

Gang Bai

Mobile Phone Programming - Based on Mobile Sensor API for User Interface

Bachelor's Thesis
Information Technology


May 2010



MIKKELIN AMMATTIKORKEAKOULU

Mikkeli University of Applied Sciences

DESCRIPTION

 <p>MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences</p>		Date of the bachelor's thesis 10 May 2010
Author Gang Bai	Degree programme and option Information Technology	
Name of the bachelor's thesis Mobile Phone Programming - Based on Mobile Sensor API for User Interface		
Abstract <p>Recently, it is a popular trend to incorporate sensors into personal consumer devices. This action provides more interaction with users and optimizes users' experiences. For instance, the remote controller of Wii employs an accelerometer that enables players to simulate motions in real life. The ipod touch can automatically change the screen to display vertically or horizontally according to the placement of the device.</p> <p>Also, various kinds of sensors are embedded in mobile devices to provide more services and functions. The usual types of mobile sensors include the accelerometer, ambient light sensor, magnetometer and rotation sensor. Besides, the mobile phone company Nokia has opened the access to the hardware to some extent and provided API for developers to access them conveniently. In this way, the users are able to not only enjoy the sensor-related services provided by the phone manufacturers, but also to run those sensor-based applications developed by individual developers.</p> <p>In this final thesis, I programmed a sensor-controlled user interface by implementing the accelerometer inside the mobile device. The application is based on Java 2 ME platform and Java SDK emulator. A three-axis accelerometer and the mobile sensor API (JSR 256) are necessarily required for the mobile device to run the application. This user interface provides a means to switch the menu by the movement of the device, compared with the traditional approach of pressing a button.</p>		
Subject headings, (keywords) Mobile phone, programming, sensor, accelerometer, API		
Pages 51p.+app.16	Language English	URN
Remarks, notes on appendices		
Tutor Matti Koivisto	Employer of the bachelor's thesis Mikkeli University of Applied Sciences	

Acknowledgement

I want to express my appreciation to all those people who helped me in completing the diploma thesis. Without all their help, I cannot finish the thesis much soon.

Firstly, I would like to thank my supervisor, Matti Koivisto. He gave me critical guidance when I was confused what to do at the initial stage of the thesis. Also, when I came across difficulties and felt frustrated, he always encouraged me not to give up. I learned a lot as well from his rich experience in programming when debugging my own application.

Secondly, I would like to express my deep appreciation to Selin Jukka. He is really an energetic and rich-experienced engineer in programming. I would always listen carefully to his advice when I encounter with technical challenges. He is very patient and kindhearted.

Then, thanks to those smiling faces of my friends and strangers when I was down, I could keep on going instead of being obsessed in the failures.

Lastly, I want to give my deepest thanks to my beloved parents. Whenever I hear their voices from China, I simply forgot all the difficulties and become vitalized again. Their continuous support is the biggest motivation for me to keep up with an optimistic attitude. And I will always keep on striving in my life for that reason.

LIST OF ABBREVIATIONS

AMS	Application Management Software
API	Application Programming Interface
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
J2SE	Java Standard Edition
J2EE	Java Enterprise Edition
J2ME	Java 2 MicroEdition
JVM	Java Virtual Machine
JAR	Java Archive (File)
JSR	Java Specification Request
MIDP	Mobile Information Device Profile
PDA	Personal Digital Assistant
SDK	Software Development Kit
RDA	Remote Device Access

LIST OF FIGURES

Figure 2.1: A mercury-in-glass thermometer	4
Figure 2.2: A thermocouple	4
Figure 2.3: A voltagemeter	5
Figure 2.4: A touch-sensitive elevator button	6
Figure 2.5 A three-axis accelerometer board.....	8
Figure 2.6: An example of sensitivity and linearity of an accelerometer	10
Figure 2.7: An example of iphone using the accelerometer for UI control.....	12
Figure 2.8: Wii Remote	12
Figure 3.1: Various Java platforms with theirs areas of application	15
Figure 3.2: The structure of a digital media platform based on CDC.....	18
Figure 3.3: The structure of a CLDC wireless platform based on CLDC	19
Figure 3.4: The structure of the JSRs and their fields of application	22
Figure 3.5: The lifecycle of a MIDlet.	23
Figure 4.1: Class diagram of javax.microedition.sensor package	25
Figure 4.2: The parameter values of an accelerometer	28
Figure 4.3: Mono-display device and its co-ordinate axes.....	29
Figure 4.4: ChannelInfo information of axis_y.....	29
Figure 5.1: Java platform manager.....	31
Figure 5.2: Choose the type of the platform.....	32
Figure 5.3 Installing Java SDK 3.0	33
Figure 5.4: Create a new project of MIDP application	33
Figure 5.5: The setting of device configuration and device configuration profile.....	34
Figure 5.6: The running result of the Hello MIDlet.....	36
Figure 5.7: The SDK progress.....	37
Figure 5.8: Nokia PC suite	38
Figure 5.9: The working flowchart of RDA.....	38
Figure 5.10: The initial menu of my application.....	39
Figure 5.11: External events generator with axisZ value less than -0.5.....	40
Figure 5.12: The screen of menu Level1.....	41
Figure 5.13: The screen of menu Level2.....	41

Figure 5.14: External events generator with axisZ value higher than +0.5	42
Figure 5.15: Error message of no valid sensor found	46
Figure 5.16: Running results of the SensorInfo MIDlet on the real device and emulator.....	46

TABLE OF CONTENTS

1 INTRODUCTION.....	1
2 SENSOR AND SENSOR TECHNOLOGY	3
2.1 Definition of Sensor.....	11
2.2 Usage of Sensors	5
2.3 Sensitivity of Sensors.....	6
2.4 Types of Sensors	6
2.5 Accelerometer	8
2.5.1 Definition	8
2.5.2 Types of Accelerometers	8
2.5.3 Terminologies	9
2.5.4 Application of Accelerometer.....	10
2.5.5 Application of Sensors in Electrical Devices	11
3 MOBILE PROGRAMMING ENVIRONMENT.....	14
3.1 Introduction to Java.....	14
3.1.1 Derivation and New Characteristics of Java.....	14
3.1.2 Main Technology Values of Java	14
3.2 An Introduction of Java ME.....	16
3.2.1 Java ME Overview.....	16
3.2.2 Connected Device Configuration (CDC).....	17
3.2.3 Connected Limited Device Configuration (CLDC).....	18
3.2.4 Profiles	19
3.2.5 MIDP	19
3.3 JSR and API.....	21
3.4 MIDlet.....	22
4 MOBILE SENSOR API AND JSR 256.....	24
4.1 Introduction.....	24
4.2 Structure of the Sensor Package.....	24
4.3 Important Classes and Interfaces in the Package	25
4.3.1 Class SensorManager.....	26
4.3.2 Interface SensorConnection.....	26

4.4 Sensor Definition of an Accelerometer.....	27
4.4.1 Basic Parameters of an Accelerometer	28
4.4.2 Axes of a phone defined by an accelerometer	28
4.4.3 Necessary Configurations of the Accelerometer	29
5 DESIGN MY OWN PROGRAMME	31
5.1 Build the Programming Environment.....	31
5.2 Create a Project	33
5.3 Install the Nokia PC Suite.....	37
5.4 Nokia Remote Device Access.....	38
5.5 Function of My Own Application.....	39
5.6 Debugging and Troubleshooting.....	42
5.6.1 Sequential Switching of the Menu.....	42
5.6.2 Choosing a compatible device	44
5.6.3 Finding the sensor in the real device	45
6 CONCLUSION	48
6.1 Aim of the Study	48
6.2 Future Prospect of JSR 256.....	48
REFERENCES	50
APPENDICES.....	52
Appendix 1: Sensor MIDlet code.....	52

1 INTRODUCTION

Mobile phone is becoming a more and more indispensable tool in our daily life. Its main and only function was to make and receive phone calls when it was just invented. However, with the development of communication technology and other related technologies, a lot of new functions are integrated into this device. People can now use their mobile phones of the latest versions to take photographs, surf the internet, locate himself/herself with the GPS or play games based on Java, or even edit their PowerPoint slides for the presentation. Not only more practically useful functions are added into the mobile phone, but also it has changed the appearance from the big brick-like block to the cute and delicate device filled within one's hand. Nowadays, you may even judge some as a fashionable and cool guy by his florid and multi-function smart phone among youngsters.

Besides, the new functions do provide people with more conveniences. GPS, for instance, becomes critically important for those who drive out for a trip. It enables them to be aware of their exact positions and directions in order to avoid getting lost. Also, the embedded camera into the mobile phones gives tourists an alternative in case their cameras run out of battery.

In the meantime, the user interfaces of the mobile phone keeps updating. A touchable screen has partly replaced the traditional keyboards, or even completely like for example in iPhone. The aim of my thesis is also to analyze the new possibilities of the user interface in mobile phones. In the thesis I design a graphical user interface which uses the embedded sensors like the accelerator sensor and the rotation sensor, to control and interact with the device. The Sensor API (JSR256) is necessary for implementing this scheme. This package contains the classes and methods needed in calling the sensor and obtaining its output value.

The structure of my thesis is as the following.

Chapter 2 will provide a view of the application of the sensor technology. For instance, Wii is a play station mostly based on its sensor controller. There have been lots of games based on the sensor embedded in mobile phones as well.

In Chapter 3, I will give a comprehensive view of the programming environment I am using, ranging from the programming language to the Nokia PC suite. Initially, I would like to explain Java, the programming language I use. Then I will go through the related issues concerning how to build an entire programming environment.

In Chapter 4, the Sensor API is inspected in details. This part being the critical section of the project, I will focus more on this section. Elaborated specifications will be made on the classes and methods needed for calling the sensor and obtaining its output value.

In Chapter 5, a step-by-step guidance to build the proper programming environment is illustrated. Further, I will explain how to perform some operations by merely moving the device. I will also introduce the main idea of my code and explain the core part of it.

Chapter 6 will conclude the thesis. Here, I will discuss the problems which I faced and the trouble-shooting process designing this application. Further, I will provide some prospect of the sensor technology and the future development related to it.

2 SENSOR AND SENSOR TECHNOLOGY

2.1 Definition of Sensor

One of the most important tools in this final thesis is the accelerometer. Thus it is necessary for us to investigate these issues beforehand: what is a sensor, how it is used and what the types are.

The definition of a sensor says it is an instrument which is used to measure a physical quantity and then transform it into a readable signal by an observer. [1]

In other words, sensors are those equipment that respond to as well as receive a stimulus or signal. The word stimulus stands for a kind of property to be transformed into an electrical form. Therefore, a sensor can be described as an instrument that transforms a given signal into the form of electricity that could be processed for electronic devices. A transducer can be distinguished from a sensor because the transducer switches the forms of various kinds of energy while a sensor merely transforms the detected stimulus into the form of electricity.

Here is an example of a real sensor that we are all familiar with, which is a temperature-sensitive sensor. It transforms the measured temperature into extension and compression of the liquid-formed mercury. The following Figure 2.1 shows a mercury-in-glass thermometer.



Figure 2.1: A mercury-in-glass thermometer [2]

There is another example of sensor which is a thermocouple. The ambient temperature is transformed into an electrical voltage that is readable with a voltmeter.

Common standards are used to calibrate those sensors in the sense of accuracy.

Figure 2.2 and 2.3 show respectively a thermocouple and a voltmeter.



Figure 2.2: A thermocouple [3]



Figure 2.3: A voltagemeter [4]

Above are merely two instances of sensors. In real life, numerous kinds of sensors are implemented and applied in the industry and our daily life. The next section will give a view of their various usages.

2.2 Usage of Sensors

Think about the buttons of an elevator and why they brighten when pressed, and those sound-controlled lamps in the corridor of your apartment building. You may never be aware of how many applications of sensors there are in our daily life. The applications have been widely implemented in the industries as well, for instance in manufacturing, robotics, and especially in the field of aerospace.



