ISSN 2278-5612

APPLICATION OF MOBILE GUI DESIGN THEORY TO THE DEVELOPMENT OF AN OPEN SOURCE TOUCHSCREEN SMARTPHONE GUI

Daniel Sooknanan, Ajay Joshi

Department of Electrical and Computer Engineering, University of the West Indies, St. Augustine, Trinidad and Tobago

Daniel.sooknanan@my.uwi.edu

Department of Electrical and Computer Engineering, University of the West Indies, St. Augustine, Trinidad and Tobago

Ajay.joshi@sta.uwi.edu

ABSTRACT

It can be argued that the focal point of every smartphone is its GUI. The Graphical User Interface is the aspect of any touchscreen smartphone with which users directly interact. Of late, the touchscreen smartphone has become ubiquitous. As such there is a myriad of smartphone platforms, each of which boasts a user-friendly GUI in its own accord. This paper outlines the design and development of an open source touchscreen smartphone GUI, implemented on an Embedded Linux development board, which serves as a hypothetical smartphone. It gives an overview of GUI design theory and mobile usability studies and shows the application of this theory in the programming process for an embedded platform. This paper goes in depth into the development process for an ARM-based Linux platofrm. The major outcomes of this study include the successful formulation and design of a hierarchical, touchscreen GUI suitable for a smartphone, as well as successful development and target-specific implementation of this GUI on an Embedded Linux, ARM-based platform, by adopting of an open source philosophy.

Indexing terms/Keywords

Graphical User Interface, Human-Computer Interaction, Mobile Computing, Mobile Environments

Academic Discipline And Sub-Disciplines

Computer Engineering, Software Engineering

SUBJECT CLASSIFICATION

Mobile Computing and Applications, Open Source Tools

TYPE (METHOD/APPROACH)

Analytical study and Investigation, System Development and Design

INTRODUCTION

Of late, with the increasing use of touchscreens in numerous handheld devices, it can easily be said that the touchscreen smartphone is the most commonplace in the digital age. The graphical user interface (GUI), which acts as the primary means of human-machine interaction in touchscreen smartphones, is essential in influencing a user's choice of smartphone OS. The GUI should ideally be usable, flexible and unique.

A graphical user interface is an interface through which users interact with electronic devices, including computers and handheld devices [20]. According to [20], the interface uses icons, menus and other visual/graphic representations to display information and related user controls. The GUI facilitates human-machine interaction, whereby humans can interact with the device and in return, gain meaningful feedback.

This paper will explore the design, development and implementation of an open source smartphone GUI for a hypothetical touchscreen smartphone. For the purpose of this work, an open source GUI is developed and tested on an Embedded Linux development board, the friendlyARM Mini2440. Open source tools are used during development to enhance flexibility and decrease developmental costs. As a result, the final, tested version of the software will be available as open source. It covers the application of mainstream GUI design theory and highlights the programming process on a Linux platform using Qt SDK.

Due to limitations imposed by the hardware platform, the scope of this work will exclude such features as multi-touch capabilities, haptic feedback and automatic screen reorientation. This paper will focus on the software development and hardware implementation aspects of this work.

OVERVIEW OF GUI DESIGN THEORY

In touchscreen smartphone GUIs, user friendliness and ease of use are crucial characteristics. When designing a touchscreen GUI, there are certain factors which designers must consider. The most obvious is the form factor of these devices, i.e. screen size. Mobile screens are relatively small and as such, only critical functions and content should be included, and these should be laid out strategically in the available screen space [22]. Designers should keep in mind the areas where fingers typically come to rest on the device, in particular, the thumbs. A user's thumb can effectively sweep the entire screen, but some regions require extension of the thumb. This is an important factor when considering control layout – frequently used buttons and primary controls should be placed in easy-reach regions (typically at the bottom of the screen) for easy tapping. Placing primary controls in easy-reach areas prevents obstruction of vision by fingers, known as occultation [3]. For the sake of flexibility, an ambidextrous design must be considered. In delivering the same experience to all users, designers should adopt a vertically symmetrical layout, which further simplifies the interface [11].

Another factor designers should consider is the input mechanism - users typically interact with the GUI via fingers. User interface controls should be of adequate size to capture fingertip actions, without preventing user mishits [27]. The minimum size of buttons and other interface elements should be determined by the size of the pad of an adult finger since touchscreen users would prefer to use the pad of their finger rather than the very tip [5, 23]. For icons used to launch applications from a smartphone menu, the ISO/IEC in [10] states that all icon graphics should be displayed with a resolution of atleast 32×32 px. The "iOS Human Interface Guidelines" states that the minimum target size of any control should be 44×44 px. Also, this guide states that the typical size of an icon for a handheld device should be 57×57 px [1]. Microsoft Inc.'s "Windows Phone UI Design and Interaction Guide" suggests that if any UI element is frequently touched, located toward the edge of the screen or difficult to hit, or if the UI element is part of a sequential task, such as a dial pad, it should be larger than 9mm, in order to prevent touch errors [27].

