REMOTE CONTROLLING PRESENTATION USING ANDROID APPLICAITON

by

Mohamad Firdaus Bin Zahari

Electrical and Electronic Engineering

11386

DISSERTATION

submitted to the Electrical & Electronics Engineering Programme

in Partial Fulfillment of the Requirements

for the Degree

Bachelor of Engineering (Hons)

(Electrical & Electronic Engineering)

MAY 2012

UNIVERSITI TEKNOLOGI PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Remote Controlling Presentation using Android Application

by

Mohamad Firdaus bin Zahari

A project dissertation submitted to the

Electrical and Electronics Engineering Program

Universiti Teknologi PETRONAS

in partial fulfilment of the requirement for the

Bachelor of Engineering (Hons)

(Electrical and Electronics Engineering)

Approved by,

_____

(Mr Azman bin Zakariya)

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or one by unspecified sources or persons.

_____

MOHAMAD FIRDAUS BIN ZAHARI

ABSTRACT

The numbers of smartphone users are increasing day by day. One of the popular operating system used to power the smartphone is Android. Android is maintained and developed by Google which they are providing Android SDK that contains the tools and API needed to develop a custom application. By manipulating the capability to access built-in sensors and reading of the smartphones such as touch screen, phone orientation, accelerometer and GPS, a custom presentation remote controlling application can be developed. This will be proven useful as the users will not have to buy extra remote devices to assist them when they are doing presentation as they can install this application directly to their phones. The application will be designed and develop using Android SDK and coded in Java programming language. Result shows that by manipulating the available sensors on the smartphones, we can utilize the Bluetooth and Wi-Fi functionality of the smartphones to connect with remote computer thus communicating with it. The application will send the data based on user input which then will be interpreted by the server application on the remote computer to control the slide presentation. The elements in this report contain introduction, problem statement, objectives, literature review and methodology which are used to develop the application. The discussion of the obtained results will be looked further in this project.

ACKNOWLEDGEMENT

Firstly, I would like to express my gratitude to the God for giving me guidance, assistance and strength in completing this Final Year Project successfully on time.

I extend my gratitude to my supervisor and mentor, Dr. Azman bin Zakariya whose help, advice and guidance have been a great help in order for me to complete this project successfully. His willingness to teach and share me his knowledge contributed greatly to my project.

My gratitude also goes to my co-supervisor, Dr Zuhairi Baharudin, for his full support towards my project. His helpful comments and suggestions had contributed a lot to me not only to complete the project but also further enhance the results. His kindness, patience and friendly approach will always be appreciated.

Lastly, my appreciation goes to my family and friends who are supporting me thoroughly towards the completion of my project. I would also like to express my gratitude to all people who have contributed directly or indirectly in accomplishment of this project.

TABLE OF CONTENTS

CERTIFICATION OF APPROVAL……...…i

CERTIFICATION OF ORIGINALITY.....ii

ABSTRACT……..…….....iii

ACKNOWLEDGEMENT……..…….iv

TABLE OF CONTENTS……...….….v-vii

LIST OF FIGURES……..……...viii-ix

LIST OF TABLES…….x

LIST OF ABBREVIATION…….……xi

CHAPTER 1: INTRODUCTION……..……1

1.1 Background of Study……...…1-2

1.2 Problem Statement……..2

1.3 Significance of Project…….……...3

1.4 Objectives……...3

1.5 Scope of Study……3-4

1.6 Relevancy of Project……..…….4

1.7 Feasibility of Project……...4

CHAPTER 2: **LITERATURE REVIEW**……..5

2.1 Android Application Framework……..…5-9

2.2 Machine-to-Machine Network……..9-11

2.3 Android-based SoD Client…….……..11-14

2.4 Android User Interface Design……..…15-16

CHAPTER 3: **METHODOLOGY**……..…….17

3.1 Research Methodology……17

3.2 Flow Chart……...…….……...18

3.3 Project Task……..19

3.4 Gantt Chart and Milestones...……..…20-22

3.5 Tools……...…..22

3.6 Project Schedule……..….22

CHAPTER 4: **RESULT AND DISCUSSION**…….23

4.1 Results…….23-32

4.2 Discussion……..……..……33

CHAPTER 5: **CONCLUSION AND RECOMMENDATION**…….34

5.1 Conclusion……34

5.2 Recommendation.…….34

REFERENCES……35-36

APPENDICES……..37

Appendix A……38

Appendix B……44

Appendix C……49

Appendix D……52

Appendix E……57

LIST OF FIGURES

Figure 1: Android system structure……...5

Figure 2: Position of DVM in Android system……...…..7

Figure 3: Components structure of Android application……..7

Figure 4: Lifecycle of an activity……..8

Figure 5: M2M architecture for different domains…….10

Figure 6: Communication principle between the components……..…….……10

Figure 7: Software architecture for SoD……..…12

Figure 8: Resource connection after starting program……13

Figure 9: Resource disconnection after stopping program…….13

Figure 10: Operating procedures for resource allocation……...…14

Figure 11: Operating procedures for resource deallocation……14

Figure 12: The design procedure of Android smartphone user interface……...15

Figure 13: A Tree-structured UI…….16

Figure 14: Project methodology……...…..18

Figure 15: Application workflow……23

Figure 16: Main screen of the application……..…24

Figure 17: Connection selection in the application……24

Figure 18: Application requesting to turn on the Bluetooth……..……25

Figure 19: Application turns on the Bluetooth once permission is given……..25

Figure 20: Application lists the available devices that can be connected with……..26

Figure 21: Application will prompt for IP address…….……26

Figure 22: Control screen of the application……..…….……27

Figure 23: Data sent by the application is being interpreted on the server……28

Figure 24: BluetoothScreen.java flow……..…….29

Figure 25: BluetoothCommandService.java flow……..……30

Figure 26: WifiScreen.java flow……..……..31

Figure 27: ProcessConnectionThread.java flow…….32

LIST OF TABLES

Table 1: Function for SoD Client……....12

Table 2: Gantt chart for Final Year Project I……..20

Table 3: Milestones of the Final Year Project I……..21

Table 4: Gantt chart for Final Year Project II…….21

Table 5: Milestones of the Final Year Project II…….22

Table 6: Project schedule for Final Year Project I.....22

Table 7: Project schedule for Final Year Project II……22

Table 8: List of available commands in the application…….28

LIST OF ABBREVIATION

3G	3rd Generations
GPS	Global Positioning System
SDK	Software Development Kit
API	Application Programming Interface
GUI	Graphical User Interface
DVM	Dalvik Virtual Machine
UI	User Interface
M2M	Machine-to-machine
XMPP	eXtensible Messaging and Presence Protocol
CDP	Content Data Provider
WLAN	Wireless Local Area Network
LAN	Local Area Network
SoD	System on Demand
XML-RPC	eXtensible Markup Language Remote Procedure Call
TCP	Transmission Control Protocol
VM	Virtual Machine
XML	eXtensible Markup Language
JAR	Java Archive
IP	Internet Protocol

CHAPTER 1
INTRODUCTION

1.1 Background of Study

The numbers of mobile phone users are increasing day by day and such products are becoming powerful compared to the previous generation. Mobile phones these days are expected to bridge the gap between desktop computers and hand-held devices [1] by being able to carry out the task that can be done using a personal computer such video call, internet surfing via mobile and sharing content between two devices [2].

In order for mobile phone to do these kinds of task, mobile phones today are equipped with new and latest hardware such as faster processing power for computationally intensive applications, larger memories to handle lots of application and faster Internet connection thanks to the 3rd Generation (3G) technology [1]. Smartphones today are also different with traditional cell phones in a sense that they are equipped with multiple sensors such as global positioning system (GPS), touch screen, accelerometer and the phone orientation itself to enhance user experience [3]. These sensors open up possibilities for new functionality that cannot be achieved by conventional cell phones.

Plus, the application programming platforms and development tools that are used to build application for mobile devices such as Java, Open C, Objective C, Python and Flash Lite help the developer to create highly functional mobile application that are benefited from state-of-the-art hardware equipped with the smart phones [4].

Together with the advancement in hardware of smartphones, the operating system that supports the application and running the devices are also becoming increasingly powerful and flexible. One of popular operating system available for smartphones is Google Android. The reason that Android becomes so popular is because of their open-source policy that is widely adopted by industry and end users. Android

includes an operating system, middleware and core applications, and they are also comes with downloadable Software Development Kit (SDK) [5].

The SDK provides necessary tools and Application Programming Interface (API) to help the developers in developing application for Android platform. Another interesting feature of Android is that there are different types of hardware and sensors that are accessible to developers where can benefit the developer to write better and creative applications compared to the other smartphones as they can access different hardware in the smartphones such as magnetometers, proximity and pressure sensors, touchscreens and gyroscopes [6].

1.2 Problem Statement

Google Android opens up vast possibilities to develop highly functional applications since they are provided with SDK that ease up the development process. Plus, the nature of Android devices that can be accessible directly by the application allowing infinite possibilities to comes up with an application that is useful yet creative.

Even though there are already remote devices that are capable of controlling PowerPoint presentation remotely, but the users need to buy extra devices that are costly. Besides, these kinds of device such as wireless mouse or dedicated remote control usually offer basic functionality only.

This project aims to develop an application that can be used to remote control PowerPoint presentation from Android smartphones. Due to this problem stated above, the application will helps the user to remote controlling a presentation without the need to buy extra devices yet offering extended functionality to enhance the user experience. Multiple sensors and hardware can be manipulated with the help of Java programming to create remote presentation control specifically for Android platform.

1.3 Significance of Project

The improvement of technology used in smartphones be it software or hardware are significant nowadays. With multiple built-in sensors, equipped with open source operating system such an Android, we can use this technology to our advantage.

By utilizing the built-in sensors and hardware such as accelerometer, thermometer and touch screen that are directly accessible by the Android application, a presentation remote control application can be develop where input registered by the sensors will be sent to the remote computer to handle basic presentation function.

1.4 Objectives

To develop a fully functional Android based applications that capable of controlling presentation remotely on supported devices such as personal computer and laptops. The sub objectives of the project are listed as following:

i. To learn about the basic framework architecture of Android operating system.
ii. To utilize multiple hardware and sensors equipped on smartphones to interact with connected devices.

1.5 Scope of Study

The scope of this project is doing research and literature review on Android applications, methodology of graphical user interface (GUI) designing for Android application and also utilization of sensors in the smartphone to create the applications. Basic application framework structure will be drawn and coded using Java language in the provided Android SDK. Further testing will be carried out to design the best presentation remote control application for Android-powered mobile phones.

The application will be built specifically for Android 2.3 (Gingerbread) and higher. The application can be installed to all Android-based smartphone and the server-side application can be run using Java Runtime Environment in Windows and Linux OS.

1.6 Relevancy of Project

Since the number of smartphone users is increasing day by day, with proper programming tools and SDK coupled with advancement of hardware technology used, a custom presentation remote control application will be beneficial. This will help to save the cost from buying an extra dedicated device that is not only restricted with limited functionality but also is expensive. Another important aspect is that if the user owns Android-powered mobile phones, they can install this application directly.

1.7 Feasibility of Project

This project will be done in two semesters which basically includes three basic areas, which are research, application development and also beta-testing and improvement of the final prototype. Android SDK will be utilized to develop the application from scratch using Java programming language. Once the prototype is done, application testing will be done. Based on the description above, it is very clear that this project will be feasible to be carried out within the time frame.

CHAPTER 2

LITERATURE REVIEW

2.1 Android Application Framework

Android is Linux based operating system for mobile devices such as smartphones and tablets. It is an open source project led by Google so that means its openness will promote technology innovation and at the same time reduce the development cost. Android SDK is available to download if the end-users decide to develop an application on their own to suit their own taste.

Common structure of a program is called application framework [7]. This framework is reusable, meaning that the very same framework that consist a set of architectures of application can be used again to create another application with another set of classes and instances. Android software structure can be divided to four basic levels; applications, application framework, library and Android runtime, and lastly Linux kernel as operating system.