Figure 2.4: A touch-sensitive elevator button [5]

2.3 Sensitivity of Sensors

The sensitivity of a sensor shows the slope of the sensor between its change of output and the change of the property. To take this for example, the sensitivity is $1 \text{ cm}/^\circ\text{C}$ if the liquid inside a thermometer goes up 1 cm when the temperature increases 1°C . [6] Therefore, a sensor with high sensitivity is indispensable if you measure small changes.

2.4 Types of Sensors

Generally, sensors are classified into fifteen major categories, which are [7]:

- 1 Acoustic, sound, vibration
- 2 Automotive, transportation
- 3 Chemical
- 4 Electric current, electric potential, magnetic, radio
- 5 Environment, weather, humidity
- 6 Flow, fluid velocity
- 7 Ionising radiation, subatomic particles
- 8 Navigation instruments

- 9 Position, angle, displacement, distance, speed, acceleration
- 10 Optical, light, imaging
- 11 Pressure
- 12 Force, density, level
- 13 Thermal, heat, temperature
- 14 Proximity, presence
- 15 Sensor technology

Furthermore, the major types are divided into various subtypes. The accelerometer, which is used in this thesis, belongs to the ninth major type, namely “Position, angle, displacement, distance, speed, acceleration”.

This major type contains a couple of subtypes, which are listed below:

- Accelerometer
- Capacitive displacement sensor
- Free fall sensor
- Inclinometer
- Laser rangefinder
- Linear encoder
- Linear variable differential transformer (LVDT)
- Liquid capacitive inclinometers
- Odometer
- Piezoelectric accelerometer
- Position sensor
- Rotary encoder
- Rotary variable differential transformer
- Selsyn
- Sudden Motion Sensor

2.5 Accelerometer

2.5.1 Definition

An accelerometer is the equipment which measures acceleration forces. The forces can be either static or dynamic. The static, for example, is like the constant force of gravity pulling at your feet whereas the dynamic may be caused by moving or vibrating the accelerometer. [8]

Both single-axis and multi-axis modes can be used to detect magnitude and direction of the acceleration. It has now been popular to embed accelerometers into portable devices as mobile phone and game controllers like Wii, in order to perceive the position of the equipment or act as some kind of game input.

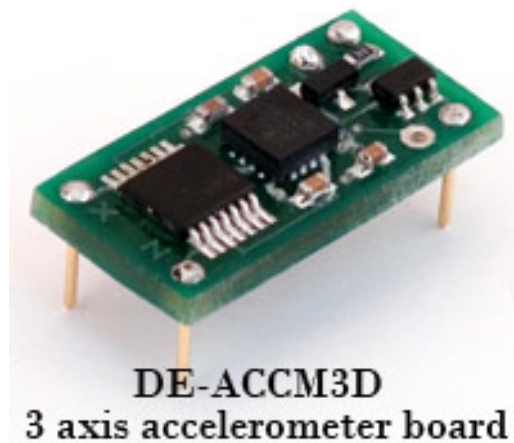


Figure 2.5 A three-axis accelerometer board [9]

2.5.2 Types of Accelerometers

There are different types of accelerometers as well. They are classified according to the key technologies in use. A brief classification with the key technologies of each kind is listed below. [10]

Capacitive-Metal

Key-tech: beam or micromachined feature produces capacitance; change in capacitance related to acceleration

Piezoelectric

Key-tech: Piezoelectric crystal mounted to mass –voltage output transformed to acceleration

Piezoresistive

Key-tech: Beam or micromachined feature whose resistance changes with acceleration

Hall Effect

Key-tech: Motion transformed to electrical signal by sensing of changing magnetic fields

Magnetoresistive

Key-tech: Material resistivity changes in presence of magnetic field

Heat Transfer

Key-tech: Location of heated mass tracked during acceleration by sensing temperature

2.5.3 Terminology

Some of the main terms and concepts related to accelerometers are listed below.

Sensitivity: A parameter indicating how much the output value is influenced according to the changes of the acceleration. The unit is Volts/g. Figure 2.6 illustrates the sensitivity and linearity of an accelerometer.

V_{cc}: The working voltage of the sensor, typically –5V for the devices.

%V_{cc}: The result is represented as the output value dividing the supply voltage. This standardized the result regardless of the various supply voltage between readings.

Ratiometric: The output value of the sensor variation according to a difference in the input voltage.

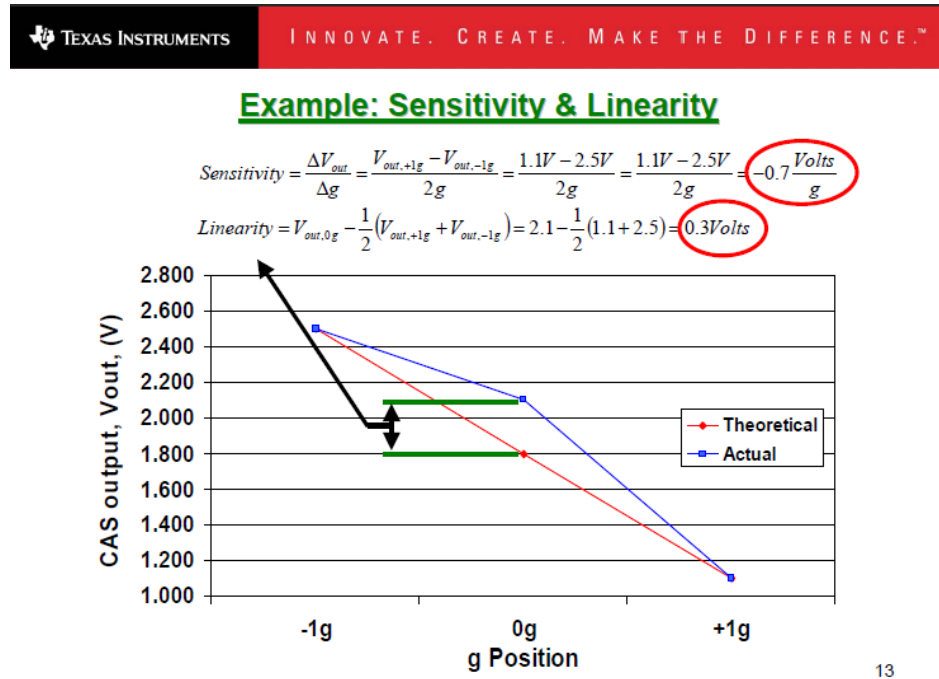


Figure 2.6: An example of sensitivity and linearity of an accelerometer

2.5.4 Application of Accelerometer

The angle of the device can be calculated with the measuring result of the magnitude of the static acceleration caused by gravity. The style of movement could be investigated according to the output value of dynamic acceleration.

There are a couple of situations where an accelerometer can help you, such as when you are driving up to a hill, or falling over the hill. What is more, you can judge whether the plane is flying horizontally or vertically. Given the output value of an accelerometer, the software developer is able to answer those questions above with ease. It is also useful in analyzing problems in a car engine with vibration testing. To summarize, the accelerometer makes you more aware of the surrounding objects and environment. [11]

Accelerometers have been adopted to shield hard disks from being damaged by companies. For instance, with the specific type of a laptop made from IBM, the

accelerometer will notify the device about the accidental freefall, therefore the laptop can turn off the hard drive to prevent the heads from crashing the platters. Similarly, the accelerometers are adopted in industry to detect car crashes and automatically deploy airbags.

2.5.5 The Application of Sensors in Electrical Devices

Accelerometers are increasingly being incorporated into personal electronic devices. They are used for acting as, for instance, motion input, orientation sensing and device integrity and so on.

2.5.5.1 Motion Input

Accelerometers are embedded into such devices as smartphones and PDAs to control the user interface. Apple iPhone and iPod touch can serve as examples for this kind of applications. [12]

Figure 2.7 illustrates how iPhone implements the embedded accelerometer to control the user interface. When you change the placement of the iPhone from horizontal to vertical, the display makes a corresponding change as well with the signal generated from the accelerometer.



Figure 2.7: An example of iPhone using the accelerometer for UI control[13]

The Wii Remote has a three-axis accelerometer and was used primarily for motion input. It is the controller for Nintendo's Wii video game. Figure 2.8 shows a Wii remote.



Figure 2.8: Wii Remote [14]

The Wii Remote has the ability to sense acceleration along three axes through the use of an ADXL330 accelerometer. The Wii Remote also features a PixArt optical sensor, allowing it to determine where the Wii Remote is pointing.

The ADXL330 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs, all on a single monolithic IC. The product measures acceleration with a minimum full-scale range of ± 3 g. It can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration. [15]

A six-axis accelerometer is implemented to make steering more lifelike in the games. It is achieved by the remote called DualShock 3 of the Sony PlayStation 3.

2.5.5.2 Orientation sensing

The feature to automatically change the direction of the display according to the physical orientation of the device is provided in a number of laptops. It is adopted in Tablet PC and digital cameras as well.

For instance, iPhone and iPod Touch both allow the device itself to know when it is tilted with the information provided by the embedded accelerometer in the device.

The Nokia N95 and N82 are also equipped with accelerometers inside. It mainly functions as a tilt sensor to tag the orientation to the photos.

2.5.5.3 Device integrity

Laptops use accelerometers to protect themselves from damage. For example Lenovo's Active Protection System detects drops with the sensor inside. The accelerometer informs the heads of the hard disk to be stopped if it detects the fall of the device, in order to prevent loss of data caused by the shocks.

3 MOBILE PROGRAMMING ENVIRONMENT

3.1 Introduction to Java

3.1.1 Derivation and New Characteristics of Java

Java is a programming language originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. [16] The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.

Java is general-purpose, concurrent, class-based, and object-oriented, and it is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere". Java is considered by many as one of the most influential programming languages of the 20th century, and it is widely used from application software to web applications.

3.1.2 Main Technology Values of Java

Richer User Experience - Whether you're using a Java technology-powered mobile phone to play a game or to access your company's network, the Java platform provides the foundation for true mobility. The unique blend of mobility and security in Java technology makes it the ideal development and deployment vehicle for mobile and wireless solutions.

The Ideal Execution Environment for Web Services - The Java and XML languages are the two most extensible and widely accepted computing languages, providing maximum reach to everyone, everywhere, every time, to every device and platform.

[17]

Enabling Business from End to End - Java technology offers a single, unifying programming model that can connect all elements of a business infrastructure.

One of the excellent qualities of Java is the extendibility which enables it to be applied in almost every situation. There are different Java platforms according to various purposes of development for programmers, providing the developers with the appropriate solutions they need. Figure 3.1 exhibits the various Java platforms with their areas of use.

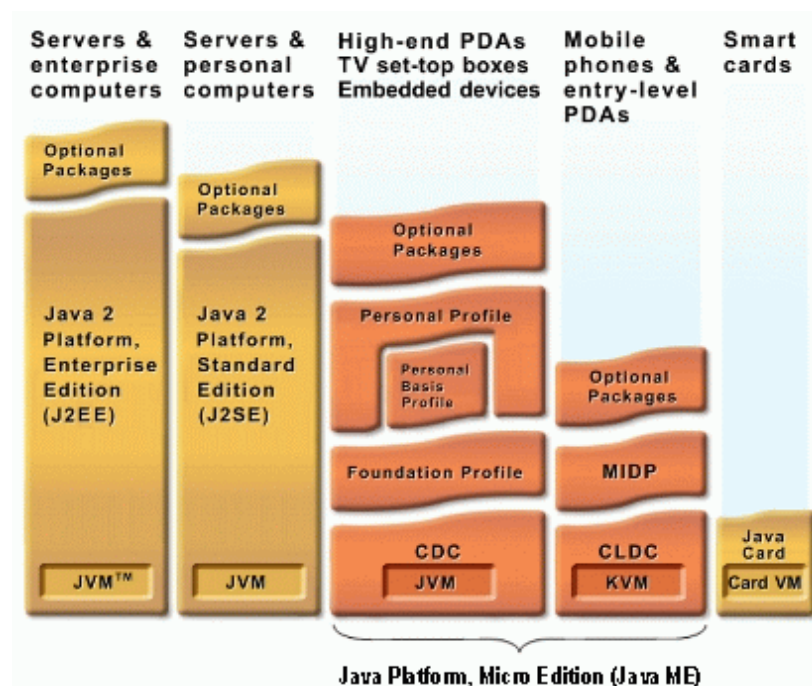


Figure 3.1: Various Java platforms with their areas of application [18]

For example, the platforms like Java Mobility, Java ME and Java Card Technology are meant for the developers in small and mobile devices.