Touchscreen GUIs incorporate various gestures to enhance interactivity and usability [22]. The core touch gestures as outlined in reference [24], which can be implemented on a single-touch surface (such as the resistive touchscreen) include: tap, double tap, drag, flick/swipe, press and hold. Since the device used for implementation imposes the limitation of single touch, multi-touch gestures will be excluded from this discussion. The gestures used during design should depend on the type of application being developed. Within the touchscreen interface environment, the speed and ease of human interaction is heightened. Therefore, the responsiveness of the interface is of utmost importance. A responsive GUI lends to enhanced usability.

Since there is no keypad on the touchscreen smartphone, there must be the inclusion of a virtual keyboard for text input. For textual input, providing a QWERTY keyboard is advised [14]. The de facto standard position for displaying the keyboard is at the bottom of the screen. Also, the size and position of the keys affect accuracy and input speed [13]. When designing the QWERTY software keyboard, the activation area for each interaction element should be as large as the corresponding visual representation to prevent user mishits [14].

DESIGN

This section of the paper outlines the design process of the GUI, based on the relevant design theory and guidelines explored in the previous section. It focuses on the formulation of an interface for a hypothetical smartphone before development and programming. The guidelines mentioned previously will be applied during design.

As stated previously, the GUI developed would be tested and verified on an embedded system which will act as the hypothetical smartphone. As such, the form factor of this device was taken into account during design – a display resolution of 240×320 px. For the sake of flexibility, the suitability to both finger and stylus input was considered. The GUI will be developed solely in the portrait orientation, since automatic screen reorientation cannot occur. Given the resolution

and form factor of the hypothetical smartphone, an area of 240×20 px will be dedicated to status indicators (battery, Wi-Fi, clock, etc.). Since no buttons were used in this design, navigation must be facilitated within the touchscreen GUI. As such, it was decided to use soft buttons to navigate between screens in the GUI.

In the design of a “Home screen”, the primary controls were placed at the bottom of the screen for easy reach. A vertically symmetrical layout of the icons was adopted, allowing for an ambidextrous design. The docked applications on the home screen will consist of the traditional features of a mobile phone: Phone, Messages, Browser and Main Menu. The main menu of the GUI was envisaged as a hub, from which all applications, including those docked on the home screen, could be launched. Essentially, it will consist of icons in a grid layout, strategically placed in convenient locations on-screen. The strategy employed here was to place icons according to frequency of use, i.e. from most used to least used. With the advent of smartphones and the myriad of features they are now capable of performing, there has been a paradigm shift from the traditional feature phone, where its sole purpose was for making calls and SMS messaging.

To determine the frequency of use of applications/features, and by extension, the order in which icons should be placed, mobile usability studies were researched. According to a report by UK mobile operator, O2, a survey of two thousand smartphone users, conducted in June 2012 detailed the average periods of time per day spent using different smartphone features. The results of the survey documented in [15] are shown in Table 1.

Table 1. Average time per day spent using features of smartphones

Activity	Average Time per Day (mins)
Browsing the Internet	24.81
Checking social networks	17.49
Playing games	14.44
Listening to music	15.64
Making calls	12.13
Checking/writing emails	11.10
Test messaging	10.20
Watching TV/Films	9.39
Reading books	9.30
Taking photographs	3.42
Total	128

The full menu system will contain the hierarchy of icons as outlined in the table above. It was decided that all multimedia functions would be grouped and placed in the fourth position of the hierarchy, thus incorporating “Listening to Music” and “Watching TV/Films”. For a GUI to be an efficient means of HMI, the use of icons is imperative. For the menu system implemented, each application will be represented by an icon 44×44 px in size, as per the guidelines stipulated by [1]. Each icon was selected based on the relevance to the application it represented. Several other native menu items not stated in Table 1 were included during implementation so that the menu would further resemble that of a smartphone. These include: Contacts, Calendar, Notes, Clock, Settings and Maps.

3.1 GUI Applications

In order to add functionality and authenticity to the GUI, the interfaces of certain popular smartphone features will be developed as separate applications, with the ability to be launched from the main menu/home screen.

The “Phone” interface (or dialing/calling interface) appears when a user is making a call and is launched by touching the “Phone” icon. It consists of a dial pad containing numbers “1” through “9”, “0”, “*” and “#”. The numbers on the dial pad also contain associated letters as specified by the ITU in [9]. Additionally, this interface contains Call, End and backspace buttons. When a number is dialed, if an error is made, the user can press the backspace button to clear individual digits, or the End button to clear the entire number. If a number is not entered and the End button is pressed, the application closes and returns to the screen which called it. The Call button is only functional if a number is dialed. After dialing, when Call is pressed, the user is taken to the Call Screen, which displays the number previously dialed and provides options typically used during a call, including the abilities to: mute the call, engage speakerphone, put the call on hold and end the call.