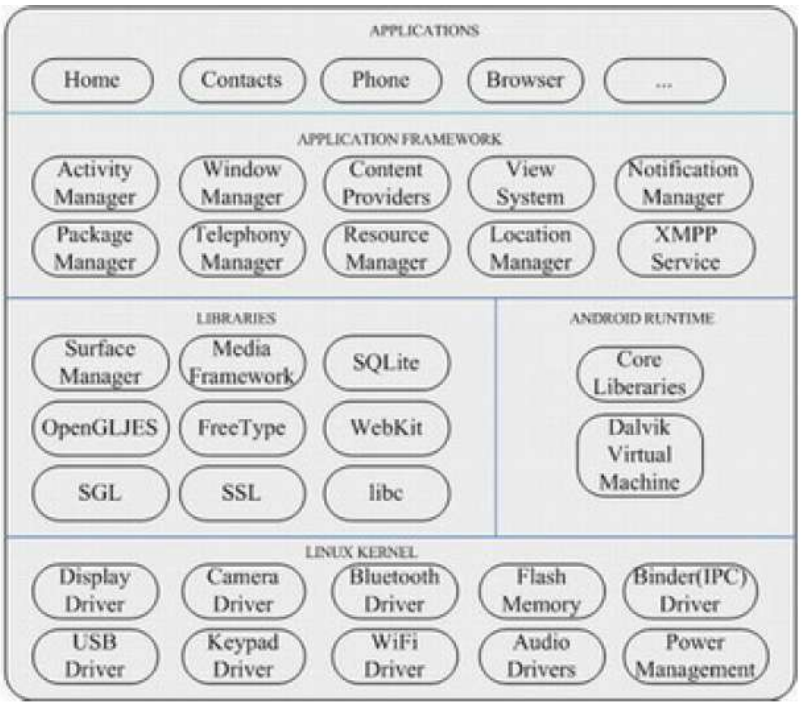

Figure 1: Android system structure

The first and top layer is applications layer. This is where all Android applications such as web browser, music player and email client are residing [7]. These applications are written in Java language and developer can benefit from the API provided in application framework layer below to develop their own application for application layer.

The second layer below the application layer is application framework. This is where Android operating system stands out compared to the other mobile operating systems. There are handfuls of services and managers that can be reuse by the users for their applications. As an example, View System is used to build and develop application with beautiful user interfaces, and it includes things such as grids, text boxes, buttons and lists. Content Providers allows the application to access the data from another application or even share their own data.

Next layer is the Libraries and Runtime. This layer contains a set of C/C++ libraries that are used by various Android components. It also provides support and dependencies needed by various Application Framework of higher layer. As being told earlier, Android application is written in Java language but they do not use Java runtime components to run the application. Instead the application will use Android runtime component [7].

Android runtime component basically consisted of two main parts. The first one is core library and Dalvik virtual machine (DVM). The former is responsible for storing all the necessary Java classes which are needed by the application meanwhile the latter is essentially a Java virtual machine that is optimized and redesigned by Google for usage in Android-capable smartphone devices [2].

DVM is register-based virtual machine that can have multiple instances on one device. As for example, one Android application is running on an instance of DVM, and another application can use another instance of DVM to run. It also use little memory resources and relies on underlying Linux kernel to function properly [7].

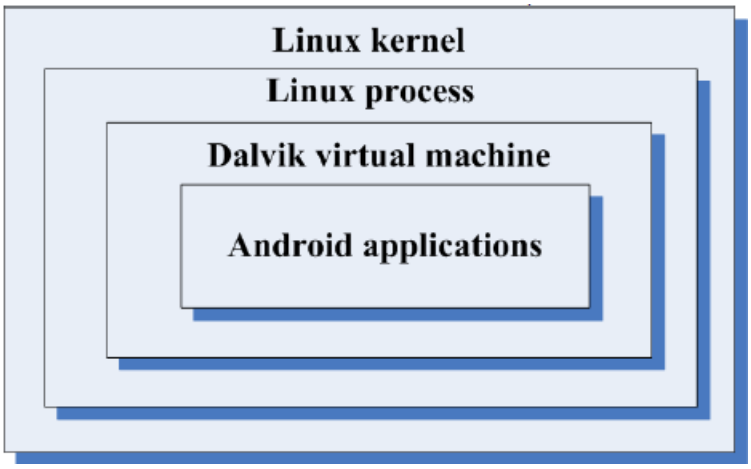

Figure 2: Position of DVM in Android system

The last and bottom layer is the Linux kernel itself. It contains all the necessary drivers for the hardware to function properly such as camera driver, Bluetooth driver, audio driver and keypad driver. It also acts as an abstraction layer between the hardware and the software. Besides, some components and functions of DVM also depend on the Linux kernel itself.

Android is unique as a certain application can use other application to complete its own task or function. As an example, an application that needs to play some audio files can directly use the function in a dedicated audio player application to achieve its task. Furthermore, Android components can be further divided into several components which are Activity, Intent Receiver or Broadcast Receiver, Service and Content Provider [2].

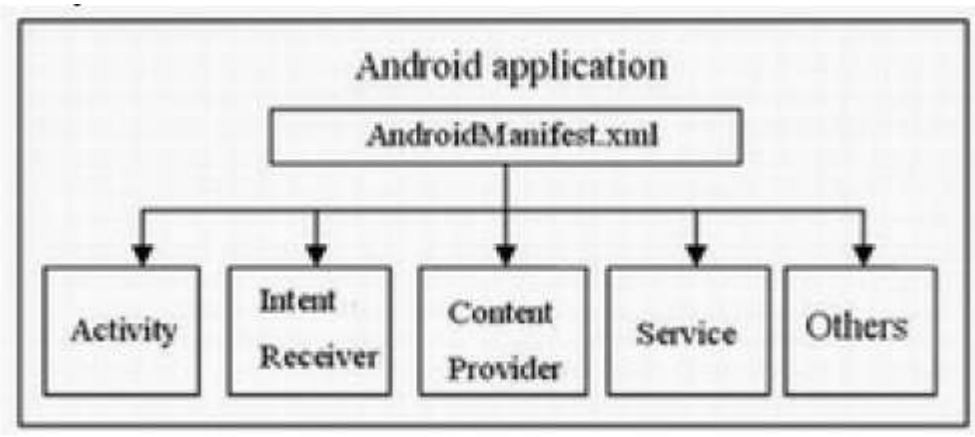

Figure 3: Components structure of Android application

The first component, Activity is usually represented with a single screen with a user interface (UI) [7]. The activities are independent from one another and work together to improve the user experience when using an application. Usually, there will be a "main" activity and other activities can be created as a subclass of Activity. Each activity performs different function and when a new activity starts, the previous activity are stopped as shown is Figure 4.

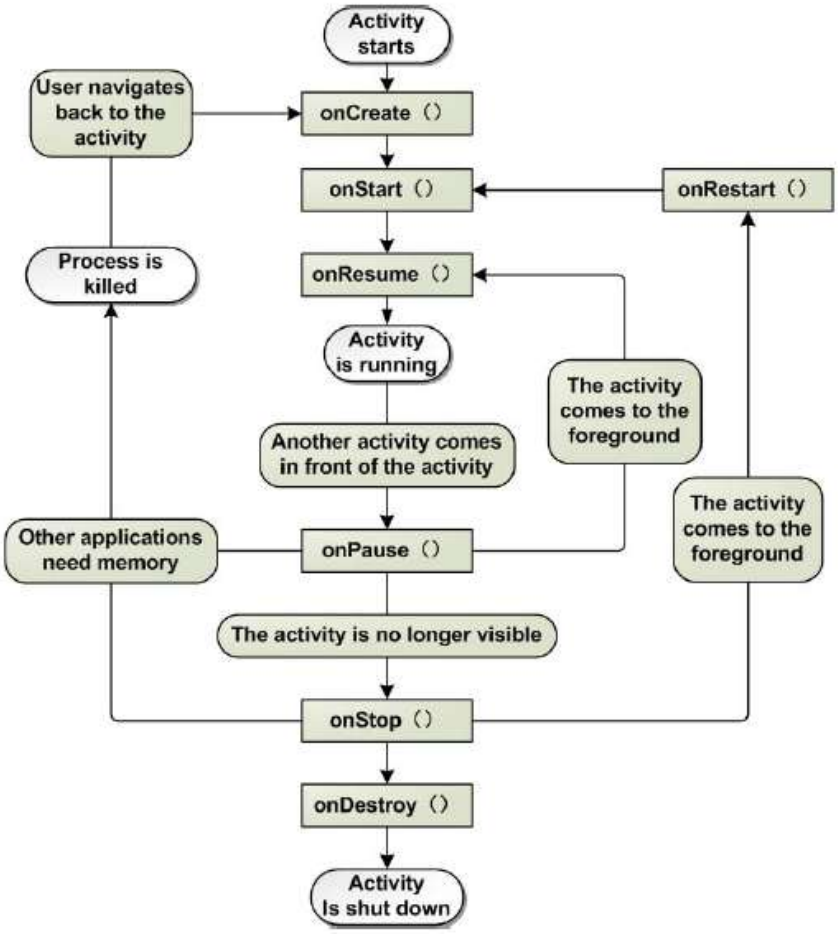

Figure 4: Lifecycle of an activity

Service is an Android component that performs work for remote process and usually running in background so it does not provide user interface to the users. The activity will first connect to the service required and from there, the activity can communicate with the service to perform the task or achieve a specific function.

The third component is Content Providers. As its name suggests, Content Provider functions to provide data share mechanism between applications. The data is accessible to the application via file system, a database such as SQLite database or any other persistent storage location. A content provider can be created as a subclass of Content Provider, which will define the data format that is supported by the application. It also helps other application to query and modify the data inside the application [7].

Lastly, Intent Receiver or also known as Broadcast Receiver handles the reception of all system wide broadcast. One example of broadcast is when a battery is low; the system will broadcast the signal to notify the user that the battery needs to be recharged. Broadcast can be initiated either by system or even an application. Broadcast does not have a specific user interface, but it can also come in form of notification to alert the user [8].

2.2 Machine-to-Machine Network

In a network, there are multiple types of device that are connected to it, be it smartphones, tablets, personal computers and laptops. Machine-to-machine (M2M) network is a network of devices with diverse capabilities to interact only and if only with human intervention. This concept is getting popular as the numbers of devices connecting to a network is increasing days by days.

Such devices are communicating through a network via different methods such as wire lines, Wi-Fi, Bluetooth and 3G [9]. During this communication, data can be then collected, analyzed and acted upon. Context-awareness involves the collection and assessment of data depending on the system design.

Usually, for a M2M based application to work, it will need to fulfill four stages. Firstly data will be collected through sensors. As for smartphones today, they are already equipped with different kinds of sensors such as thermometer, accelerometer, touch screen sensors and GPS. The data can be collected either periodically or upon request. The collected data will then be transmitted over the network to another M2M device that is capable to handle and store the data transmitted [10]. The

collected data will then be analyzed and assessed and lastly, act upon them based on the analysis result.

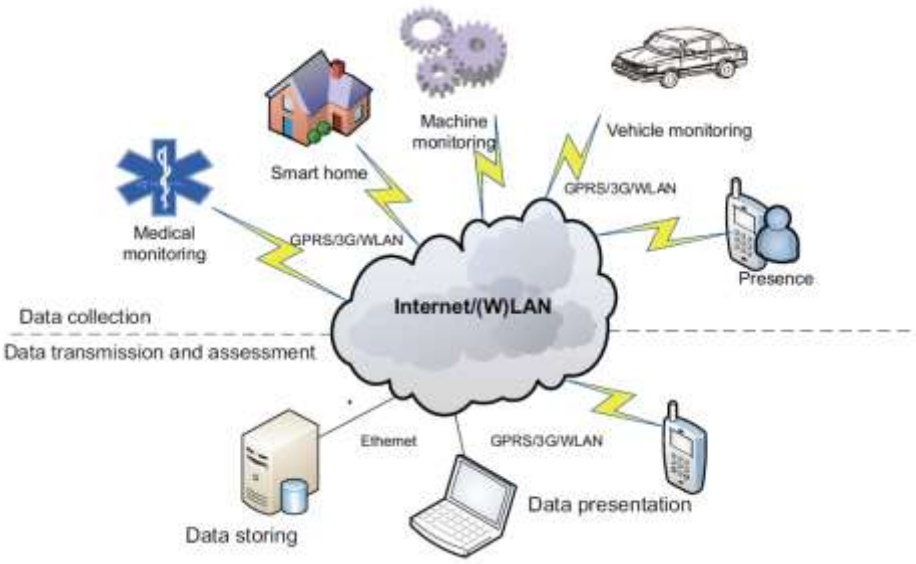

Figure 5: M2M architecture for different domains

For a basic M2M network, there are three parts involved which are the smartphone itself, eXtensible Messaging and Presence Protocol (XMPP) and Context Data Processor (CDP). The components are communicating using XMPP protocol thus for the system to work, a dedicated XMPP server will be needed. The smartphone will act as the client part of the network and the reading from the built-in sensors will be forwarded to the CDP to be processed and analyzed [9].