The Java ME Platform meets the demands of programmers to create applications for the consumers. It supplies applications for quite a number of types of devices, including wireless and wired devices, mobile phones and even set-top boxes. [19]

Also, the platforms such as Desktop Java Technology and Java SE provide solutions

for those who want to program in PC desktops.

The Java SE platform is mainly designed for the desktop computing platforms, for example Linux, Microsoft Windows and Sun Solaris. Compatible desktops, especially in heterogeneous environments, represent a boost in user productivity, communication, and collaboration, as well as considerable cost-of-ownership savings.

Lastly, the Java EE platform, which is short for the enterprise edition of the Java platform, is implemented for medium to large sized businesses. It is quite necessary for companies with great deployment needs. There have been over five million downloads of Java technology for the Enterprise for the advantages that apply across virtually all industries and application types.

This thesis deals mostly with the Java ME. Therefore I will bring more details in the next section about this platform of Java.

3.2 Introduction of Java ME

3.2.1 Java ME Overview

Sun Company originally designed the Java ME technology to cope with the limited resources available for those small devices such as mobile phones and set-top boxes. The resources on the small devices are always limited, for example, the limited power supply, size of the screen and memory. According to these restrictions, Sun Company defined the foundations for Java ME.

A complete Java runtime environment is built according to the specific requirements or restrictions of a kind of given device. It is constructed based on those technologies and specifications of Java ME platform. This offers the possibility for developers to cooperate with each other conveniently, therefore to provide good user experiences.

The three elements consist of the most basic foundations of Java ME technology.

Firstly it comes with the configuration, which is the most basic part supporting the broadest range of devices. Then, a profile is meant for a more limited range of devices. Lastly, an optional package is used in implementing technology-specific areas.

Java ME platform has been categorized into two kinds of major configurations which are CLDC and CDC. The CLDC, short for Connected Limited Device Configuration, is meant to fit in small mobile devices, whereas the CDC short for Connected Device Configuration is implemented for a wider range of devices like smart-phones.

In the next section, I will explain these two types of configurations in detail, and compare their differences and relationship with each other.

3.2.2 Connected Device Configuration (CDC)

The Connected Device Profile, known as CDC, is employed in developing applications for large devices, which are usually equipped with more system resources and internet access. [20] The core purpose of CDC is to both provide developing tools as well as support various feature sets of a wide range of devices under the restrained condition of resources. Figure 3.3 shows the structure of a digital media platform based on CDC.

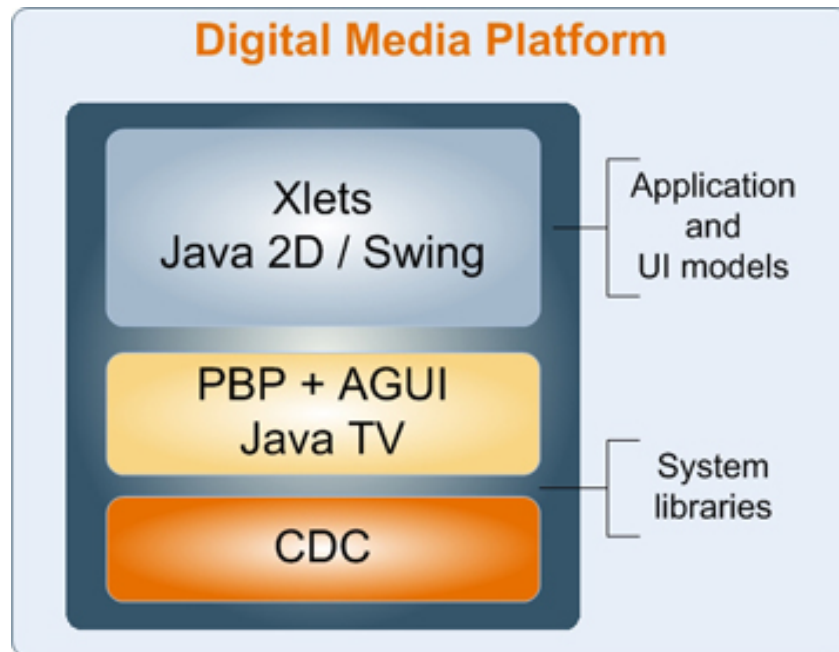


Figure 3.2: The structure of a digital media platform based on CDC. [21]

3.2.3 Connected Limited Device Configuration (CLDC)

The Connected Limited Device Configuration, known as CLDC, is designed mainly for mobile phones and those similar devices with limited resources. In other words, it helps to build an available Java run time environment given restricted conditions, such as limited memory, computing capability and screen size. In addition, Java ME platform defines lots of profiles that are helpful for building the application above the configurations. Usually, the Mobile Information Device Profile (MIDP) is combined together with CLDC to create Java run time environment for mobile devices and other similar ones.

Figure 3.2 illustrates the structure of a CLDC wireless platform based on CLDC.

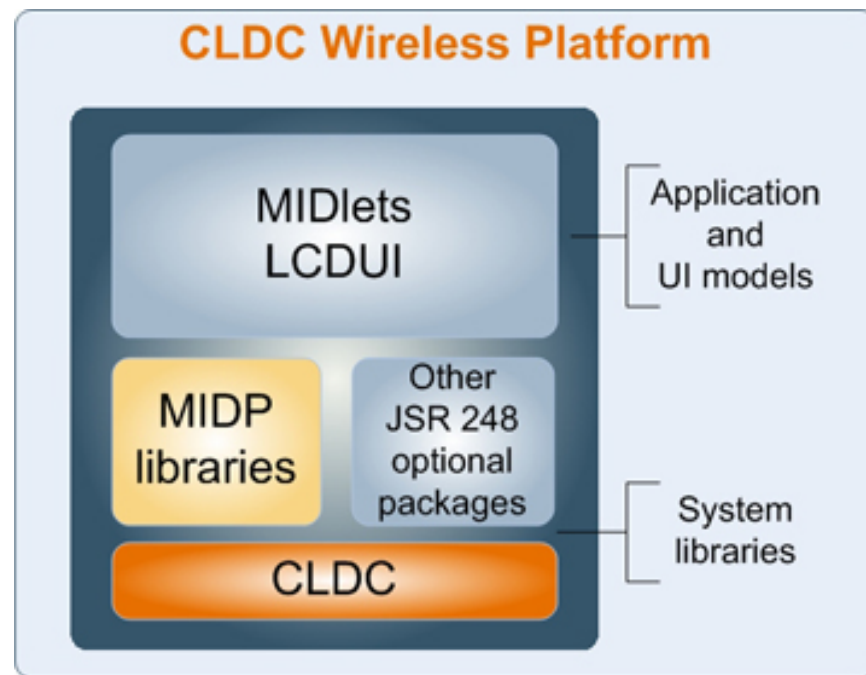


Figure 3.3 The structure of a CLDC wireless platform based on CLDC [22]

3.2.4 Profiles

Under a given configuration like CLDC, a profile is needed to provide support a series of similar types of mobile devices. It contains a set of APIs, which are short for Application Programming Interface. For instance, MIDP is a kind of profile I use in this project. Together with CLDC, MIDP builds a complete Java application environment for the target device groups.

3.2.5 MIDP

3.2.5.1 MIDP Overview

As mentioned above, MIDP is the necessary element of the J2ME platform. It provides the programmer with useful applications and interfaces that can be conveniently called in their own projects. In the form of APIs, MIDP provides developers with the capability to access specific hardware of the device without having the source code or understanding the details of the mechanism inside. Then

what benefits does the MIDP actually provide us?

Firstly, MIDP helps us in building graphical applications. It optimizes the small display size and takes the full advantage phone keypads. The touch screens, extra keys and small QWERTY keyboards are all on the basis of MIDP.

Then, it supports various kinds of connectivity standards in order to enhance the capability of the device in connecting with an outside network. The technique standards include serial port, server sockets, HTTP and datagrams.

As for the multimedia and game functionality, MIDP plays an important role as well. It is a perfect foundation on which developers could build games and multimedia applications for their cell phones. By implementing a low-level user-interface API, MIDP makes more hardware resources of the device accessible, thus leading to a greater control for developers. If combined with the game API, MIDP can make full use of the graphic capabilities of the device. There is also a Mobile Media API optional package available, which you may utilize to add the multimedia content if you need.

Another advantage of MIDP is the ability of providing you with the latest updates of your applications on the air.

3.2.5.2 MIDP versions

Two versions of MIDP exist, while the newer version MIDP 2.0 (JSR 118) is backward-compatible with MIDP 1.0 (JSR 37). [23]

The original version, MIDP 1.0, provides the most basic functionalities such as the user interface and network connectivity. The newer one, MIDP 2.0, has added some new features such as an enhanced UI, multimedia and game functionality and more extensive connectivity, over-the-air provisioning (OTA), and end-to-end security.

Both versions target mobile information devices like mobile phones and PDAs.

3.3 JSR and API

As you may have noticed , MIDP 2.0 is also known as JSR 118. Actually, CLDC and CDC are also different JSRs distinguished by their versions. Moreover, we know that the Location API is included in the JSR 179. Then it comes to the questions: What is JSR? What is API? How about the relationship between them?

In this section, I will focus on the answers to these questions and explain how to apply them in the J2ME programming environment.

Generally, JSR is short for Java Specification Request. It is a term related to Java Community Process (JCP). JCP is an organization aimed at evolving the Java platform and it is responsible for all the development of Java technology.

An Application Programming Interface (API) is an interface implemented by a software program to enable its interaction with other software. It is similar to the way the user interface facilitates interaction between humans and computers.

When the need for a new component or API is identified the initiator creates a Java Specification Request (JSR) and sends it to the community. An expert group is formed with representatives from the participating companies with the task to create the specification. The JSR passes through the JCP and if approved the specification will be released to the community for implementation.

The relationship between them can be briefly stated that the JSR is to develop a specification for the feature and API sets for the next feature release of Java platform.

