need more fine-tuning. All known deficiencies and possible corrections related to them are described in section 6 .

The primary objective of the assigned project was to create an application that should be as easy to use and as plausible as possible. In compliance with the projects the thesis to report process of the whole development was written from receiving requirements, through designing, implementing and finally testing.

Forasmuch as the application was only meant to be only for the internal company's needs, it is impossible to present customer's feedback at this point. Despite of that, the supervisor was very pleased and satisfied with the result, moreover other superiors appreciated the application and evaluated it as more than satisfactory. From the author's point of view the result is very good regarding to the fact that the author attended only one course of Android platform programming and had no previous experience with the most parts of this project out of consideration for the deadline for it, plus the fact that it was the author's very first project in a company starting on the second day in the job.

6 DISCUSSION

As previously stated, the final version of the project meets the criteria even though it is not as good as it should be. It cannot be considered as a ready-to-sell product, because a fine-tuning is required. The main reason is the fact that the project had a deadline, although it was only an internal project for the company and I had no time to work on it in the spare time. Nevertheless I will try to focus on some areas of the project that should be enhanced and present some thoughts about them. Following paragraphs are mainly intent on pointer and gestures functions.

Delay

The delay and the responsiveness in the IT field are very important issues and every designer has to pay special attention to them during the designing process. In this particular case the responsiveness of the application itself is very good but the delay in the execution of the commands has room for improvements. Essentially, it is not a Wi-Fi issue, but a combination of few aspects.

The first aspect is the frequency of the generating position changes of the smartphone which would be sufficient if there were not inaccurate values. Averaging of the raw values takes some time causing a significant part of the delay. The second delay issue is caused by the way of processing the received command. At first it has to be received and after that processed.

Users are very familiar with regular mice. They copy the movements instantly and every small delay is immediately perceived. From this point of view the regular mouse has a big advantage and this application is not very convenient for working but only for occasional usage.

Precision

Flows in the precision have few main reasons. Sensors in the smartphones are excessively sensitive and inaccurate comprising especially magnetometer (compass). The fact that they produce changed values even when the device stands still, shows it. Without any averaging the cursor jumps from one place to another enormously. On the other hand, the more values are averaged, the huger is the delay and the more the movement reminds floating. No sharp movements are possible. This is the major problem, to find a compromise between delay and precision. Usage of a compass sensor was a part of the requirements; however, Android platform smartphones have also other types of receptors. Exertion to research them could give better results as well. The next problem is caused by Robot's move() function. Instead of fluent move from one point to another it simply blinks to a new position. As a result a path calculation had to be implemented which is obviously not the real path of the cursor. Casting from more precise values to less could slightly affect precision.

Miscellaneous

Even though this project focused on the utilization of the mobile phone as a pointing device, it uses Java's Robot which is capable of simulating also button presses. The application includes few demonstration buttons such as Arrows, Enter, Backspace, Escape but adding a virtual keyboard would be really easy.

A user could also appreciate modification of the project to the version without the server part. Somehow it sends the commands directly to the OS for executing. This approach reduces delay and with change of Wi-Fi with Bluetooth or IrDA also reduces required hardware dependencies.

Altogether

Despite of all the minor shortages and the lack of the time, the application is very handy and easy to use. Not only in emergency cases when a user forgets / loses his mouse but also as an attractive way to handle PC. Gesture function will definitely do. Especially touchpad behaves like a real one and the pointing stick as well. Its only disadvantage in comparison with a built-in keyboard is that a user can not feel it since it is only virtual. It is generally known that users prefer sensational feedback, whether something real to touch, or feedback in form of sounds or visual response.

During the developing process I discovered similar, already existing solutions on the market. One is for iPhones and the other is for Android smartphones. In this regard the presented project is apparently not “ground-breaking” regarding to its idea. Nevertheless, I have not only deepened my knowledge in for me known matters such as Android and Java programming but acquired plenty of new skills and information. For me that matters.

The last remaining thing is to hope for interest of customers in the idea so it would become a real project or possibly a part of a more complex solution.

REFERENCES

Accelerometer EN Wiki, http://en.wikipedia.org/wiki/Accelerometer, Referred to on 27 April 2011.

Android Developers – sensors, http://developer.android.com/reference/android/hardware/SensorEvent.html, Referred to on 27 April 2011.

Android Operating System EN Wiki, http://en.wikipedia.org/wiki/Android_(operating_system), Referred to on 11 April 2011.

Android Operating System SK Wiki, http://sk.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%BD_syst%C3%A9m), Referred to on 11 April 2011.

Compass EN Wiki, http://en.wikipedia.org/wiki/Compass, Referred to on 27 April 2011.

Eclipse EN Wiki , http://en.wikipedia.org/wiki/Eclipse_(software), Referred to on 11 April 2011.

Eclipse SK Wiki, http://cs.wikipedia.org/wiki/Eclipse_(v%C3%BDvojov%C3%A9_prost%C5%99ed%C3%AD), Referred to on 11 April 2011.

Gyroscope Cambridge Encyclopedia, http://encyclopedia.stateuniversity.com/pages/9304/gyroscope.html, Referred to on 27 April 2011.

Ixonos internal brochure, Ixonos's internal informative brochure, Referred to on 11 April 2011.

Java CS Wiki, http://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk), Referred to on 11 April 2011.

List of Android devices EN Wiki, http://en.wikipedia.org/wiki/List_of_Android_devices, Referred to on 11 April 2011.

Mouse SK Wiki, http://sk.wikipedia.org/wiki/My%C5%A1_(hardv%C3%A9r), Referred to on 9 April 2011.

Scrum EN Wiki, http://en.wikipedia.org/wiki/Scrum_(development), Referred to on 11 April 2011.

Sensor EN Wiki, http://en.wikipedia.org/wiki/Sensor, Referred to on 27 April 2011.

Touchpad SK Wiki, http://sk.wikipedia.org/wiki/Touchpad, Referred to on 10 April 2011.

Resources

http://stackoverflow.com/

http://developer.android.com/guide/index.html

http://download.oracle.com/javase/1.4.2/docs/api/overview-summary.html