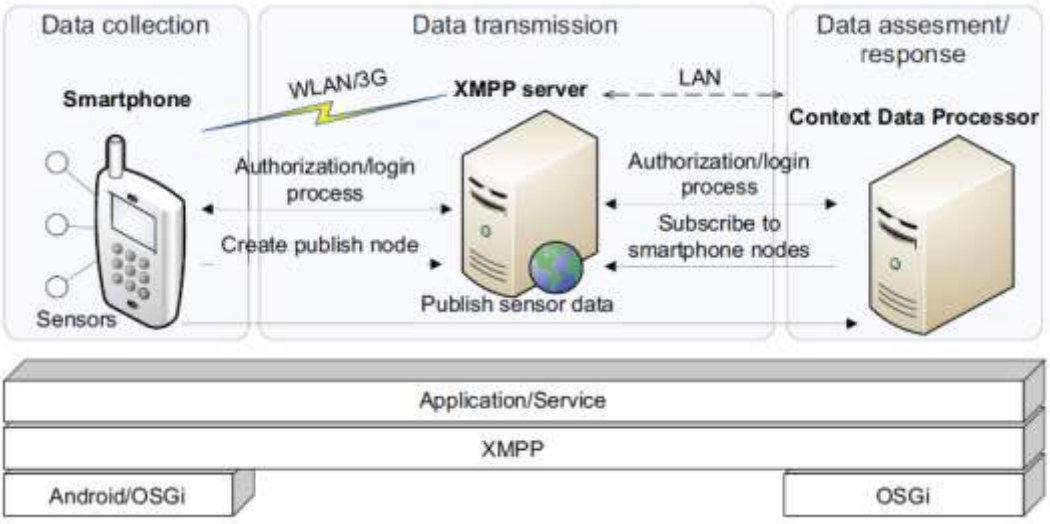

Figure 6: Communication principle between the components

The communication between the smartphone and XMPP server can be done either by 3G networks or Wireless Local Area Network (WLAN) meanwhile for a better and more reliable connection, the communication between the CDP and XMPP will be done via Local Area Network (LAN). The CDP will be responsible to track the status of every available smartphone in the network and received the required context data via the data collected from the sensors reading of the smartphones.

From the client side, context data is provided by the built-in sensors in the Android-based smartphones. The application will then transmitted the data to the XMPP server over the WLAN via formatted XMPP messages. This application will run as long as it needed until the service is stopped by the user.

On the other hand, at the server side, XMPP server is run on a dedicated computer that is connected to the network. The data available will then be collected by the CDP that is connected to the XMPP via local network. CDP then will assess and examine the data collected. This data will be proven useful for tasks such as tracking system, alarming and remote controlling [9].

2.3 Android-based SoD Client

System on demand (SoD) is a framework that provides a function of dynamically configurable peripheral devices for virtual peripheral resources such as monitors, keyboards and mice [11].

There are several applications in Google Android market that enable remote presentation which allow the user to view a remote presentation of his/her current computer state via smartphones. The only downside is that this kind of application cannot separate resources that consisting remote presentation such as keyboard and mouse.

For SoD client, there are basically three main functions which are system login, resource allocation/deallocation and resource virtualization.

Table 1: Function for SoD Client

Function	Operation	
SoD System Login	Authenticate users through XML-RPC communication	
Resource allocation / deallocation	Receive list for currently existing virtual machines and virtual resources	
	Execute the virtual machine through user's input	
	Allocate resources through user's input	
	Deallocate resources through user's input	
Resource virtualization	Keyboard	Transmit keyboard inputs to the server
	Mouse	Transmit touch screen inputs to the server
	Monitor	Receive frame buffer data of server and draw on the screen

The client software of SoD consists of eXtensible Markup Language Remote Procedure Call (XML-RPC) module, a Command Controller module and each module for monitor, mouse and keyboard respectively.

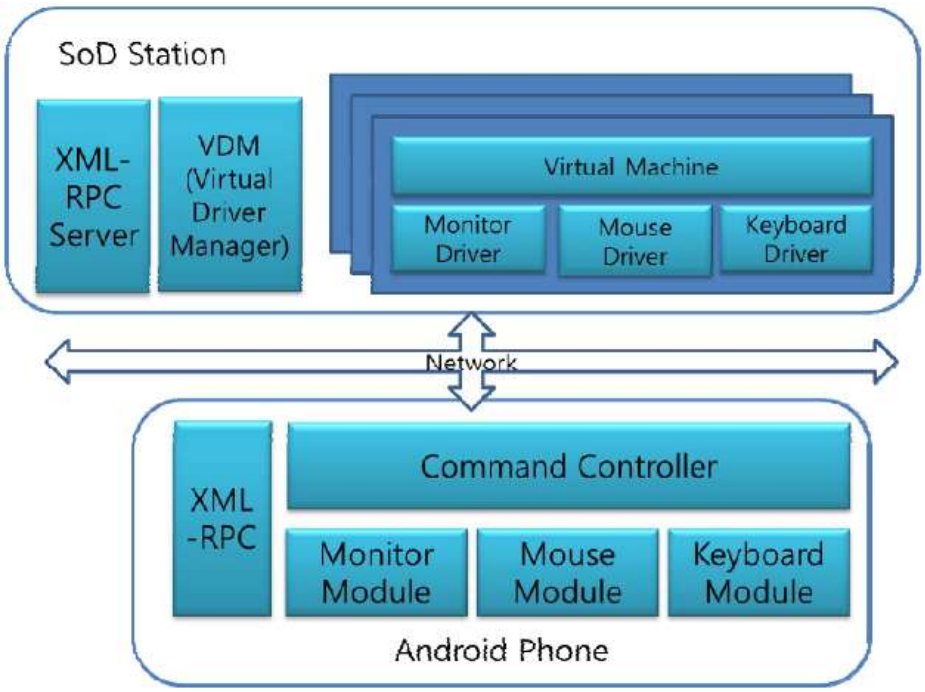

Figure 7: Software architecture for SoD

XML-RPC server functions to receive the list of resources and virtual machine available. It will then allocate/deallocate the resource accordingly and also updating the resources whenever there are changes to the resources. Meanwhile the Command Controller module is responsible to virtualize the resources of the phone [11]. All keyboard, mouse and monitor modules perform transmission control protocol (TCP) connection to the virtual machine (VM).

Design and operating procedures of SoD can be seen from Figure 8 to Figure 11. Figure 8 describe the resource connection right after starting the program. Figure 9 describes the operating procedures for resource disconnecting after stopping the program and Figure 10 show the operating procedures for resource allocation. Lastly, Figure 11 describes the operating procedures for resource deallocation.

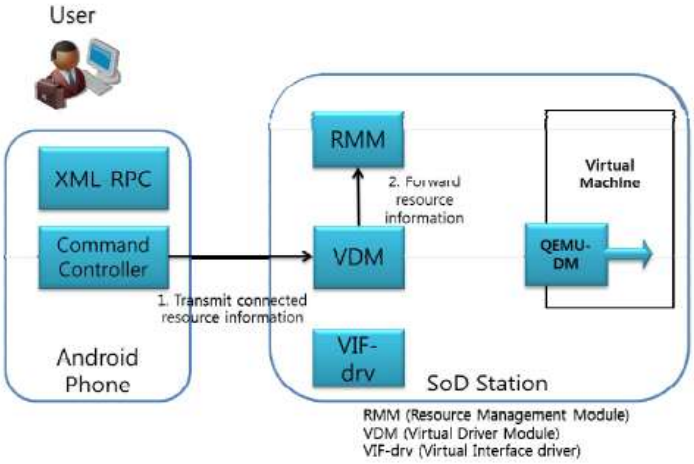

Figure 8: Resource connection after starting program

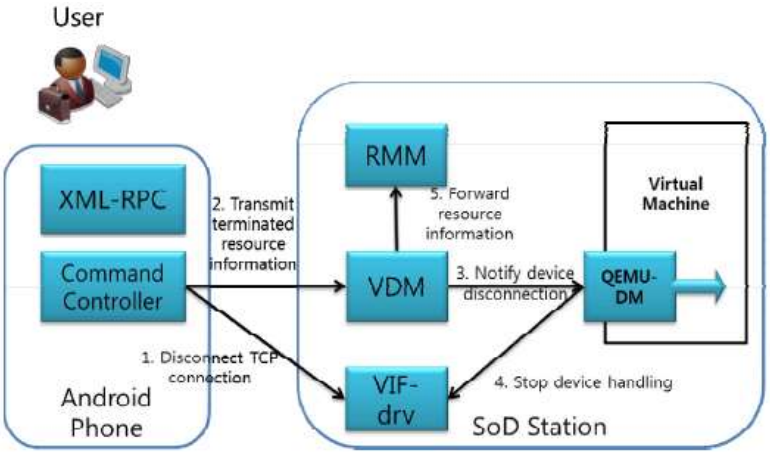

Figure 9: Resource disconnection after stopping program

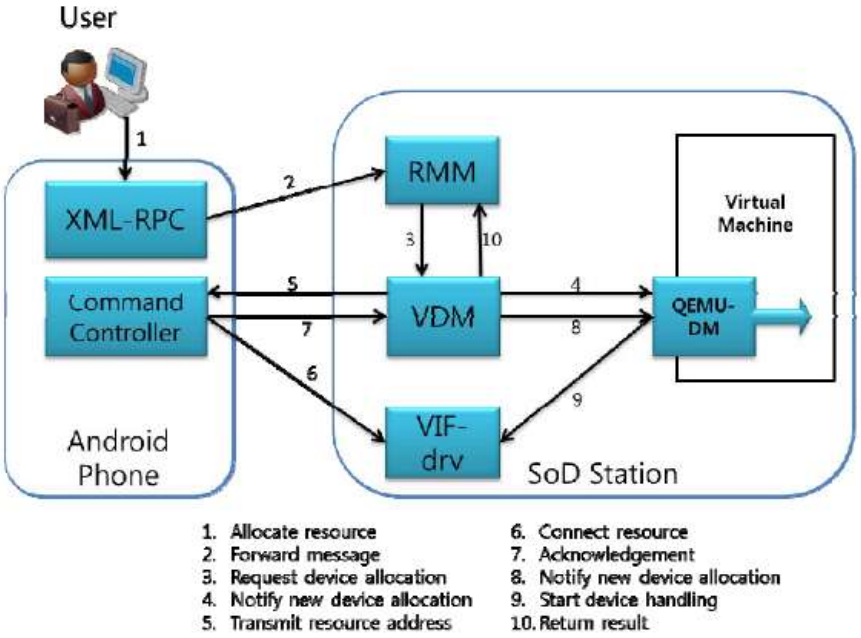

Figure 10: Operating procedures for resource allocation

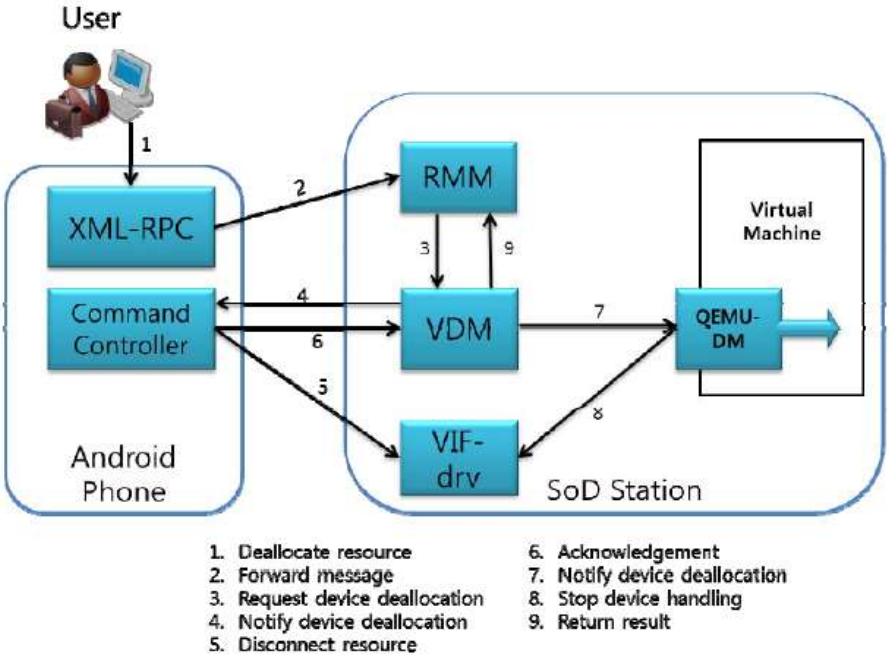

Figure 11: Operating procedures for resource deallocation

2.4 Android User Interface Design

An application should not only be functional but must also have a good GUI as an application with good GUI design will achieve better user's demand and sold better [12]. There are three main steps involved when developing a GUI for Android application.