Figure 3.4 shows the structure of the JSRs and their applied fields

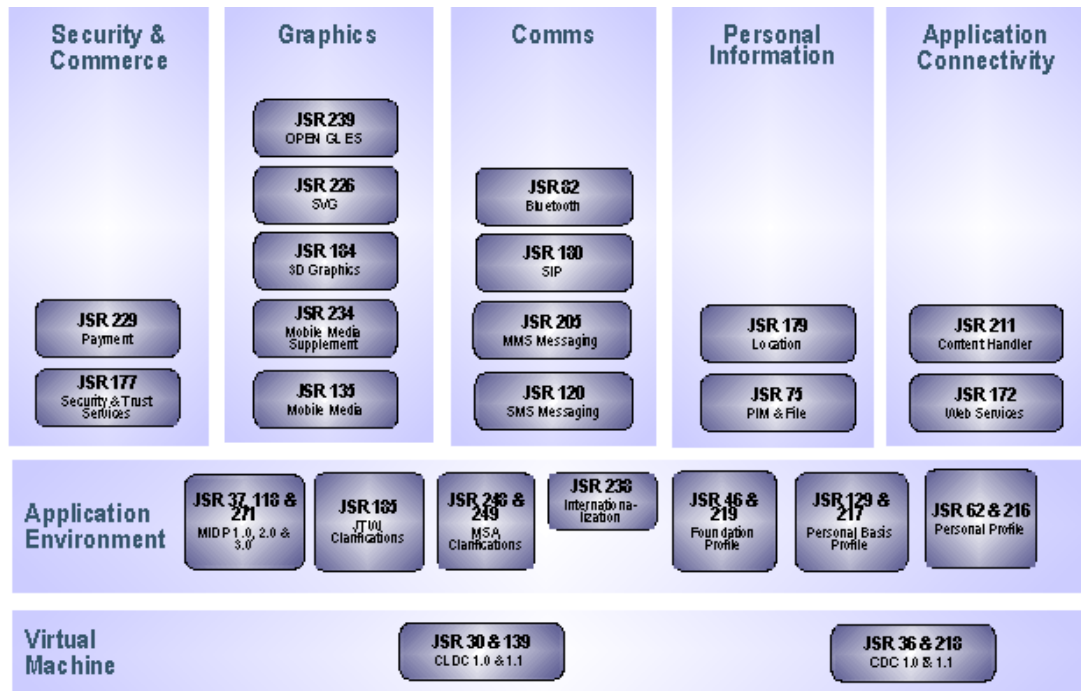


Figure 3.4: The structure of the JSRs and their fields of application [24]

3.4 MIDlet

A MIDlet is a Java application designed to be run on a mobile device. The application I created in this thesis is actually a MIDlet.

A MIDlet typically has three states, including Paused state, Active state, and Destroyed state. The Application Management Software (AMS) is in charge of the lifecycle, in which the MIDlet switches its state.

Figure 3.5 illustrates the lifecycle of a MIDlet.

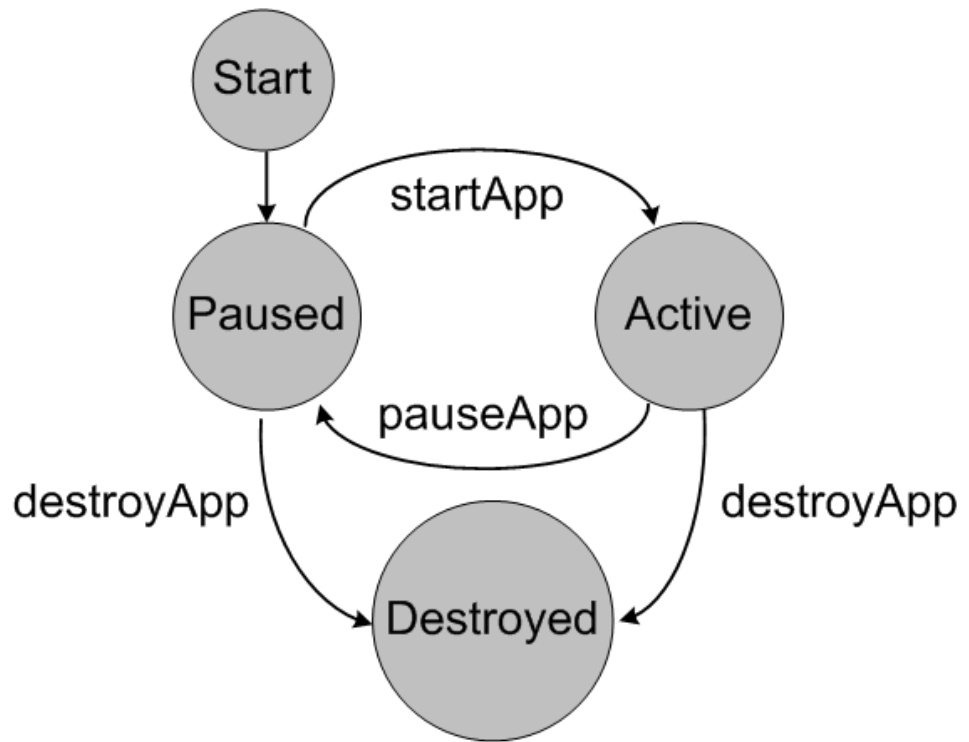


Figure 3.5: The lifecycle of a MIDlet. [25]

4 MOBILE SENSOR API AND JSR 256

4.1 Introduction

The Mobile Sensor API allows Java ME application developers to fetch data easily and uniformly from sensors. A sensor is any measurement data source. Sensors vary from physical sensors such as magnetometers and accelerometers to virtual sensors, which combine and manipulate the data they have received from other kinds of physical sensors.

The API also provides means to monitor measured data. The application can register a listener, and set limits and ranges for monitoring. If the measured value meets any of defined conditions, the listener is notified.

4.2 Structure of the Sensor Package

In this section I will briefly introduce the main interfaces and classes, and illustrate how these components are combined to work together. The content of this section is mainly based on the documentation of Java Mobile Sensor API specification 1.1.

Firstly let us have a look at the structure diagram of the `javax.microedition.sensor` package shown in Figure 4.1.

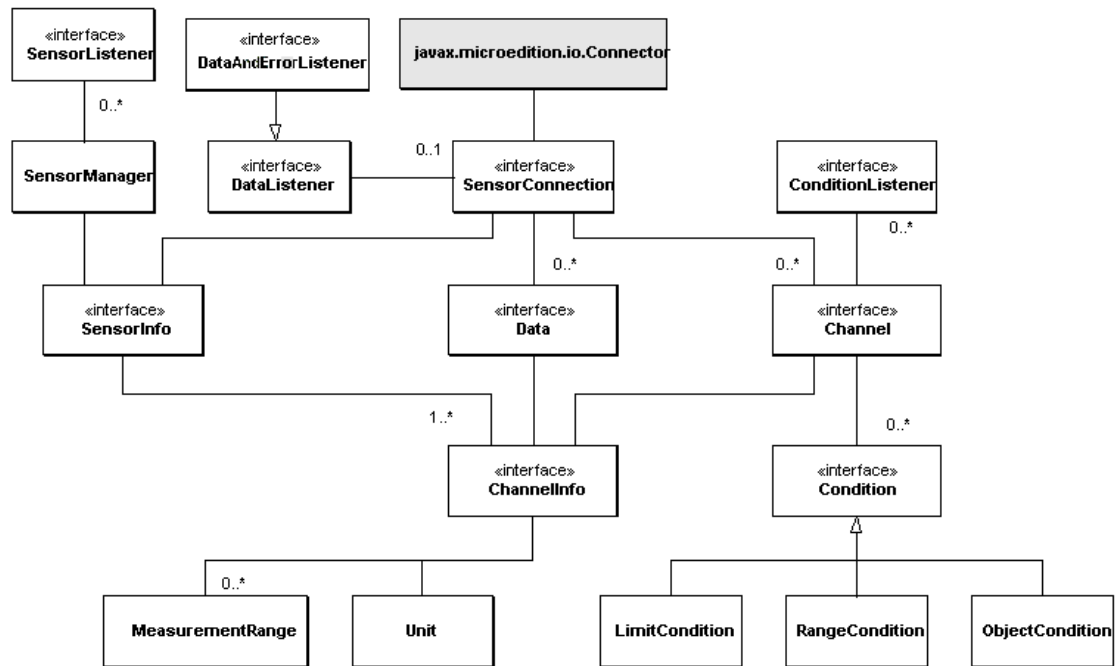


Figure 4.1: Class diagram of javax.microedition.sensor package [26]

We can observe that the interface `SensorConnection` is the core of this package. The source is `SensorManager` known as a sensor finder. The method `findSensors()` of `SensorManager` returns an array of matching `SensorInfo` objects, which contains the information of sensor properties.

By the sensor URL a connection is established between the class `SensorInfo()` and `SensorConnection()`. The `Connector.open()` method provides an active connection to the sensor.

The `SensorConnection()` class is also connected with the class `Data` and class `Channel`. Instance can be used to receive the measurements. Channels represent different dimensions of the measurement. For example, in 3D acceleration there are three dimensions which are axis x, y, and z; these are different channels.

4.3 Important Classes and Interfaces in the Package

4.3.1 Class SensorManager

The SensorManager class is used for finding sensors and monitoring their availability. SensorManager provides two static methods to find sensors. They both return an array of SensorInfo objects listing the found sensors:

1. findSensors(quantity, contextType)
2. findSensors(URL)

The method I used in my application is the former one. I would like to give a specification of this method.

The synopsis of the method is

```
public static SensorInfo[] findSensors(java.lang.String quantity, java.lang.String contextType);
```

The first parameter, quantity, defines the desired sensor. Its values can be such as “acceleration”, “rotation” or “temperature”. The second parameter, contextType, defines the context type qualifying the desired sensor. Valid values are:

1. CONTEXT_TYPE_AMBIENT
2. CONTEXT_TYPE_DEVICE
3. CONTEXT_TYPE_USER
4. CONTEXT_TYPE_VEHICLE

The method returns an array of SensorInfo objects of all supported sensors, with the given quantity and context type, or a zero-length SensorInfo array if no match is found.

4.3.2 Interface SensorConnection

The `SensorConnection` is an abstraction of an actual sensor. It provides the functionality to retrieve data from a sensor. A sensor can be widely understood as any measurement data source. It provides a direct output to the physical stimuli such as heat, light, sound, pressure, magnetism, or motion. Alternatively, a sensor can be a virtual data source that supplies data which is collected from various sources and manipulated.

Two modes coexist to retrieve data, including a synchronous mode and an asynchronous mode. Both synchronous and asynchronous retrieval deliver the data encapsulated in `Data` objects.

In the synchronous mode the data retrieval is implemented by calling `getData(int, long, boolean, boolean, boolean)` methods.

In the asynchronous mode, the `DataListener` interface is needed. When the application implements this interface and registers itself with the `setDataListener()` method, it starts to receive `DataListener.dataReceived()` notifications.

I used the asynchronous mode to receive the data in my application. Therefore I needed to call `setDataListener()` method to register a `DataListener`.

The synopsis of the method `setDataListener` is :

```
public void setDataListener (javax.microedition.sensor.DataListener listener,int  
bufferSize);
```

The first parameter, `listener`, indicates the `DataListener` to be registered. The next parameter, `bufferSize`, is the size of the buffer. Valid values must be bigger than 0.

4.4 Sensor Definition of an Accelerometer

4.4.1 Basic Parameters of an Accelerometer

The basic parameters of an accelerometer are important in the measurement. They include such parameters as channel name, data type, scale, measurement range, resolution and symbol of the unit. Figure 4.2 demonstrates the parameter values with the measurement range $[-2g, 2g]$

Channel name	Data type	Scale	MeasurementRange	Resolution	Symbol of the unit
axis_x	double	0	$[-19.61, +19.61]$	8-bit/12-bit	m/s^2
axis_y	double	0	$[-19.61, +19.61]$	8-bit/12-bit	m/s^2
axis_z	double	0	$[-19.61, +19.61]$	8-bit/12-bit	m/s^2

Figure 4.2: The parameter values of an accelerometer [27]

4.4.2 Axes of a phone defined by an accelerometer

The main plane of the phone is defined by the x axis together with the y axis. The z axis is vertical to the main plane. The direction of -x is to the left from the phone and +x is to the right. -y is down, +y is up. -z is the direction away from the user whereas +z goes towards the user. Figure 4.3 exhibits a mono-display device and its co-ordinate axes.