First, target users will be located and core users' demands will be gathered by the requirement analyst. He/she then will transform the demands into basic functionality. Secondly, based on the ideas gathered and functions analyzed, the UI designer will then develop a GUI. Lastly, software engineers will translated and implement the design with eXtensible Markup Language (XML) coding.

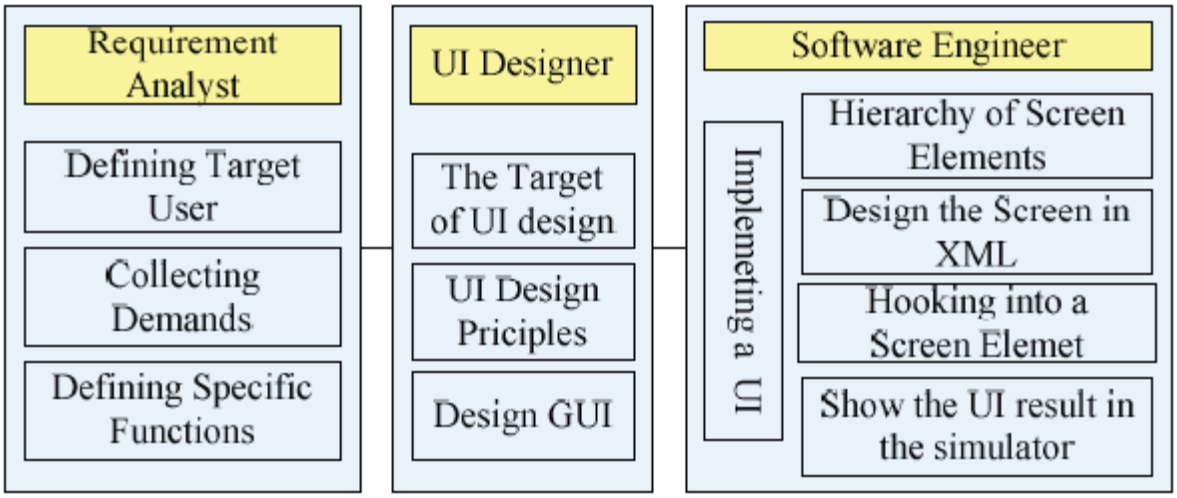

Figure 12: The design procedure of Android smartphone user interface

The GUI design will be implemented by the system engineers. As being mentioned earlier, Activity class control the function of application but the Activity itself does not have presence on the screen so the engineers need to work views and view groups. View is basically the rectangular area on the screen that can be drawn to, handle clicks or any other interaction events [12] meanwhile a ViewGroup is merely a container that holds multiple View child together. Activity can be defined as a tree-structured UI by using a tree of view and view group nodes.

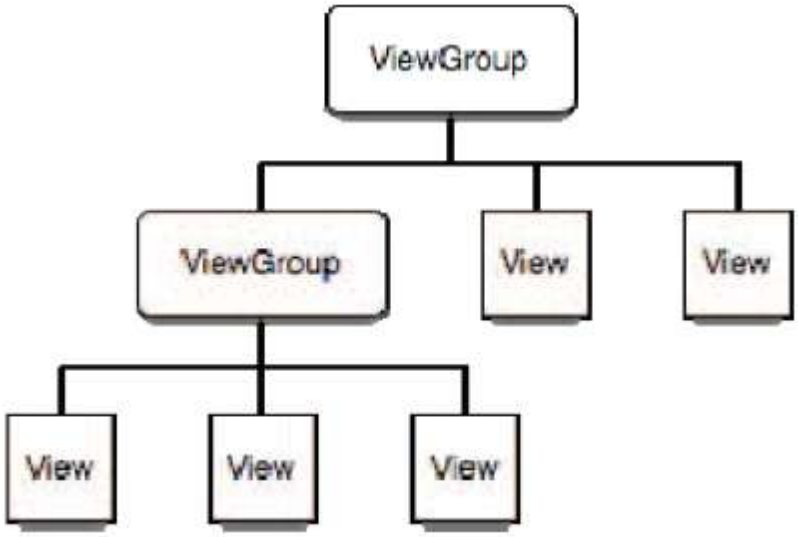

Figure 13: A Tree-structured UI

Once the GUI is structured, it can then be designed in XML using Layout class. Layout class is basically a special class of ViewGroup that is used to display the widgets. There are several types of Layout, such as LinearLayout which aligns the children in either vertically or horizontally. For more flexible aligning and complex UI, RelativeLayout can be used. AbsoluteLayout is used to specify the exact location of children using x/y coordinates.

CHAPTER 3

METHODOLOGY

3.1 Research Methodology

In order to achieve the main objective of this project, the goals for two sub objectives highlighted in the earlier part need to be accomplished. To develop a fully functional Android application, brief research and literature review needs to be done on the selected papers that concentrate on the Android application development including GUI design and sensor utilization. The relevancy between selected papers and project objectives need to be taken into account to ensure the credibility of the project.

Basic application framework structure will then be drawn and coded using Java language in the provided Android SDK. The source code and brief research regarding the development are carried out on several resources such as books and also internet. The basic functionality of the application will be tested during the beta-testing stage, and once it is achieved, the overall UI will be designed and improved.

3.2 Flow Chart

The following flow chart explains the methodology in executing the project.

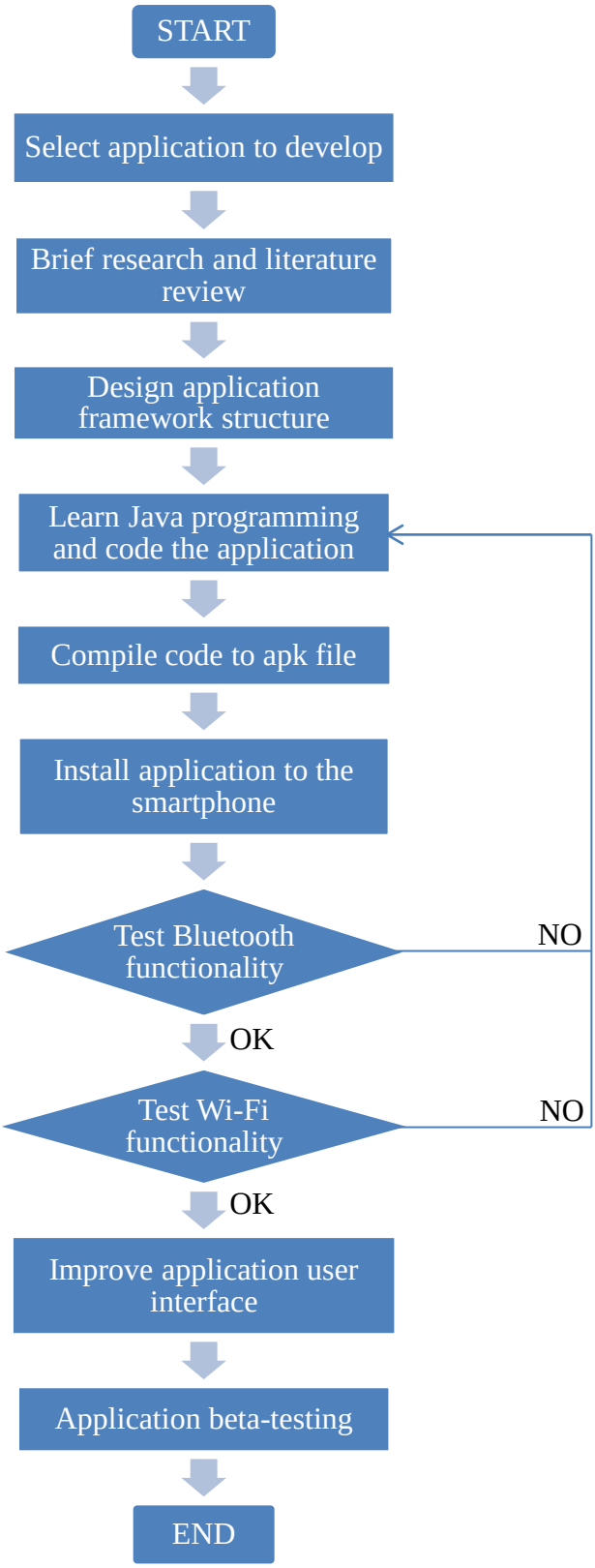

Figure 14: Project methodology

3.3 Project Task

- **Select application to develop**
  Based on background study and problem statement, one application to be developed using Android SDK is decided.
- **Brief research and literature review**
  Once application has been decided, brief research and literature review is done to gain more knowledge and information regarding the project to determine the feasibility of the project.
- **Design application framework structure**
  The basic application structure will be designed where all the coding will be applied based on the design decided.
- **Learn Java programming and code the application**
  Since Android application is coded using Java language, all coding is written in Java after basic Java programming is learnt.
- **Compile code to apk file**
  Once the coding is done, it will then be compiled to apk file to make sure this file is installable to the Android-based smartphones.
- **Install application to the smartphone**
  The apk file will be installed to the smartphone for preliminary testing.
- **Test Bluetooth functionality**
  Bluetooth connection between smartphone and connected personal computer/laptop is verified to make sure it is working properly.
- **Test Wi-Fi functionality**
  Once Bluetooth is verified working correctly, Wi-Fi module is checked to make sure that the phone is able to connect to the selected computer/laptop via WLAN.
- **Improve application user interface**
  Once all the intended functionality for the application is working correctly, the overall GUI will be improved for better user experience.
- **Application beta-testing**
  This is the final stage of the project where the beta-testing of the application occurs. The application will be distributed among few Android-based smartphone users to test its functionality.

3.4 Gantt Chart and Milestones

In order to effectively conduct the project, a Gantt chart consisted of two semesters duration has been constructed.

Table 2: Gantt chart for Final Year Project I

ACTIVITIES	FINAL YEAR PROJECT I													
	WEEK NO.													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Select application to develop														
Brief research and literature review														
Design application framework structure														
Learn JAVA language and code the application														
Compile to apk and install to the smartphone for preliminary testing														
Report writing														

Table 3: Milestones of the Final Year Project I

Activities	Date
Completion of application selection	Week 1
Completion of brief research and literature review	Week 5
Completion of application framework structure design	Week 6
Completion of application coding using Java	Week 11
Completion for application compilation and installation to smartphones for preliminary testing	Week 14

Table 4: Gantt chart for Final Year Project II

ACTIVITIES	FINAL YEAR PROJECT II													
	WEEK NO.													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Improve application user interface														
Application beta-testing														
Report writing														

Table 5: Milestones of the Final Year Project II

Activities	Date
Completion of application user interface development	Week 8
Completion of application beta testing	Week 14

3.5 Tools

The main software used for Android application development is Android SDK as it provides necessary tools and APIs to build the application from scratch using Java programming language. The other tools that are needed are Android-based smartphones and a personal computer/laptop with Bluetooth and Wi-Fi technology.

3.6 Project Schedule

The planned schedule for Final Year Project I and Final Year Project II are as follows.

Table 6: Project schedule for Final Year Project I

Title selection	Week 1
Extended proposal	Week 6
Proposal defense and progress evaluation	Week 9
Draft report	Week 13
Final report	Week 14

Table 7: Project schedule for Final Year Project II

Pre-EDX	Week 8
Draft report	Week 13
Final report	Week 14
VIVA	Week 15

CHAPTER 4

RESULT AND DISCUSSION

4.1 Results

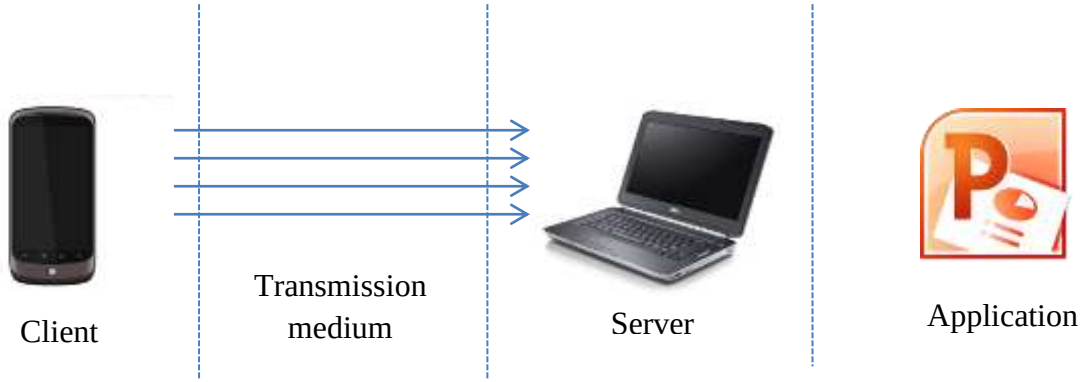

Figure 15: Application workflow

The working mechanism for the application can be divided to four basic parts. The first is the client which contains the smartphone that have been installed with the proposed application. The application (client) will interact with the remote computer (server) via the transmission medium. On the server, data that had been passed from the smartphone will be interpreted and executed to control the application.

When the application is first started, the application will create an instance of DVM to run its own functionality. It will then build its interface and function based on the framework that is already available in the phone including activity manager, view system and resource manager. At the same time, it will load up necessary drivers from Linux kernel such as Bluetooth driver and Wi-Fi driver. The application can be connected to the remote computer via Bluetooth or Wi-Fi depending on the user needs.

Figure 16: Main screen of the application

Figure 17: Connection selection in the application

When the user chooses to connect the smartphone via Bluetooth, firstly the application will check for the availability of Bluetooth adapter. If the application detects no Bluetooth adapter available, the program would not proceed. Next, the application will check whether the Bluetooth is already turned on or not. If it is not, then the application will request the permission from the user to turn on the Bluetooth.

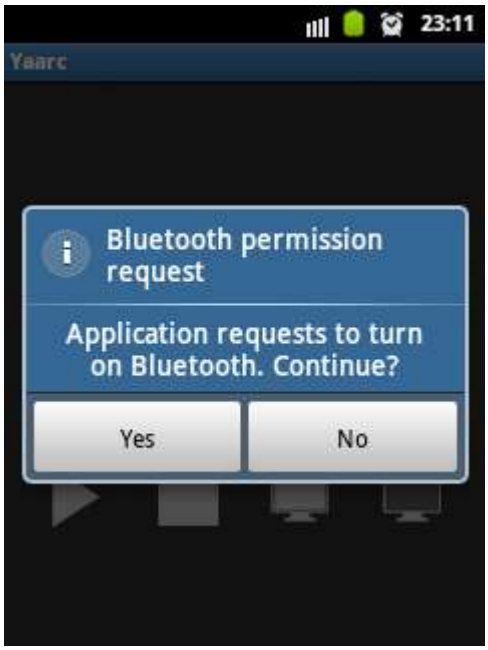

Figure 18: Application requests to turn on the Bluetooth

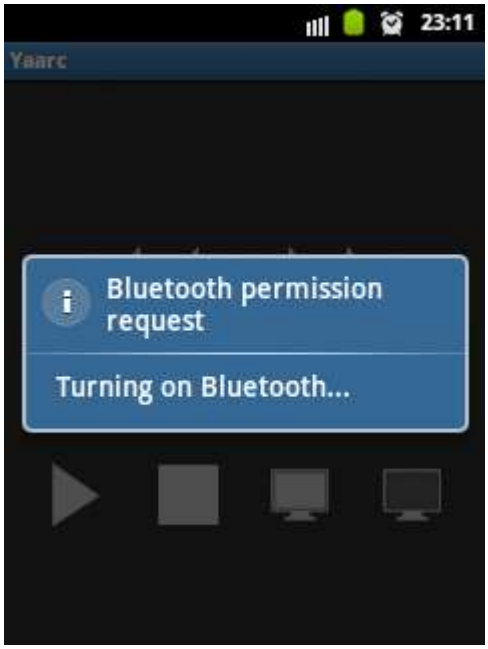

Figure 19: Application turns on the Bluetooth once permission is given

The application then can search for available devices in range and retrieve the already paired devices with the smartphones. The application will open up a socket to establish connection with the selected remote device.

Figure 20: Application lists the available devices that can be connected with

If the user chooses to connect the devices via Wi-Fi, the user will then be prompted to enter the IP address of the remote computer. For a faster connection, it is recommended that the smartphone and the computer to be connected under a same Wi-Fi network.

Figure 21: Application will prompt for IP address

When the user press the connect button, the application will validate the IP address to make sure that it is valid. The value of IP address should be in the range of 0-255 only. It will then try to establish the connection to the server port as it had been predefined in the application.

The application will then redirect the user to the control screen no matter of which type of connection is used. There are basically six buttons in the control screen which are Start, Stop, Next, Previous, Black Screen and White Screen.

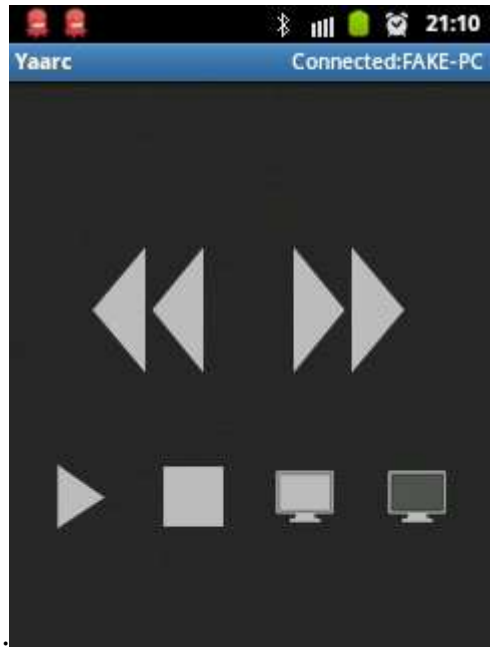

Figure 22: Control screen of the application

Each button represents an integer value that will be sent to the server. As example, when the user press the Start button, the application will send the value of 1 to the server where on the server, the data then will be interpreted for the appropriate response.

The server application must be running at the remote computer so that the data sent by the smartphone will be received and handled. As being said before, each integer value will represent a command to the Microsoft PowerPoint. When the server received the data, it will then match the value with the respective command. The

command is executed by the Java Robot API where it can emulate the keyboard press. Since the Microsoft PowerPoint has the predefined keyboard shortcut, it will react to the different emulated keyboard press generated by the server application. As an example, the value 1 that is sent by the application when the user clicks the Start button is received by the server. It will then be matched with the appropriate command, which is to emulate "F5" key press. The built-in keyboard shortcut of Microsoft PowerPoint will start the presentation.

Table 8: List of available commands in the application

Button	**Integer**	**Keyboard press**
Start	1	F5
Next	2	Right arrow
Previous	3	Left arrow
Stop	4	Escape
Black screen	5	B
White screen	6	W

Figure 23: Data sent by the application is being interpreted on the server

As being described earlier, an activity consists of the user interface and all necessary functions related to it. The proposed application can be breakdown to three core activities, which are BluetoothScreen.java, BluetoothCommandService.java, and WifiScreen.java. For the server application on the remote computer, the core activity is ProcessConnectionThread.java.

The simplified program flow for each activity can be seen in the following Figure 24, Figure 25, Figure 26 and Figure 27.

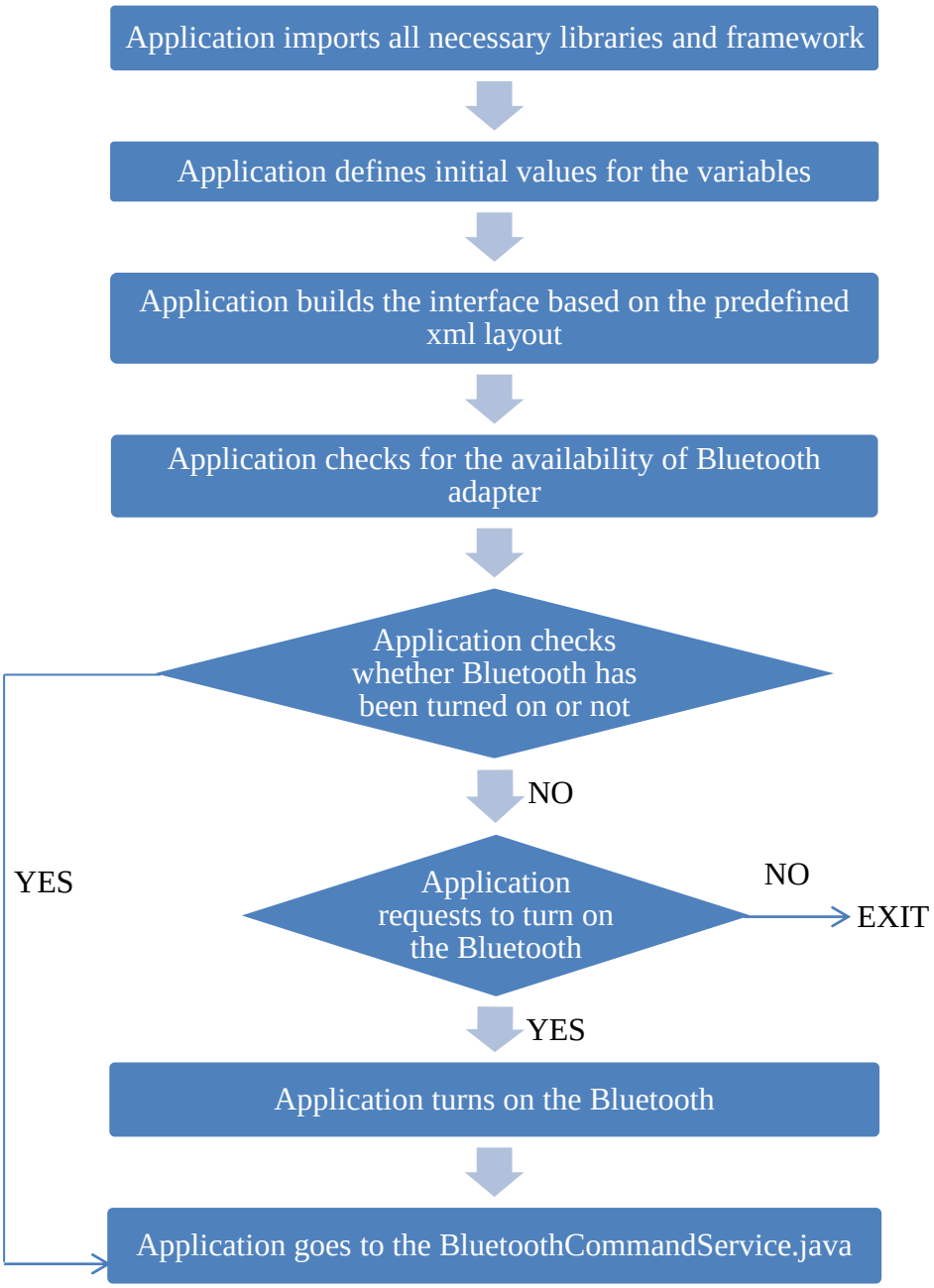

Figure 24: BluetoothScreen.java flow

BluetoothCommandService.java handles all the backend function of the Bluetooth functionality of the application including creating the Bluetooth socket and initiating the input and output streams of the Bluetooth connection.