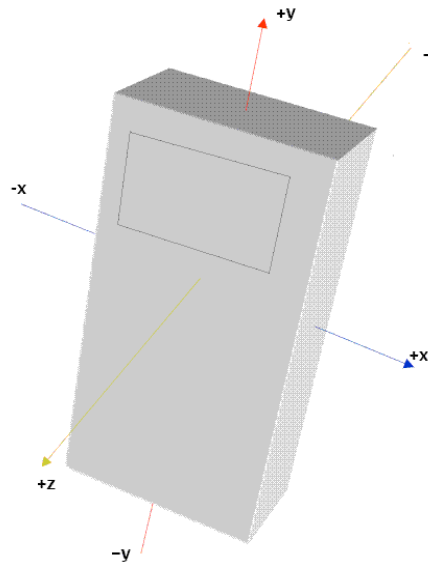


Figure 4.3: Mono-display device and its co-ordinate axes

4.4.3 Necessary Configurations of the Accelerometer

There are totally three channels within an accelerometer. Each channel consists of several basic fields with default values such as the name, accuracy, data type, measurement range, resolution and unit. Figure 4.4 gives an example of the ChannelInfo information of axis_y. The information of axis_x and axis_z is quite much alike.

Field	Value
Name	"axis_y"
Accuracy	0.025
Data type	ChannelInfo.TYPE_DOUBLE
Measurement range	[-19.61,19.61],[-58.84,58.84]
Resolution	0.01 in [-19.61,19.61]; 0.003 in [-58.84,58.84]
Unit	"m/s^2"

Figure 4.4: ChannelInfo information of axis_y

5 DESIGNING THE SENSOR CONTROLLED USER INTERFACE

5.1 Building the Programming Environment

In my application, I used the NetBeans IDE 6.8 as the programming environment. It can be downloaded on the website www.NetBeans.org without any charges. I chose the Chinese language as the preferred language since it is more convenient for me.

Mobility Java Development Kit (JDK) is also necessary for J2ME programming. It is related to the process of Java compiler and application, enabling us to build the Java Archive Files (jar) automatically.

Also, I need to have the Java SDK 3.0 simulation platform. This platform contains all the APIs I needed, such as the Mobile Sensor API (JSR 256). On this platform, I could simulate the mobile phone and test my application before I downloaded it to the real device. The platform could be downloaded from the Sun developer network.

Now I describe to the installation of the Java SDK 3.0. Firstly I entered the Java platform manager in the Netbeans 6.8. The platform manager is shown in Figure 5.1.

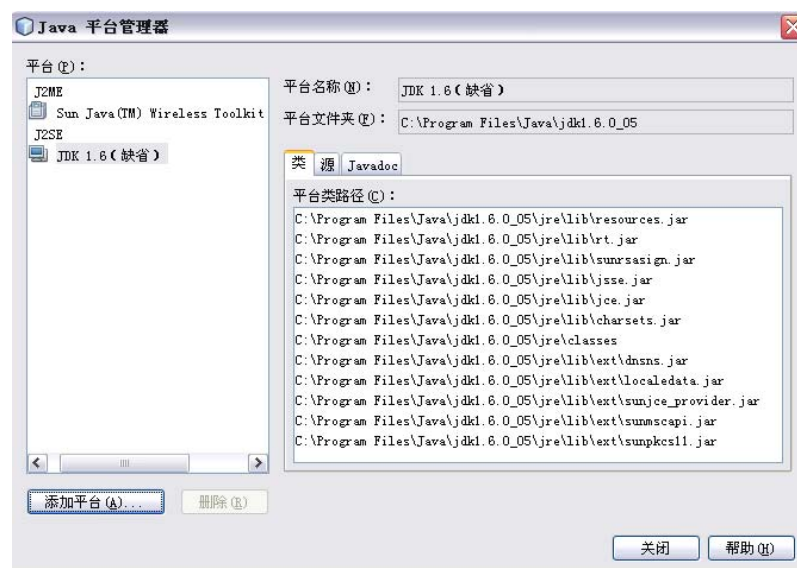


Figure 5.1: Java platform manager

Then I pressed “Add platform”. The window suggested me to select the type of platform to be installed, as is shown in Figure 5.2.



Figure 5.2: Choose the type of the platform

Then I chose the Java ME MIDP platform simulator. Subsequently, I browsed to find the Java ME platform SDK 3.0 and selected it to install. Figure 5.3 shows the process of installing Java SDK 3.0.

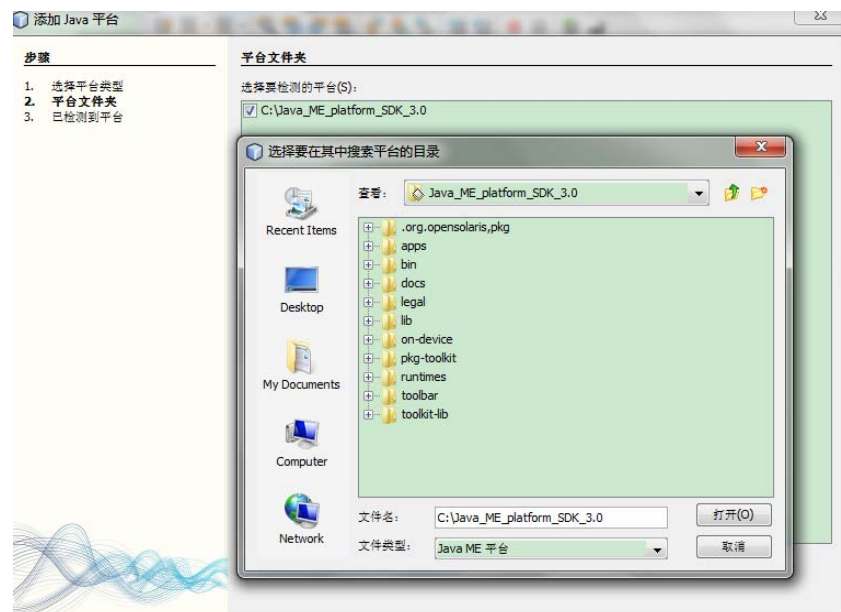
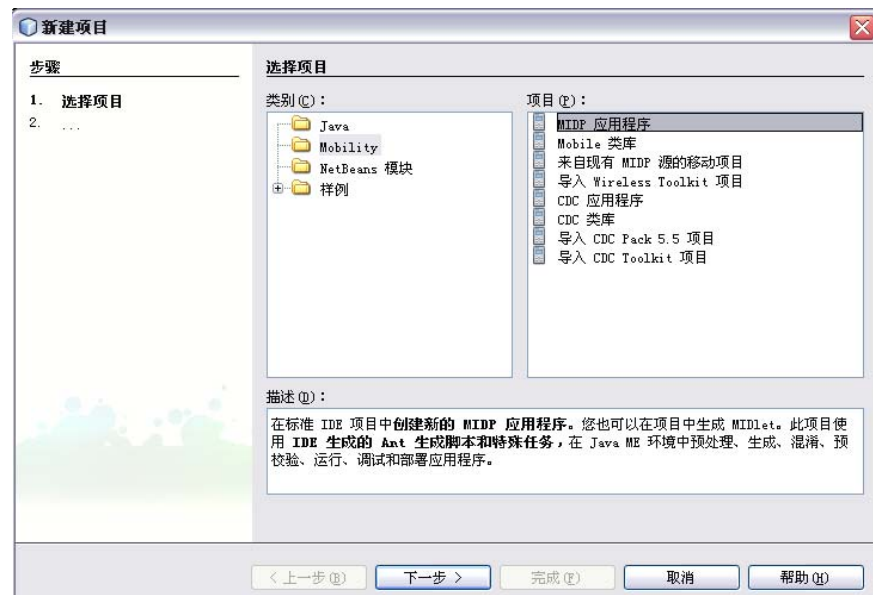


Figure 5.3 Installing Java SDK 3.0

Then I just followed the instructions. After the installation of Java ME platform SDK, the entire programming and simulation environment was built successfully.

5.2 Creating a Project

To create a new project is the first step. You can select the File button in Netbeans, and follow the sequence of New Project->Mobility->MIDP application. Figure 5.4 illustrates how to perform that.

**Figure 5.4:** Create a new project of MIDP application

Then I chose the emulator carefully and set the configurations. The device configuration should be CLDC-1.1 and the device configuration profile should be MIDP-2.0. The configurations should be set correctly; otherwise there may be errors of incompatibility in the later process. Figure 5.5 demonstrates the setting of device configuration and device configuration profile.



Figure 5.5: The setting of device configuration and device configuration profile

At last, we could choose to create a Hello MIDlet, as our first program to test the platform. The source code is listed below.

Code:

```
package hello;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloMIDlet extends MIDlet implements CommandListener {
    private boolean midletPaused = false;

    public Display getDisplay ()
    {
        return Display.getDisplay(this);
    }

    /* Exits MIDlet. */

    public void exitMIDlet() {
```

```
switchDisplayable (null, null);
destroyApp(true);
notifyDestroyed();
}

/*Called when MIDlet is started.Checks whether the MIDlet have been
already started and initialize/starts or resumes the MIDlet. */

public void startApp() {
if (midletPaused) {
resumeMIDlet ();
} else {
initialize ();
startMIDlet ();
}
midletPaused = false;
}

/*Called when MIDlet is paused.*/

public void pauseApp() {
midletPaused = true;
}

/*Called to signal the MIDlet to terminate. */

public void destroyApp(boolean unconditional) {
}
}
```

The Figure 5.6 shows the result of the Hello MIDlet.

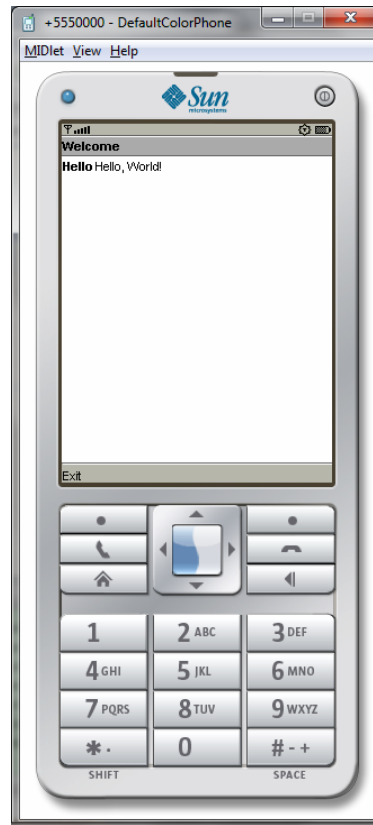


Figure 5.6: The running result of the Hello MIDlet

A window titled “SDK progress” will be popped up to the screen, as shown in Figure 5.7. It executes all the operations step by step, such as loading user preferences and creating Jad/Jar file. If done correctly, the signal before the specific operation turns green; otherwise it will turn a red cross.

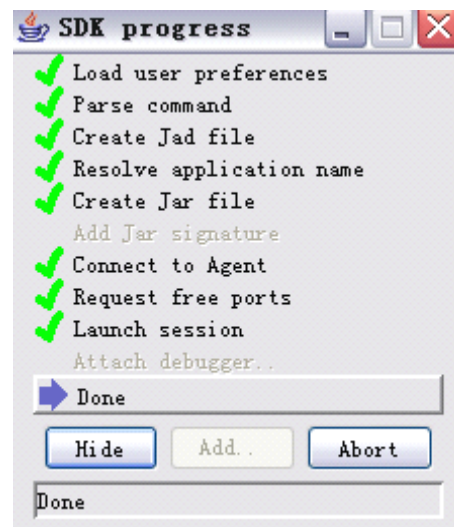


Figure 5.7: The SDK progress

5.3 Installing the Nokia PC Suite

Once we have successfully tested the application in the emulator, we should download it to the real device and observe the result. Generally, the jar file is copied into the device and installed to run.

There are several ways of performing this operation. Depending on the hardware of the device, the file transmission can be done through Bluetooth, infrared (IR) link, over the air (OTA), Memory Card or a serial cable.

Here I chose to use the serial cable because Nokia PC suite is compatible with this connection. Installation can be suggested and done automatically using Nokia PC suite, therefore it is more reliable and convenient. The software is freely available from Nokia website.

We can see the operation from Figure 5.8. The Jar file is located in the Folder “dist” under the directories of the Netbeans project. The file is copied from the left to the right side, which stands for your Nokia device. After the installation, the MIDlet can run on the real device.