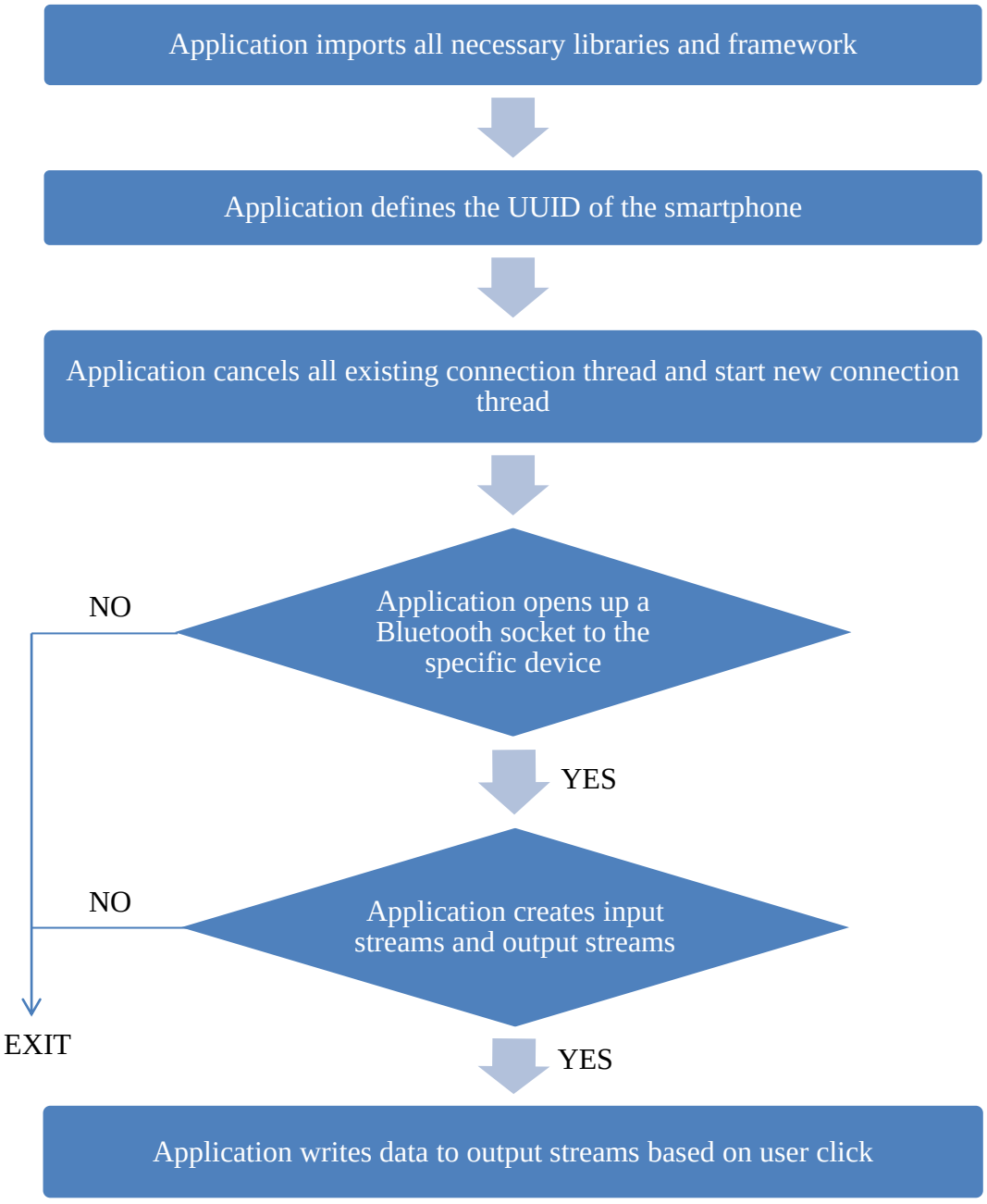

Figure 25: BluetoothCommandService.java flow

WifiScreen.java handles the Wi-Fi functionality of the application including defining the output port and set up the IP address of the remote server on the computer that the application wishes to connect to.

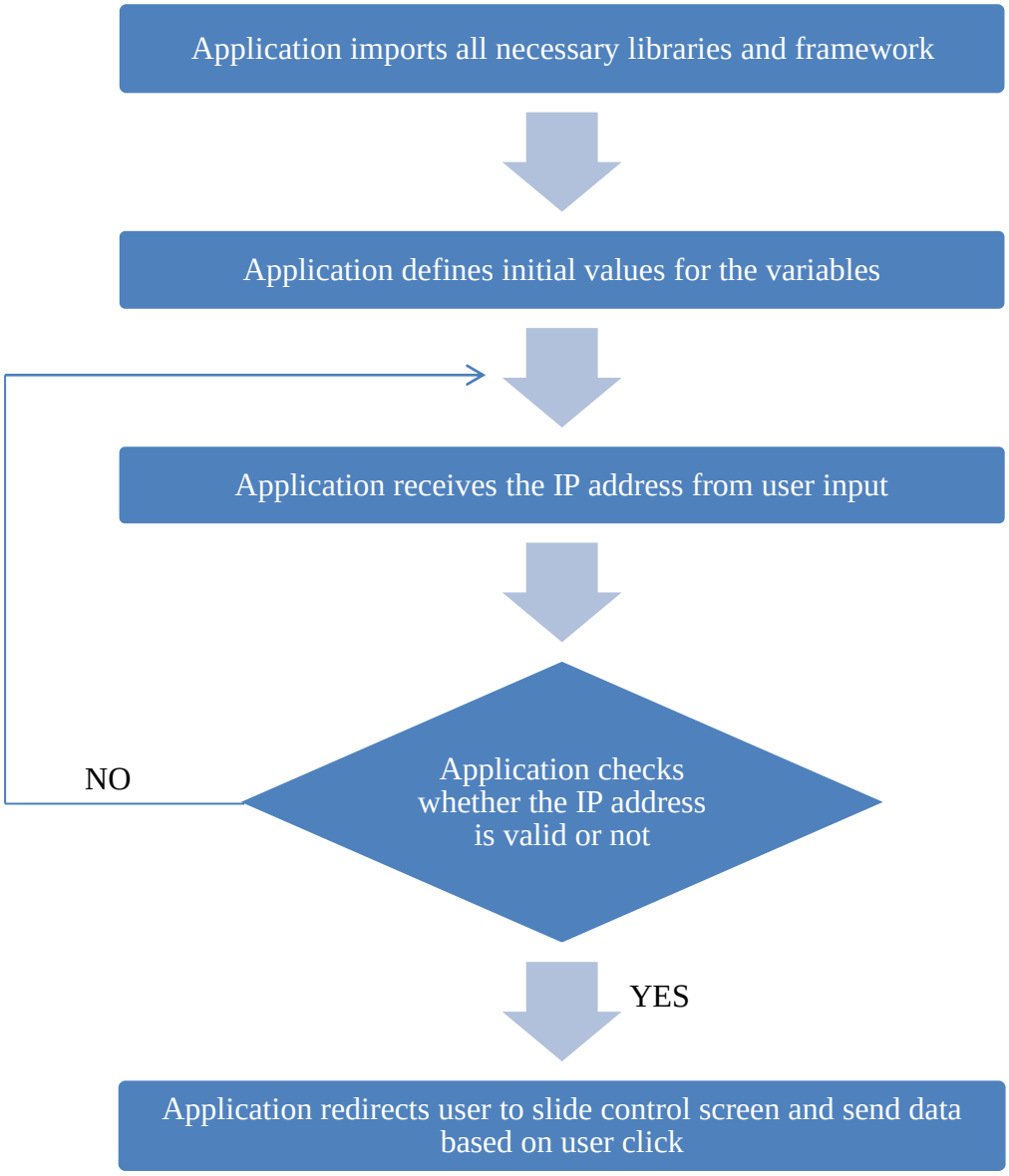

Figure 26: WifiScreen.java flow

ProcessConnectionThread.java handles the server part of the application. It receives the data sent by the smartphones and translates it to the appropriate command to the presentation control.

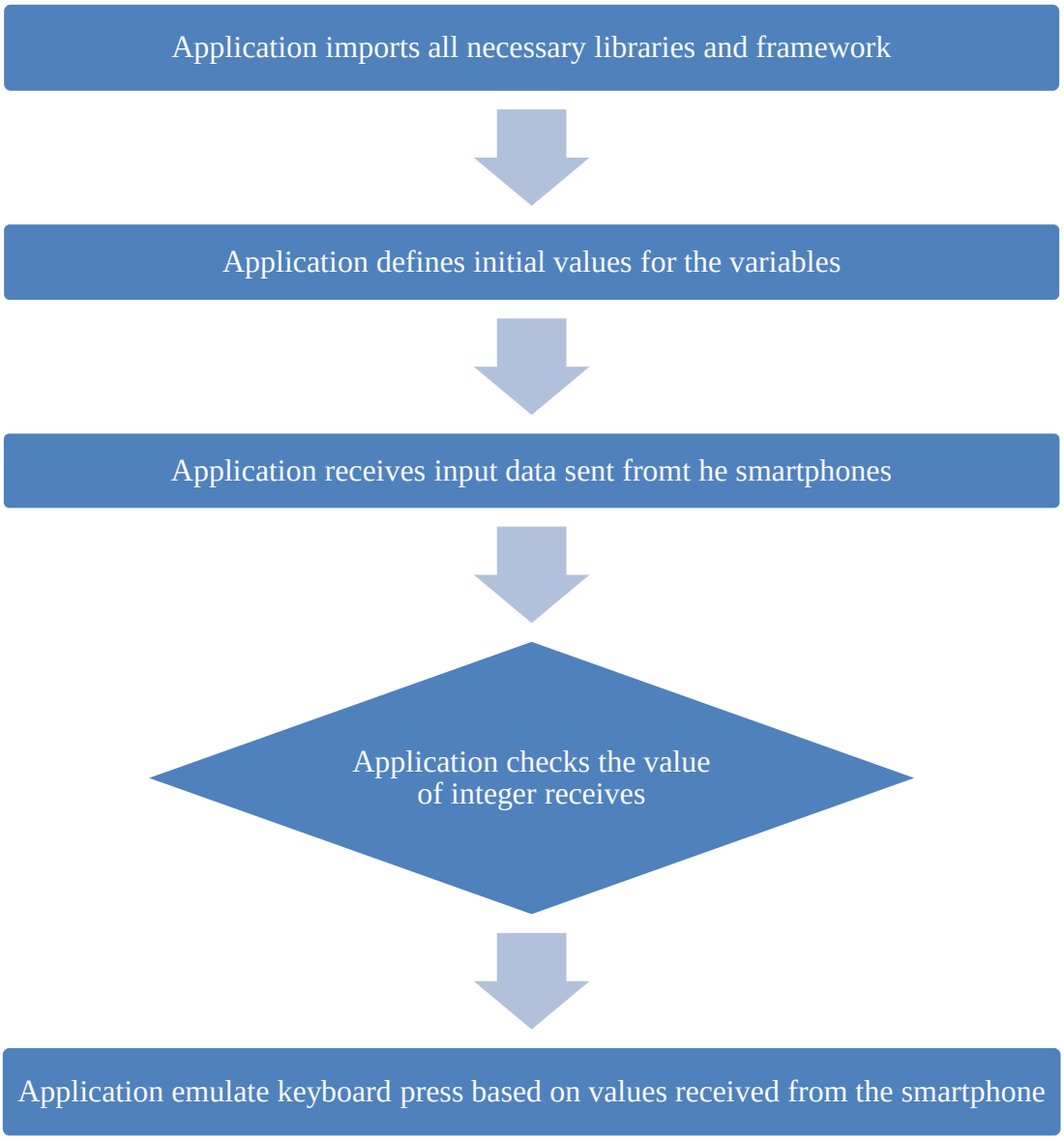

Figure 27: ProcessConnectionThread.java flow

4.2 Discussion

From the result that is obtained, the application can be run in the Android-based smartphone with Android version 2.3 or higher. The application can be started instantly and received the user input based on the touch screen. The application will first ask the user permission to turn on the Bluetooth before scanning for available devices in the range.

The implementation of the client-side of the application for the Bluetooth part can be seen in the Appendix A meanwhile the code for Bluetooth device listing is available in the Appendix B. The Java code of the server-side implementation for Bluetooth part is available in Appendix C.

As for the Wi-Fi, full source code of the client-side implementation can be seen in Appendix D meanwhile the server-side implementation itself can be seen in Appendix E.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

Android SDK enables the users to write their own application using Java programming language. The transparency of Android OS is useful in a sense that it allows the user to access available hardware and sensors equipped with the phone provided appropriate security permission is given. The application can be coded to check for the hardware availability in such smartphones then further utilize it to interact with user input and communicate with remote devices.

The application coded successfully connects to the remote computer via Bluetooth and Wi-Fi. Depending on user interaction, the application can control the flow of slide presentation by manipulating the keyboard press on the remote computer to trigger the built-in keyboard shortcut of such application.

5.2 Recommendation

Through this project, Bluetooth connection can be initiated and the application on the Android-based smartphone can be connected to a remote desktop or laptop via Bluetooth and Wi-Fi. However, there are plenty of rooms for improvements. This project can be expanded to remote controlling other application such as media player, or imitating keyboard and touchpad gesture. Furthermore, the application's user interface can be vastly improved to give a better user experience when users are using the application.

REFERENCES

[1] Amalfitano, D.; Fasolino, A.R.; Tramontana, P.; "A GUI Crawling-Based Technique for Android Mobile Application Testing," Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference, pp.252-261, 21-25 March 2011

[2] Pan Yong-Cai; Liu Wen-chao; Li Xiao; "Development and Research of Music Player Application Based on Android," Communications and Intelligence Information Security (ICCIIS), 2010 International Conference, pp.23-25, 13-14 Oct. 2010

[3] Torunski, E.; El Saddik, A.; Petriu, E.; "Gesture recognition on a mobile device for remote event generation," Multimedia and Expo (ICME), 2011 IEEE International Conference, pp.1-6, 11-15 July 2011

[4] Gavalas, D.; Economou, D.; "Development Platforms for Mobile Applications: Status and Trends," Software, IEEE , vol.28, no.1, pp.77-86, Jan.-Feb. 2011

[5] Chiu-Chiao Chung; Ching Yuan Huang; Shiau-Chin Wang; Cheng-Ming Lin; "Bluetooth-Based Android Interactive Applications for Smart Living," Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference, pp.309-312, 16-18 Dec. 2011

[6] Wu Shyi-Shiou; Wu Hsin-Yi; "The Design of an Intelligent Pedometer Using Android," Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference, pp.313-315, 16-18 Dec. 2011

[7] Jianye Liu; Jiankun Yu; "Research on Development of Android Applications," Intelligent Networks and Intelligent Systems (ICINIS), 2011 4th International Conference, pp.69-72, 1-3 Nov. 2011

[8] OL. Google Android Developers, Android Develop Guide, http://developer.android.com/guide/topics/fundamentals.html

[9] Kuna, M.; Kolaric, H.; Bojic, I.; Kusek, M.; Jezic, G.; "Android/OSGi-based Machine-to-Machine context-aware system," Telecommunications (ConTEL), Proceedings of the 2011 11th International Conferenc, pp.95-102, 15-17 June 2011

[10] M2M Communications website, http://www.m2mcomm.com/about/what-is-m2m/index.html

[11] Kyuchang Kang; Kiryong Ha; Jeunwoo Lee; "Android-based SoD client for remote presentation," Advanced Communication Technology (ICACT), 2011 13th International Conference, pp.1162-1167, 13-16 Feb. 2011

[12] Maoqiang Song; Haiyan Song; Xiangling Fu; "Methodology of user interfaces design based on Android," Multimedia Technology (ICMT), 2011 International Conference, pp.408-411, 26-28 July 2011

APPENDICES

APPENDIX A

Below is the Java code for the client-side Bluetooth implementation that is used to turn on the Bluetooth, search the available device and initiate the connection.

```
package com.android.yaarc;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.Window;
import android.widget.TextView;
import android.widget.Toast;

public class BluetoothScreen extends Activity {

      private TextView yTitle;
      private static final int REQUEST_CONNECT_DEVICE = 1;
      private static final int REQUEST_ENABLE_BT = 2;
      public static final int MESSAGE_STATE_CHANGE = 1;
      public static final int MESSAGE_READ = 2;
      public static final int MESSAGE_WRITE = 3;
      public static final int MESSAGE_DEVICE_NAME = 4;
      public static final int MESSAGE_TOAST = 5;
```

```
	public static final String DEVICE_NAME = "device_name";

	public static final String TOAST = "toast";

	private String yConnectedDeviceName = null;

	private BluetoothAdapter yBluetoothAdapter = null;

	private BluetoothCommandService yCommandService = null;


	@Override

	public void onCreate(Bundle savedInstanceState) {

		super.onCreate(savedInstanceState);

		requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);

		getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
R.layout.custom_title);

		setContentView(R.layout.bluetooth);


		yTitle = (TextView) findViewById(R.id.title_left_text);

		yTitle.setText(R.string.app_name);

		yTitle = (TextView) findViewById(R.id.title_right_text);

		yBluetoothAdapter= BluetoothAdapter.getDefaultAdapter();


		if (yBluetoothAdapter == null) {

			Toast.makeText(this, "Bluetooth is not available
for your device", Toast.LENGTH_LONG).show();

			finish();

			return;

		}	}


	@Override

	protected void onStart() {

		super.onStart();


		if (!yBluetoothAdapter.isEnabled()) {

			Intent enableBTIntent = new Intent
(BluetoothAdapter.ACTION_REQUEST_ENABLE);

			startActivityForResult(enableBTIntent,
REQUEST_ENABLE_BT);
```

```
            }else {
                  if (yCommandService == null)
                        setupCommand();   }}


      @Override
      protected void onResume() {
            super.onResume();


            if (yCommandService != null) {
                  if (yCommandService.getState() ==
BluetoothCommandService.STATE_NONE) {
                        yCommandService.start();}}}


      private void setupCommand() {
            yCommandService = new BluetoothCommandService(this,
yHandler);  }


      @Override
      protected void onDestroy() {
            super.onDestroy();


            if (yCommandService != null)
                  yCommandService.stop(); }


      private void ensureDiscoverable() {
            if (yBluetoothAdapter.getScanMode() !=
BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
                  Intent discoverableBTIntent = new Intent
(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);


      discoverableBTIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERA
BLE_DURATION, 300); //5minutes
                  startActivity(discoverableBTIntent);}}


      private final Handler yHandler = new Handler() {
            @Override
```

```
            public void handleMessage(Message msg) {
                  switch (msg.what) {

                  case MESSAGE_STATE_CHANGE:
                        switch (msg.arg1) {
                        case
BluetoothCommandService.STATE_CONNECTED:

      yTitle.setText(R.string.title_connected_to);
                              yTitle.append(yConnectedDeviceName);
                        break;
                        case
BluetoothCommandService.STATE_CONNECTING:

      yTitle.setText(R.string.title_connecting);
                        break;
                        case BluetoothCommandService.STATE_LISTEN:
                        case BluetoothCommandService.STATE_NONE:

      yTitle.setText(R.string.title_not_connected);
                        break;
                        }
                  break;

                  case MESSAGE_DEVICE_NAME:
                        yConnectedDeviceName =
msg.getData().getString(DEVICE_NAME);
                        Toast.makeText(getApplicationContext(),
"Connected to " + yConnectedDeviceName, Toast.LENGTH_SHORT).show();
                  break;

                  case MESSAGE_TOAST:
                        Toast.makeText(getApplicationContext(),
msg.getData().getString(TOAST), Toast.LENGTH_SHORT).show();
                  break;
                  }
            }};
```

```
      public void onActivityResult(int requestCode, int resultCode,
Intent data) {

            switch(requestCode) {



            case REQUEST_CONNECT_DEVICE:

                  if (resultCode == Activity.RESULT_OK) {

                        String address =
data.getExtras().getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);

                        BluetoothDevice device =
yBluetoothAdapter.getRemoteDevice(address);

                        yCommandService.connect(device);

                  }

                  break;



            case REQUEST_ENABLE_BT:

                  if (resultCode == Activity.RESULT_OK) {

                        setupCommand();

                  }

                  else {

                        Toast.makeText(this,
R.string.bt_not_enabled_leaving, Toast.LENGTH_SHORT).show();

                        finish();

                  }}}



      @Override

      public  boolean onCreateOptionsMenu(Menu menu) {

            MenuInflater inflater = getMenuInflater();

            inflater.inflate(R.menu.option_menu, menu);

            return true;

      }



      @Override public boolean onOptionsItemSelected(MenuItem item)
{

            switch (item.getItemId()) {
```

```
            case R.id.scan:
                  Intent connectBTServer = new Intent(this,
DeviceListActivity.class);
                  startActivityForResult(connectBTServer,
REQUEST_CONNECT_DEVICE);
                  return true;


            case R.id.discoverable:
                  ensureDiscoverable();
                  return true;}
            return false;     }


      @Override
      public boolean onKeyDown(int keyCode, KeyEvent event) {
            if (keyCode == KeyEvent.KEYCODE_VOLUME_UP) {

      yCommandService.write(BluetoothCommandService.VOL_UP);
                  return true;            }
            else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN) {

      yCommandService.write(BluetoothCommandService.VOL_DOWN);
                  return true;            }
            return super.onKeyDown(keyCode, event);   }}
```

APPENDIX B

Below is the Java code that is used to search for available device, retrieved the paired devices of the phone and list the devices to the user.

```
package com.android.yaarc;

import java.util.Set;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

public class DeviceListActivity extends Activity {

      private static final String TAG = "DeviceListActivity";
      private static final boolean D = true;
      public static String EXTRA_DEVICE_ADDRESS = "device_address";
      private BluetoothAdapter yBtAdapter;
```

```
	private ArrayAdapter<String> yPairedDevicesArrayAdapter;
	private ArrayAdapter<String> yNewDevicesArrayAdapter;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
	requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
		setContentView(R.layout.device_list);
		setResult(Activity.RESULT_CANCELED);

		Button scanButton = (Button)
findViewById(R.id.button_scan);
		scanButton.setOnClickListener(new OnClickListener() {
			public void onClick(View v) {
				doDiscovery();
				v.setVisibility(View.GONE);   }});

		yPairedDevicesArrayAdapter = new
ArrayAdapter<String>(this, R.layout.device_name);
		yNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
R.layout.device_name);

		ListView pairedListView = (ListView)
findViewById(R.id.paired_devices);
	pairedListView.setAdapter(yPairedDevicesArrayAdapter);
	pairedListView.setOnItemClickListener(yDeviceClickListener);

		ListView newDevicesListView = (ListView)
findViewById(R.id.new_devices);
	newDevicesListView.setAdapter(yNewDevicesArrayAdapter);
	newDevicesListView.setOnItemClickListener(yDeviceClickListener
);
		IntentFilter filter = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
		this.registerReceiver(yReceiver, filter);
```

```
            filter = new
IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);

            this.registerReceiver(yReceiver, filter);

            yBtAdapter = BluetoothAdapter.getDefaultAdapter();


            Set<BluetoothDevice> pairedDevices =
yBtAdapter.getBondedDevices();


            if (pairedDevices.size() > 0) {


      findViewById(R.id.title_paired_devices).setVisibility(View.VIS
IBLE);

                  for (BluetoothDevice device : pairedDevices) {

                  yPairedDevicesArrayAdapter.add(device.getName() +
"\n" + device.getAddress());}}


                  else {

                        String noDevices =
getResources().getText(R.string.none_paired).toString();

                        yPairedDevicesArrayAdapter.add(noDevices);
            }}


      @Override

      protected void onDestroy() {

            super.onDestroy();

                  if (yBtAdapter != null) {

                  yBtAdapter.cancelDiscovery(); }

            this.unregisterReceiver(yReceiver); }


      private void doDiscovery() {

            if (D) Log.d(TAG, "doDiscovery()");


            setProgressBarIndeterminateVisibility(true);

            setTitle(R.string.scanning);
```

```
      findViewById(R.id.title_new_devices).setVisibility(View.VISIBL
E);


            if (yBtAdapter.isDiscovering()) {

                  yBtAdapter.cancelDiscovery(); }

            yBtAdapter.startDiscovery();  }


      private OnItemClickListener yDeviceClickListener = new
OnItemClickListener() {

            public void onItemClick (AdapterView<?> av, View v, int
arg2, long arg3) {

                  yBtAdapter.cancelDiscovery();


                  String info = ((TextView) v).getText().toString();

                  String address = info.substring(info.length() -
17);


                  Intent intent = new Intent();

                  intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

                        setResult(Activity.RESULT_OK, intent);

                  finish();}};


            private final BroadcastReceiver yReceiver = new
BroadcastReceiver() {

            @Override

            public void onReceive(Context context, Intent intent) {

                  String action = intent.getAction();


                  if (BluetoothDevice.ACTION_FOUND.equals(action)) {

                        BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

                        if (device.getBondState() !=
BluetoothDevice.BOND_BONDED) {


      yNewDevicesArrayAdapter.add(device.getName() + "\n" +
device.getAddress());}}
```

```
                  else if
(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {


      setProgressBarIndeterminateVisibility(false);

                        setTitle(R.string.select_device);

                        if (yNewDevicesArrayAdapter.getCount() == 0)
{

                              String noDevices =
getResources().getText(R.string.none_found).toString();


      yNewDevicesArrayAdapter.add(noDevices);

                        }}}};}
```