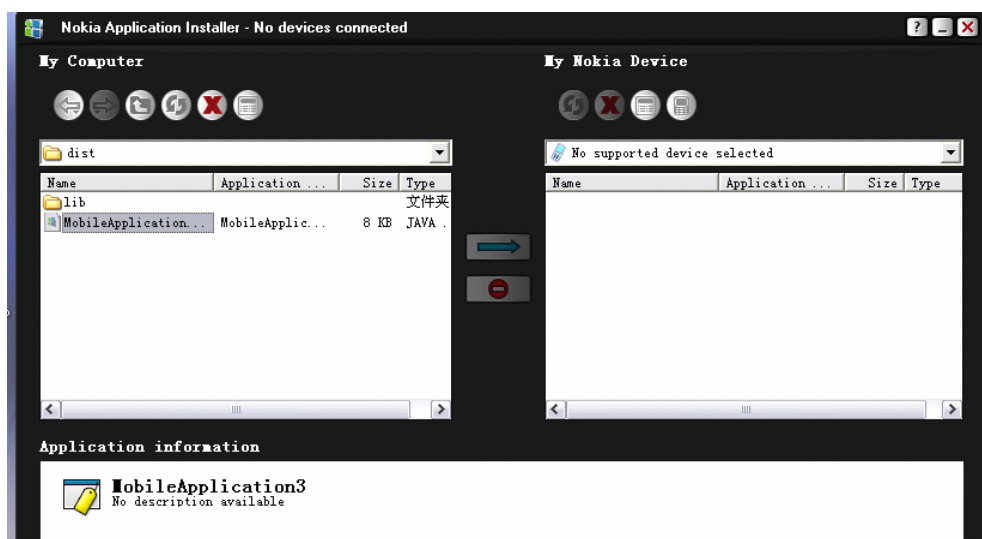
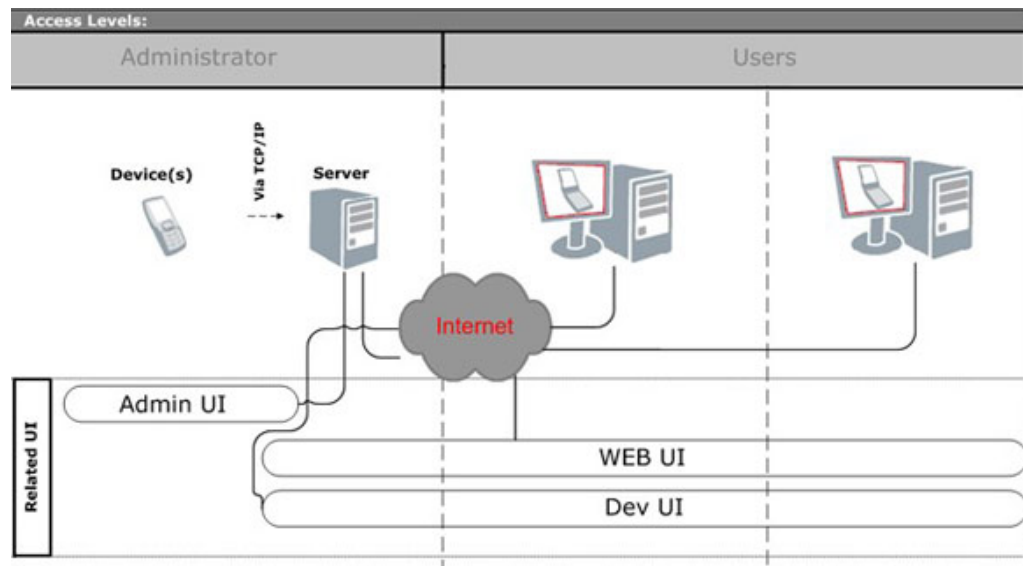


Figure 5.8: Nokia PC suite

5.4 Nokia Remote Device Access

Remote Device Access is a service provided by Nokia to remotely control the mobile devices. It allows developers to test their applications on a variety of Symbian based Nokia devices without possessing a real one. The service is free of charge and quite helpful. I used this RDA service to test my own application once the test was successful on the emulator. Figure 5.9 exhibits how RDA works. [28]

**Figure 5.9:** The working flowchart of RDA [29]

There are several exhilarating features included in this service, such as installing and running applications on the devices, file transfer client, debugging logs and changing screen orientation.

It can test Symbian, Java and Flash lite applications, Python, Open C, if the required plugin is installed first. Different types of content, for example themes and web technologies, for example Widgets and other types of web applications/pages.

5.5 Function of My Own Application

The application I created is a user interface, where users can sequentially switch the menu by shaking the mobile phone forwards and backwards or by pressing the buttons on the keyboard. The initial menu is shown below in Figure 5.10.

In this menu named Welcome, there are two buttons “Forward” and “Exit”. The Forward button leads to the menu Level1 and the Exit button ends this MIDlet. Plus, a fast movement forward changes the current menu to the menu Level1 as well.



Figure 5.10: The initial menu of my application

I opened the external events generator on the Java SDK platform and clicked the tab

Sensor. Here we can see two available sensors of the simulation device. The one with Id 0 is the sensor I used. I set the value of axisZ to -0.6m/s^2 to simulate a fast movement forward. For the value of the accelerometer on Z axis, -0.5m/s^2 is the threshold value for the device to switch to the menu of the next level while 0.5 m/s^2 is the threshold value to return to the upper level. In other words, if the value of axisZ w less than -0.5 , which implies that the device is making a swift move away from the user, the display would switch to the menu of the next level. The external events generator is shown in Figure 5.11.

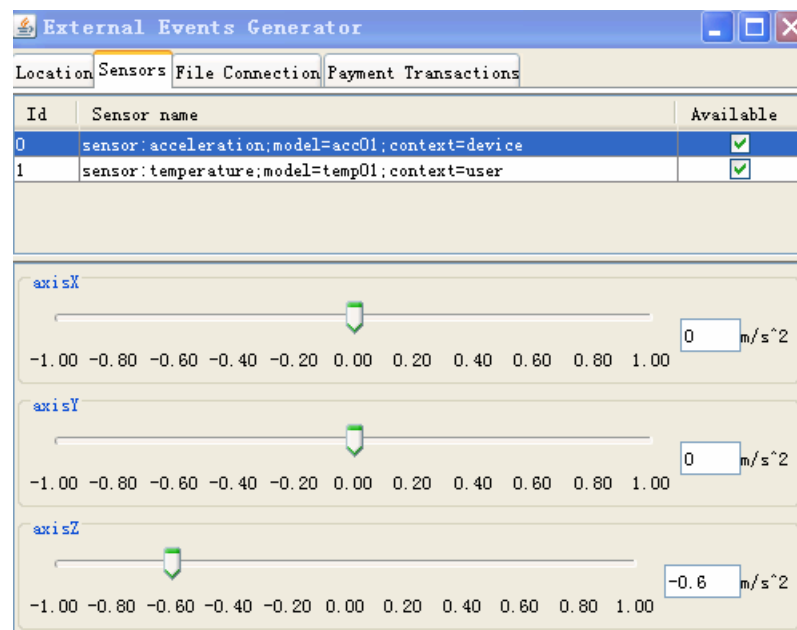


Figure 5.11: External events generator with axisZ value less than -0.5

Then I entered the menu Level1. I could either choose to move to the next level by moving the phone forward or pressing the Forward button, or go back to the original menu by shaking the device backwards or pressing the Back button. Figure 5.12 exhibits the screen of menu Level1.

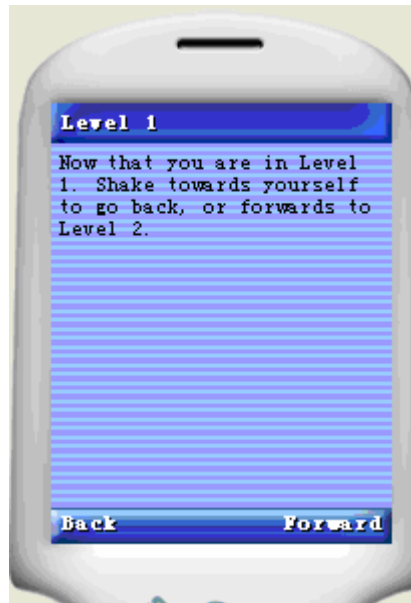


Figure 5.12: The screen of menu Level1

Firstly I tried to enter the next menu Level2. I set the value of axisZ to -0.6m/s^2 again in the external events generators and the screen turned into the appearance of Level2, as is shown in the Figure 5.13 below.



Figure 5.13: The screen of menu Level2

I wanted to return to the menu Level1. So I set the value of axisZ to 0.6 m/s^2 in the

external events generator shown in Figure 5.14. As mentioned earlier, a value higher than 0.5m/s^2 indicates a fast movement backwards. After this operation, the screen returned to the menu Level1.

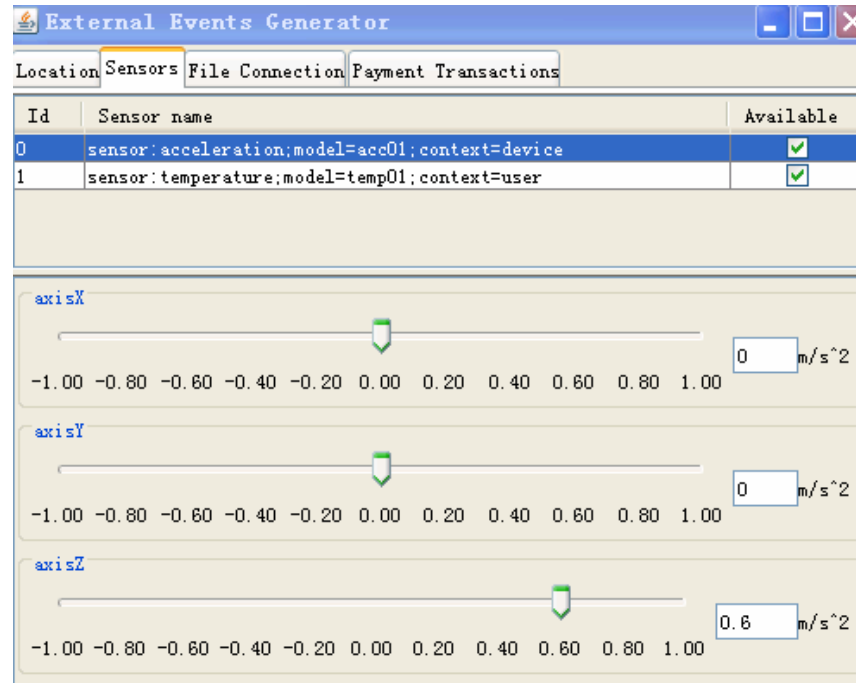


Figure 5.14: External events generator with axisZ value higher than $+0.5$

5.6 Debugging and Troubleshooting

5.6.1 Sequential Switching of the Menu

In the method `datareceived()`, the program judges according to the variable `accelZ`, which indicates the value of the accelerometer on Z-axis. The threshold value is -0.5 for the device to switch to the menu of the next level while 0.5 is the threshold value to return to the upper level. In other words, if the value of `accelZ` is less than -0.5 , which implies the device is making a swift move away from the user, the display will switch to the menu of next level. For instance, if the menu being displayed is the initial menu Welcome, then it will turn into the menu Level1 after the user shakes the device forward very fast. Menu Level2 replaces Level1 in exactly the same way. Sequential switching is performed in this way.

The original code is written as below.