APPENDIX C

Below is the JAVA code that is implemented on the server-side where it will handle all the incoming input data and interpret it to allow appropriate response action for every possible input data.

```
package com.apps.yaarc.server;
import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.io.InputStream;
import javax.microedition.io.StreamConnection;

public class ProcessConnectionThread implements Runnable{

	private StreamConnection yConnection;
	private static final int EXIT_CMD = -1;
	private static final int KEY_F5 = 1;
	private static final int KEY_RIGHT = 2;
	private static final int KEY_LEFT = 3;
	private static final int KEY_ESC = 4;
	private static final int KEY_B = 5;
	private static final int KEY_W = 6;

	public ProcessConnectionThread(StreamConnection connection) {
		yConnection = connection;}

	@Override
	public void run() {
		try {
			InputStream inputStream =
yConnection.openInputStream();
			System.out.println("Waiting for input");
			while(true) {
```

```
                int command = inputStream.read();

                if (command == EXIT_CMD) {
                      System.out.println("Exit");
                      break;}
                processCommand(command);}}
          catch (Exception e) {
                e.printStackTrace();}}

private void processCommand(int command) {
          try {
                Robot robot = new Robot();

                switch (command) {
                case KEY_F5:
                      robot.keyPress(KeyEvent.VK_F5);
                      robot.keyRelease(KeyEvent.VK_F5);
                      System.out.println("Start presentation");
                break;
                case KEY_RIGHT:
                      robot.keyPress(KeyEvent.VK_RIGHT);
                      robot.keyRelease(KeyEvent.VK_RIGHT);
                      System.out.println("Next slide");
                break;
                case KEY_LEFT:
                      robot.keyPress(KeyEvent.VK_LEFT);
                      robot.keyRelease(KeyEvent.VK_LEFT);
                      System.out.println("Previous slide");
                break;
                case KEY_ESC:
                      robot.keyPress(KeyEvent.VK_ESCAPE);
                      robot.keyRelease(KeyEvent.VK_ESCAPE);
                      System.out.println("Stop presentation");
```

```
            break;
            case KEY_B:
                  robot.keyPress(KEY_Event.VK_B);
                  robot.keyRelease(KEY_Event.VK_B);
                  System.out.println("Black screen");
            break;
            case KEY_W:
                  robot.keyPress(KEY_Event.VK_W);
                  robot.keyRelease(KEY_Event.VK_W);
                  System.out.println("White screen");
            break;
            }}
      catch (Exception e) {
            e.printStackTrace();
      }}}
```

APPENDIX D

Below is the JAVA code that is implemented on the client-side for Wi-Fi implementation where it will receive the input from the user and send the appropriate data to the server running in the remote computer.

```
package com.android.yaarc;
import java.net.InetAddress;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import com.illposed.osc.OSCMessage;
import com.illposed.osc.OSCPort;
import com.illposed.osc.OSCPortOut;

public class SlideControl extends Activity {

	public static final int EXIT_CMD = -1;
	public static final int START_SLIDE = 1;
	public static final int NEXT_SLIDE = 2;
	public static final int PREV_SLIDE = 3;
	public static final int STOP_SLIDE = 4;
	public static final int BLACK_SCREEN = 5;
	public static final int WHITE_SCREEN = 6;
	public static final String TAG= "yaarcWifi";
	private OSCPortOut sender;

	public SlideControl() {
		super();
	}
```

```
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.bluetooth);


		try {
			this.sender = new
OSCPortOut(InetAddress.getByName(Settings.ip),
OSCPort.defaultSCOSCPort());}
		catch (Exception e) {
			Log.d(TAG, e.toString());}


	Button startSlide = (Button) findViewById(R.id.startSlide);
		startSlide.setOnClickListener(new View.OnClickListener()
{
			public void onClick(View v) {
				startSlide();}});


	Button nextSlide = (Button) findViewById(R.id.nextSlide);
		nextSlide.setOnClickListener(new View.OnClickListener()
{
			public void onClick(View v) {
				nextSlide();}});


	Button prevSlide = (Button) findViewById(R.id.prevSlide);
		prevSlide.setOnClickListener(new View.OnClickListener()
{
			public void onClick(View v) {
				prevSlide();}});


	Button stopSlide = (Button) findViewById(R.id.stopSlide);
		stopSlide.setOnClickListener(new View.OnClickListener()
{
			public void onClick(View v) {
				stopSlide();}});
```

```
      Button blackScreen = (Button) findViewById(R.id.blackScreen);

            blackScreen.setOnClickListener(new
View.OnClickListener() {

                  public void onClick(View v) {

                        blackScreen();}});


      Button whiteScreen = (Button) findViewById(R.id.whiteScreen);

            whiteScreen.setOnClickListener(new
View.OnClickListener() {

                  public void onClick(View v) {

                        whiteScreen();}});}


      @Override

      protected void onStart() {

            super.onStart();}


      @Override

      protected void onResume() {

            super.onResume(); }


      @Override

      protected void onDestroy() {

            super.onDestroy();}


      private void startSlide() {

            Object args[] = new Object[2];

            args[0] = 1;

            args[1] = "Start presentation";

            OSCMessage msg = new OSCMessage("/startslide", args);


            try {

                  sender.send(msg); }

            catch (Exception e) {

                  Log.d(TAG, e.toString());}}
```

```
private void nextSlide() {
      Object args[] = new Object[2];
      args[0] = 2;
      args[1] = "Next slide";
      OSCMessage msg = new OSCMessage("/nextslide", args);

      try {
            sender.send(msg); }
      catch (Exception e) {
            Log.d(TAG, e.toString());}}

private void prevSlide() {
      Object args[] = new Object[2];
      args[0] = 3;
      args[1] = "Prev slide";
      OSCMessage msg = new OSCMessage("/prevslide", args);

      try {
            sender.send(msg);}
      catch (Exception e) {
            Log.d(TAG, e.toString());}}

private void stopSlide() {
      Object args[] = new Object[2];
      args[0] = 4;
      args[1] = "Stop slide";
      OSCMessage msg = new OSCMessage("/stopslide", args);

      try {
            sender.send(msg);}
      catch (Exception e) {
            Log.d(TAG, e.toString());}}
```

```
private void blackScreen() {
      Object args[] = new Object[2];
      args[0] = 5;
      args[1] = "Black screen";
      OSCMessage msg = new OSCMessage("/blackscreen", args);

      try {
            sender.send(msg);}
      catch (Exception e) {
            Log.d(TAG, e.toString());}}

private void whiteScreen() {
      Object args[] = new Object[2];
      args[0] = 6;
      args[1] = "White screen";
      OSCMessage msg = new OSCMessage("/whitescreen", args);

      try {
            sender.send(msg);}
      catch (Exception e) {
            Log.d(TAG, e.toString());}}}
```

APPENDIX E

Below is the JAVA code that is implemented on the server-side for the Wi-Fi implementation where it will handle all the incoming input data and interpret it to allow appropriate response action for every possible input data.

```
package com.android.yaarc.wifi.server;

import java.awt.Robot;
import java.net.InetAddress;
import java.awt.event.KeyEvent;
import com.illposed.osc.OSCListener;
import com.illposed.osc.OSCMessage;
import com.illposed.osc.OSCPort;
import com.illposed.osc.OSCPortIn;

public class WaitThread implements Runnable{

	private OSCPortIn receiver;
	private static final int KEY_F5 = 1;
	private static final int KEY_RIGHT = 2;
	private static final int KEY_LEFT = 3;
	private static final int KEY_ESC = 4;
	private static final int KEY_B = 5;
	private static final int KEY_W = 6;

	public WaitThread() {}

	public void run() {
		waitForConnection();}

	private void waitForConnection() {
		try {
```

```
                InetAddress local = InetAddress.getLocalHost();
                if (local.isLoopbackAddress()) {
                      this.receiver             =             new
OSCPortIn(OSCPort.defaultSCOSCPort());}
                else {
                      this.receiver             =             new
OSCPortIn(OSCPort.defaultSCOSCPort());}}
          catch (Exception e) {
                e.printStackTrace();}
          System.out.println("Waiting for input");

          OSCListener listener = new OSCListener() {
                public  void  acceptMessage(java.util.Date  time,
OSCMessage message) {
                      Object[] args = message.getArguments();
                      if (args.length == 2) {

    processCommand(Integer.parseInt(args[0].toString()));}}};
          this.receiver.addListener("/startslide", listener);

          listener = new OSCListener() {
                public  void  acceptMessage(java.util.Date  time,
OSCMessage message) {
                      Object[] args = message.getArguments();
                      if (args.length == 2) {

    processCommand(Integer.parseInt(args[0].toString()));}}};
          this.receiver.addListener("/nextslide", listener);

          listener = new OSCListener() {
                public  void  acceptMessage(java.util.Date  time,
OSCMessage message) {
                      Object[] args = message.getArguments();
                      if (args.length == 2) {

    processCommand(Integer.parseInt(args[0].toString()));}}};
          this.receiver.addListener("/prevslide", listener);
```

```
            listener = new OSCListener() {
                  public  void  acceptMessage(java.util.Date  time,
OSCMessage message) {
                        Object[] args = message.getArguments();
                        if (args.length == 2) {

      processCommand(Integer.parseInt(args[0].toString()));}}};
            this.receiver.addListener("/stopslide", listener);


            listener = new OSCListener() {
                  public  void  acceptMessage(java.util.Date  time,
OSCMessage message) {
                        Object[] args = message.getArguments();
                        if (args.length == 2) {

      processCommand(Integer.parseInt(args[0].toString()));}}};
            this.receiver.addListener("/blackscreen", listener);


            listener = new OSCListener() {
                  public  void  acceptMessage(java.util.Date  time,
OSCMessage message) {
                        Object[] args = message.getArguments();
                        if (args.length == 2) {

      processCommand(Integer.parseInt(args[0].toString()));}}};
            this.receiver.addListener("/whitescreen", listener);


            this.receiver.startListening();}


      private void processCommand(int command) {
            try {
                  Robot robot = new Robot();


                  switch (command) {
                  case KEY_F5:
                        robot.keyPress(KeyEvent.VK_F5);
```

```
                robot.keyRelease(KeyEvent.VK_F5);
                System.out.println("Start presentation");
            break;
            case KEY_RIGHT:
                robot.keyPress(KeyEvent.VK_RIGHT);
                robot.keyRelease(KeyEvent.VK_RIGHT);
                System.out.println("Next slide");
            break;
            case KEY_LEFT:
                robot.keyPress(KeyEvent.VK_LEFT);
                robot.keyRelease(KeyEvent.VK_LEFT);
                System.out.println("Previous slide");
            break;
            case KEY_ESC:
                robot.keyPress(KeyEvent.VK_ESCAPE);
                robot.keyRelease(KeyEvent.VK_ESCAPE);
                System.out.println("Stop presentation");
            break;
            case KEY_B:
                robot.keyPress(KeyEvent.VK_B);
                robot.keyRelease(KeyEvent.VK_B);
                System.out.println("Black screen");
            break;
            case KEY_W:
                robot.keyPress(KeyEvent.VK_W);
                robot.keyRelease(KeyEvent.VK_W);
                System.out.println("White screen");
            break;
            }}
        catch (Exception e) {
            e.printStackTrace();
        }}}
```