Code:

```

if(Display.getDisplay(this).getCurrent().equals(Welcome))
{
if(accelZ<-0.5)
{
Display.getDisplay(this).setCurrent(Level1);
}
}

```

```

if(Display.getDisplay(this).getCurrent().equals(Level1))
{
if(accelZ<-0.5)
{
Display.getDisplay(this).setCurrent(Level2);
}
}

```

The outcome to execute the codes above was, however, that the display visually changed from the initial menu Welcome to the menu Level2 directly, skipping the menu Level1 in case you trigger the conditions to switch the menu. As a matter of fact, the menu did switch from Welcome to Level1, then from Level1 to Level2. The menu Level1 existed merely an instant during the whole process. The explanation for this phenomenon was that, the program made repetitive judgments and responses to the value of accelerometer in this single time due to the fast execution of the command to switch the menu. It takes only few microseconds to execute this command.

Therefore, I add a loop after the statement to switch the menu.

```

long COUNT=50000000;
Display.getDisplay(this).setCurrent(Level1);

```

```
for (ct = 0; ct < COUNT; ct++)  
{  
    mm = mm + 1;  
}
```

The purpose of this loop is to make the program wait for a short period of time, during which the value of accelerometer on Z-axis can return from the level above threshold to normal. By this way, it avoids repetitive judgments and responses, which lead to the unavailability to perform sequential switching. This method worked after testing.

5.6.2 Choosing a compatible device

Having tested the program successfully on the Java SDK3.0 platform, I chose a device with an embedded accelerometer on the Nokia RDA platform. The type of device is Nokia N85, which has a three-channel accelerometer known as Accelerometer XYZ. The sensor tells the value of acceleration on the X-axis, Y-axis and Z-axis. But an exception is thrown after I installed the program to this device. The details of the message are shown below.

No class Def Found Error

SensorMIDlet:javax/microedition/sensor/DataListener

I checked the technical information of this Nokia N85 and observed that this device did not support JSR 256 though it was equipped with the three-channel accelerometer. Then I searched with the terms that the device should both contain the three-channel accelerometer as well as support the JSR 256, and noticed that there were ten types of phones which met these requirements. The available devices include:

Nokia 5530 XpressMusic

Nokia 5800 XpressMusic

Nokia C6-00

Nokia X6-00

Nokia 5230 Nuron

Nokia 5230

Nokia 5235 Comes With Music Edition

Nokia X6

Nokia N97

Nokia N97 mini

Afterward I tested the program with N97 on the Nokia RDA platform and the application was able to run on that platform. However, the platform can not trigger the event of the movement of the device. The test result remained to be further examined on a real device whereas there was not any equipment available then.

5.6.3 Finding the sensor in the real device

After having tested the program on the Java SDK 3.0 simulation platform, I loaded the program into the real device using the Nokia PC suite and installed it. However, there was an error message which popped up to me that said the accelerometer could not be found. The error message is shown in the Figure 5.15 below.



Figure 5.15: Error message of no valid sensor found

At first, I was quite confused about this exception. I checked through the program and ensured it could run smoothly on the simulation platform. I had also inspected the connection between the mobile device and the computer, and found no problems.

After a period of exploration, I tentatively ran a MIDlet which can list all the sensors within the device as well as the property information about them. I carefully studied the result of this MIDlet and identified the problem. According to the result of the MIDlet, the quantity of the sensor is accelerometer and the context_type is user on the real device. However, the context_type of the accelerometer is device on the simulation platform.

Figure 5.16 shows respectively the results of the MIDlet on the simulation platform and the real device.

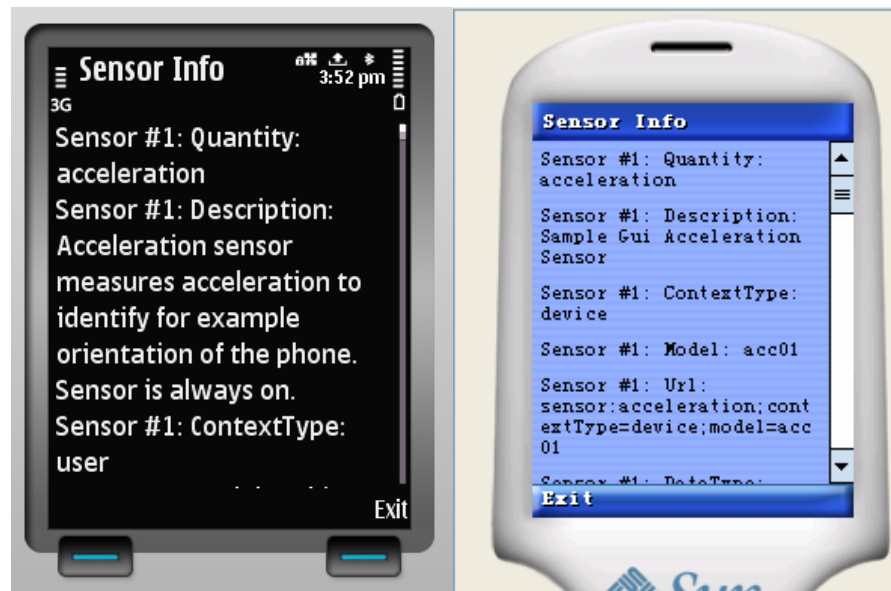


Figure 5.16: Running results of the SensorInfo MIDlet on the real device and emulator

Thus, the exception of being unable to find the sensor would be thrown if the exactly same program ran on the real device. The reason is that the program could not find the

wanted sensor according to the context_type of device.

The original statement is shown below.

```
SensorManager.findSensors("acceleration",SensorInfo.CONTEXT_TYPE_DEVICE);
```

Then I changed the parameter from SensorInfo.CONTEXT_TYPE_DEVICE to SensorInfo.CONTEXT_TYPE_USER in this statement.

The statement modified is the following:

```
SensorManager.findSensors("acceleration",SensorInfo.CONTEXT_TYPE_USER);
```

The problem was resolved after compiling and running the modified program on the real device.

6 CONCLUSION

6.1 Aim of the Study

The aim of the study was to investigate how to build an application based on the mobile sensor API. Specifically, I called the accelerometer inside the mobile phone and create a user interface which can switch the menu according to the information gathered from the sensor. The programming was done on the Java 2 ME platform. The configuration was Connected Limited Device Configuration (CLDC) that is mainly meant for mobile devices and small devices with limited resources. Together with the CLDC, I used MIDP as the profile. The mobile sensor API (JSR 256) is the package that contains the necessary methods and classes in Java to access the sensor inside the mobile phone.

The test was successful on the emulator and remote device access (RDA). The menu switched as I triggered an event of the movement of the device. The menu jumped to the next one if the device had a fast movement forwards, and went back to the last menu if the device moved backwards. As there was no real device supporting the mobile sensor API (JSR 256) available, the test result on a real mobile device remains to be examined.

6.2 Future Prospect of JSR 256

There is a broad prospect of applications for JSR 256. The trend is that more and more sensors are now embedded into the mobile devices, such as the rotation sensor, field strength sensor, thermometers, and the accelerometer. The rotation sensor, for instance, can be implemented in the games, replacing the up arrow with the forward rotation. Also, the whole device may act as a remote. Other device connected wirelessly with the phone can receive the movement of the phone as a signal and judge on that.

The mobile sensor API enables more developers to access the hardware of the mobile

devices using Java. Thus, the device in support of JSR 256 will have a broad marketing prospect.

REFERENCES

- [1]<URL <http://en.wikipedia.org/wiki/Sensor>>
- [2]<URL http://www.freefoto.com/images/16/11/16_11_52---Thermometer_web.jpg>
- [3]<URL <http://www.jacksonoven.com/images/Parts/thermocouples6and12.jpg>>
- [4]<URL
http://img.alibaba.com/photo/10122091/Analog_Meter_Moving_Iron_Instruments_A_C_Voltmeter.jpg>
- [5]<URL
http://static-p4.fotolia.com/jpg/00/06/62/93/400_F_6629316_p12CrWBDwCMLdSYlTqJkNSrvPgsDi23O.jpg>
- [6]<URL <http://www.autooo.net/EN/18719.html>>
- [7]<URL <http://en.wikipedia.org/wiki/Sensor>>
- [8]<URL [http://www.dimensionengin ... elerometers.htm](http://www.dimensionengin...elerometers.htm)>
- [9]<URL
<http://www.dimensionengineering.com/images/products/DE-ACCM3Dbig.jpg>>
- [10]<URL
http://fortbend.k12.tx.us/misc/hhs_robotics/Resources/First_2005/Acceler1.pdf>
- [11]<URL <http://www.dimensionengineering.com/accelerometers.htm>>
- [12]<URL <http://en.wikipedia.org/wiki/Accelerometers>>
- [13]<URL <http://www.800hightech.com/images/apple-ipod-touch-accelerometer.jpg>>
- [14]<URL http://www.gossipgamers.com/wp-content/uploads/2009/05/wii_2.jpg>
- [15]<URL
<http://www.analog.com/en/sensors/inertial-sensors/adx1330/products/product.html>>
- [16]<URL <http://www.java.com/en/javahistory/>>
- [17]<URL <http://www.sun.com/software/learnabout/java/>>
- [18]<URL http://java.sun.com/javame/img/javame_components.gif>
- [19]<URL <http://www.bsg.co.uk/Page.aspx?id=3322> >
- [20]<URL <http://java.sun.com/javame/technology/cdc/>>
- [21]<URL http://java.sun.com/javame/img/digital_media_platform.jpg>
- [22]<URL http://java.sun.com/javame/img/wireless_platform.jpg>
- [23]<URL <http://java.sun.com/products/midp/>>

[24]<URL <http://java.sun.com/javame/img/javame-jsrs.gif> >

[25]<URL http://developers.sun.com/mobility/midp/articles/fsm/fsm_fig3.gif >

[26] Java Mobile Sensor API specification 1.1 Part II API reference

[27] Java Mobile Sensor API specification 1.1 Appendix E Sensor definitions

[28]<URL

http://www.forum.nokia.com/Technology_Topics/Application_Quality/Testing/Remote_Device_Access/>

[29]<URL

http://www.forum.nokia.com/piazza/wiki/images/c/cf/Structure_small.jpg >

APPENDICES

Appendix: Sensor MIDlet code

```

/*Sensor MIDlet*/

import java.io.IOException;

import java.util.Vector;

import javax.microedition.io.Connector;

//import javax.microedition.lcdui.Command;

//import javax.microedition.lcdui.CommandListener;

//import javax.microedition.lcdui.Display;

//import javax.microedition.lcdui.Displayable;

//import javax.microedition.lcdui.List;

//import javax.microedition.lcdui.Canvas;

import javax.microedition.lcdui.*;

import javax.microedition.midlet.MIDlet;

import javax.microedition.sensor.Condition;

import javax.microedition.sensor.ConditionListener;

import javax.microedition.sensor.ChannelInfo;

import javax.microedition.sensor.Data;

import javax.microedition.sensor.DataListener;

import javax.microedition.sensor.SensorConnection;

import javax.microedition.sensor.SensorInfo;

import javax.microedition.sensor.SensorManager;

import javax.microedition.sensor.LimitCondition;

// My plan: if the value of the channel z_axis is larger than 5, then it triggers the
program to exit

public class SensorMIDlet extends MIDlet implements DataListener,
CommandListener, ConditionListener {

```

```

        private static final Command CMD_SELECT = new Command("Select",
Command.SCREEN, 1);

        private static final Command CMD_EXIT = new Command("Exit",
Command.EXIT, 1);

        private static final Command CMD_BACK = new Command("Back",
Command.BACK, 2);

        private static final Command CMD_FORWARD = new Command("Forward",
Command.EXIT, 1);

        private static final Command CMD_X = new Command("X-axis",
Command.ITEM, 2);

        private static final Command CMD_Y = new Command("Y-axis",
Command.ITEM, 2);

        private static final Command CMD_Z = new Command("Z-axis",
Command.ITEM, 2);

        private SensorConnection sensor;
        private String[] channelNames;
        private SensorInfo[] sensorInfos;
        private boolean sensorSelected;

        private List sensorSelector;
//        initiali value false
        private List mainList;
        private List Level1;
        private List Level2;
        private List Level3;

        private Form accontrol;
        private Form rotcontrol;

```

```

private Form norcontrol;
private Form Welcome;

private SensorCanvas sensorCanvas;

/* Initialize MIDlet and detect accelerometer sensor. */
public SensorMIDlet() throws IOException {
    if (System.getProperty("microedition.sensor.version") == null) {
        throw new IllegalArgumentException("JSR256 is not supported!");
    }

    sensorCanvas = new SensorCanvas(this);
    sensorInfos = getSensorInfos();
    if (sensorInfos == null) {
        throw new IllegalArgumentException(
            "Valid accelerometer sensor not found");
    }
    sensorSelector = new List("Sensor User Interface", List.EXCLUSIVE);
    for (int i = 0; i < sensorInfos.length; i++) {
        sensorSelector.append(sensorInfos[i].getUrl(), null);
    }

    Welcome = new Form("Welcome to Sensor UI");
    Welcome.append("Here is a demonstration MIDlet on how to control the
phone with the embedded sensor within it. \n\nShake the phone towards yourself to go
back, or forwards to move to next level");

    Welcome.addCommand(CMD_EXIT);
    Welcome.addCommand(CMD_FORWARD);
    Welcome.setCommandListener(this);

```

```
sensorSelector.addCommand(CMD_SELECT);
```

```
sensorSelector.addCommand(CMD_EXIT);
```

```
sensorSelector.setCommandListener(this);
```

```
String[] st={"Acceleration Control","Rotation Control","Normal Control"};
```

```
Image[] im=null;
```

```
String[] st2={"Forward","Back"};
```

```
mainList= new List("Welcome to sensor UI ",List.IMPLICIT);
```

```
mainList.addCommand(CMD_EXIT);
```

```
mainList.addCommand(CMD_SELECT);
```

```
mainList.append(st[0], null);
```

```
mainList.append(st[1], null);
```

```
mainList.append(st[2], null);
```

```
Level1= new List("Level1 ",List.IMPLICIT);
```

```
Level1.append(st2[0], null);
```

```
Level1.append(st2[1], null);
```

```
Level2= new List("Level2 ",List.IMPLICIT);
```

```
Level2.append(st2[0], null);
```

```
Level2.append(st2[1], null);
```

```
Level3= new List("Level3 ",List.IMPLICIT);
```

```
Level3.append(st2[0], null);
```

```
Level3.append(st2[1], null);
```

```
accontrol= new Form("Level 1");
```

```
accontrol.append("Now that you are in Level 1. Shake towards yourself to  
go back, or forwards to Level 2.");
```

```
accontrol.addCommand(CMD_BACK);
```

```
accontrol.addCommand(CMD_FORWARD);
```

```
accontrol.setCommandListener(this);
```

```
rotcontrol= new Form("Level 2");
```

```
rotcontrol.append("Now that you are in Level 2. Shake towards yourself to  
go back, or forwards to Level 3.");
```

```
rotcontrol.addCommand(CMD_BACK);
```

```
rotcontrol.addCommand(CMD_FORWARD);
```

```
rotcontrol.setCommandListener(this);
```

```
norcontrol= new Form("Level 3");
```

```
norcontrol.append("Now that you are in Level 3. Shake towards yourself to  
go back");
```

```
norcontrol.addCommand(CMD_BACK);
```

```
norcontrol.addCommand(CMD_FORWARD);
```

```
norcontrol.setCommandListener(this);
```



```
}

public void startApp() {
    if (!sensorSelected) {

        Display.getDisplay(this).setCurrent(sensorSelector);

    } else {

//        starts from here
        Display.getDisplay(this).setCurrent(sensorCanvas);
// when I change the parameter sensorCanvas to be mainList, which is a simple List
// I made,
//        the method dataReceived() does not work
        sensor.setDataListener(this, 1);
    }
}

public void pauseApp() {
    sensor.removeDataListener();
}

public void destroyApp(boolean unconditional)
{
    if (sensor != null) {
        try {
            sensor.removeDataListener();
            sensor.close();
        }
    }
}
```

```

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    notifyDestroyed();
}

public void commandAction(Command c, Displayable d) {

    if (d.equals(sensorSelector))
    {
        if (CMD_SELECT.equals(c))
        {
            new Thread()
            {
                public void run()
                {
                    int idx = sensorSelector.getSelectedIndex();
                    channelNames =
getSensorInfoChannelsNames(sensorInfos[idx]);

                    try
                    {
                        sensor = (SensorConnection)
Connector.open(sensorInfos[idx].getUrl());
                        sensorSelected = true;

Display.getDisplay(SensorMIDlet.this).setCurrent(Welcome);

                        sensor.setDataListener(SensorMIDlet.this, 1);
                    } catch (IOException e) {e.printStackTrace();}
                }
            }.start();
        }
    }
}

```

```

} else if (CMD_EXIT.equals(c))
{
    destroyApp(false);
    notifyDestroyed();
}
}
else if (d.equals(Welcome))
{
    if (CMD_FORWARD.equals(c))
        Display.getDisplay(SensorMIDlet.this).setCurrent(accontrol);
    else if (CMD_EXIT.equals(c))
    {
        destroyApp(false);
        notifyDestroyed();
    }
}

else if (d.equals(accontrol))
{
    if (CMD_FORWARD.equals(c))
        Display.getDisplay(SensorMIDlet.this).setCurrent(rotcontrol);
    else if (CMD_BACK.equals(c))
        Display.getDisplay(SensorMIDlet.this).setCurrent(Welcome);
}

else if (d.equals(rotcontrol))
{
    if (CMD_FORWARD.equals(c))
        Display.getDisplay(SensorMIDlet.this).setCurrent(norcontrol);
    else if (CMD_BACK.equals(c))

```

```

        Display.getDisplay(SensorMIDlet.this).setCurrent(accontrol);

    }

    else if (d.equals(norcontrol))
    {

        if (CMD_BACK.equals(c))
            Display.getDisplay(SensorMIDlet.this).setCurrent(rotcontrol);
    }

}

/* pass acceleration data to the sensorCanvas.*/
public void dataReceived(SensorConnection sensor, Data[] data, boolean
isDataLost)
{
    double accelX = 0;
    double accelY = 0;
    double accelZ = 0;
    long ct;
    long COUNT=50000000;
    int mm=0,kk=0;
    for (int i = 0; i < data.length; i++)
    {
        if(channelNames[0].equals(data[i].getChannelInfo().getName()))
        {
            accelX = data[i].getDoubleValues()[0];

```

```

}else
if(channelNames[1].equals(data[i].getChannelInfo().getName()))
{
    accelY = data[i].getDoubleValues()[0];
}else
if(channelNames[2].equals(data[i].getChannelInfo().getName()))
{
    accelZ = data[i].getDoubleValues()[0];
}
}

// if(Display.getDisplay(this).getCurrent().equals(sensorCanvas))
//     Display.getDisplay(this).setCurrent(Welcome);

//     It is now displaying the interface of sensorCanvas
//     changing the displayable here does not stop the process of
//     "dataListening"

if(Display.getDisplay(this).getCurrent().equals(Welcome))
{
    if(accelZ<-0.5)
    {
        Display.getDisplay(this).setCurrent(accontrol);
        for (ct = 0; ct< COUNT; ct++)
        {
            mm=mm+1;
        }
    }

//     switch(mainList.getSelectedIndex())
//     {
//     case 0: Display.getDisplay(this).setCurrent(accontrol);

```

```

//
////          I want to make the programme wait for 4 sec in order the
sensor value could
////          go back to the normal level, but it seems not to work
//
//          for (ct = 0; ct< COUNT; ct++)
//              {
//                  mm=mm+1;
//              }
////          try
////              {
////                  Thread.sleep(4000);
////              }
////          }catch (InterruptedException e){}
//
//          break;
//          case 1: Display.getDisplay(this).setCurrent(rotcontrol);
//
//              break;
//          case 2: Display.getDisplay(this).setCurrent(norcontrol);
//
//              break;
//          }
//      }
//
//          It is now displaying the interface of Next
else if(Display.getDisplay(this).getCurrent().equals(accontrol))

```

```
//      Level 1
{
    if(accelZ>0.5)
    {
        for (ct = 0; ct< COUNT; ct++)
        {
            mm=mm+1;
        }
        Display.getDisplay(this).setCurrent(Welcome);
    }

    if(accelZ<-0.5)
    {
        for (ct = 0; ct< COUNT; ct++)
        {
            mm=mm+1;
        }
        Display.getDisplay(this).setCurrent(rotcontrol);
    }

    if(accelY>0.5)
    {
        destroyApp(false);
    }
}
```

```
        notifyDestroyed();
    }

}

if(Display.getDisplay(this).getCurrent().equals(rotcontrol))
{
    if(accelZ>0.5)
    {

        for (ct = 0; ct< COUNT; ct++)
        {
            mm=mm+1;

        }
        Display.getDisplay(this).setCurrent(accontrol);
    }

    if(accelZ<-0.5)
    {

        for (ct = 0; ct< COUNT; ct++)
        {
            mm=mm+1;

        }
        Display.getDisplay(this).setCurrent(norcontrol);
    }

}
```



```

if(Display.getDisplay(this).getCurrent().equals(norcontrol))
{
    if(accelZ>0.5)
    {

        for (ct = 0; ct< COUNT; ct++)
            {
                mm=mm+1;
            }
        Display.getDisplay(this).setCurrent(rotcontrol);
    }
}
// if(Display.getDisplay(this).getCurrent().equals(Level2))
// {
//     if(accelZ>0.5)
//     //
//     Display.getDisplay(this).setCurrent(Level1);
//     //
//     if(accelZ<-0.5)
//     //
//     Display.getDisplay(this).setCurrent(Level3);
//     //
//     //
//     //
// }
//
sensorCanvas.setValues(accelX, accelY, accelZ);
}

```

```

/* Detect appropriate sensor info. */
private SensorInfo[] getSensorInfos()
{
    /* Find all device accelerometers*/
    SensorInfo[] sensorInfos =

SensorManager.findSensors("acceleration",SensorInfo.CONTEXT_TYPE_DEVICE);

    Vector validInfos = new Vector();
    for (int i = 0; i < sensorInfos.length; i++) {
        if (getSensorInfoChannelsNames(sensorInfos[i]) != null) {
            validInfos.addElement(sensorInfos[i]);
        }
    }
    SensorInfo[] ret = null;
    if (validInfos.size() > 0) {
        ret = new SensorInfo[validInfos.size()];
        validInfos.copyInto(ret);
    }
    return ret;
}

/* Detect valid channel names for sensor. */
private String[] getSensorInfoChannelsNames(SensorInfo sensorInfo) {
    Vector channelNames = new Vector();

    ChannelInfo[] channelInfos = sensorInfo.getChannelInfos();
    if (channelInfos.length > 1) {
        /* Accelerometer must support at least 2 channels*/
        for (int i = 0; i < channelInfos.length; i++) {

```

```

        if (ChannelInfo.TYPE_DOUBLE ==
channelInfos[i].getDataType()) {
            /* The channel type must be double*/
            channelNames.addElement(channelInfos[i].getName());
        }
    }

//          sensor.getChannel(channelInfos[0]).addCondition(this, new
LimitCondition(5,Condition.OP_GREATER_THAN));

// to add a Limit Condition for channel 0 where a notification will be
sent when the value is greater than 5

    if (channelNames.size() > 2) {
        String[] names = new String[3];
        names[0] = (String) channelNames.elementAt(0);
        names[1] = (String) channelNames.elementAt(1);
        names[2] = (String) channelNames.elementAt(2);
        return names;
    }
}
return null;
}

public void conditionMet(SensorConnection sensor, Data data, Condition condition)

{
    destroyApp(false);
    notifyDestroyed();
}
}

```