The “Messages” application is designed to create and send text messages. For this application, a virtual keyboard has to be included for textual input. A standard QWERTY keyboard, designed according to the guidelines stated previously will be included at the bottom of the screen. The keyboard will consist of numerous push buttons, representing letters of the alphabet, a Spacebar, a backspace button, a Shift button and an additional number/symbol button which changes the QWERTY keyboard to an array of numbers and symbols. Therefore, the dual nature of the virtual keyboard allows for

letters, numbers and symbols to be input. In the interface, a text edit area allows messages to be composed, while a line edit area at the top of the screen allows the user to enter the recipient's phone number, via the soft keyboard.

Since the Internet browser is the most used feature of smartphones (see Table 1), design of a browser interface was included. The browser will consist of a virtual keyboard for entering the web address, an address bar and the functionality to refresh the page, stop a page from loading and navigate forward and backward between pages. After entering the address in the address bar, when viewing the page, the keyboard will be collapsed and the web page will occupy the majority of the screen.

PROGRAMMING AND DEVELOPMENT

The GUI design was materialized by developing open source software for the GUI and further implemented on an Embedded Linux development board. This section details the programming and development stages of the GUI, and includes both software and hardware implementation aspects. Open source development tools will be used to implement the GUI; this takes into account the programming platform used for development, as well as the hardware platform.

4.1 Justification of Hardware Platform

The majority of smartphones today are equipped with ARM cores. Therefore, it was desired to utilize a platform containing an ARM processor, thus yielding minimized energy consumption, high code density, a small core size and power efficiency [8]. Another major advantage of utilizing an ARM-based platform is the increased level of integration in the mass market. Since the GUI developed will be open source, in order to purport the open source philosophy, it was decided that the hardware platform should also be open source. An Embedded Linux target was chosen due to the reduced developmental costs, given its free license, and the flexibility in development.

The FriendlyARM Mini2440 development board, shown in Figure 1, was chosen to emulate the hypothetical smartphone. The Mini2440 is an Embedded Linux Single Board Computer based on the Samsung S3C2440 micro-processor (a member of the ARM9TDMI family) [17, 18]. The device contains a 3.5-inch, 4-wire, TFT LCD resistive touchscreen with a 240×320 px resolution. It uses SDRAM and Flash memory; the 64M NAND Flash memory module serves as the hard disk, being used for storage of system boot, OS kernel, file system, graphical user interface and application programs [12, 26].

Figure 1 – FriendlyARM Mini2440 Development Board

4.2 Justification of Software Platform

4.2.1 Justification of the Application Framework

The Mini2440 is preinstalled with the Linux 2.6.29 kernel. Linux itself is a kernel, although the term can be used to refer to the platform OS. Embedded Linux is the adoption of the Linux distribution in an embedded device, as Linux can be built on various platforms [7]. The GUI will be programmed on top of this kernel. The development board is also pre-installed with a user interface framework and application suite: Qtopia [12]. Qtopia is an embedded application platform/framework designed specifically for Embedded Linux [2]. Although a suitable candidate for development, its use constituted writing applications for an already existing GUI, which detracted from designing and implementing a GUI.

It was decided that Qt would be used to develop the GUI for the touchscreen smartphone. Qt is a comprehensive C++ application development framework for creating cross-platform GUI applications [2]. Qt uses C++ language constructs and syntax with added macros for rich GUI functionality. Qt is available under dual licensing – it can be used to develop both commercial applications and open source programs. The open source version of Qt is licensed under the GNU Lesser General Public License (LGPL) v2.1 [2].

By using the open source version of Qt, all software developed falls under the LGPL v2.1, by default. The typical applications used on a smartphone would be designed and implemented, thereby contributing to the authenticity and originality of the GUI.

4.3 Programming with Qt

Qt takes ordinary C++ classes and integrates them with Qt classes, objects and macros to make code easier to use [21]. The IDE used for development of Qt applications is Qt Creator. In order to develop Qt applications for the hardware

platform, a compatible version of the Qt libraries had to be compiled. It was decided that an older version of Qt libraries, Qt 4.6.2, would be used. When a Qt GUI Application is created, at least one class is created. According to [21], there is a header file with the same name as the class that contains the class' definition. Also, a UI form is created for that class.

4.3.1 The Main Function

The source code listed below represents a generic main.cpp file for a generic GUI project. Within the project, there exist a class, MainWindow, and a source header and form of MainWindow.

```
#include <QApplication>
#include "mainwindow.h"
int main (int argc, char *argv[])
{QApplication app(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();}
```

The QApplication class is included in Line 1. It should be noted that all Qt classes are prefixed by "Q" [21]. The header for the MainWindow class is included so that definitions in the header could be seen by main.cpp. The argument count (argc) and argument vector (argv) remain the arguments for main() as in C – there is no difference. Line 5 creates an object of QApplication which manages application wide resources [2]. Next, the MainWindow class is instantiated – this is done within main() for any classes within the project. The instantiation of this class is then used to call the show() method which allows the window to be displayed. Similarly, if the window is to be displayed full-screen as in the GUI being developed, the showFullScreen() method is called.

Finally main calls the exec method on a QApplication object which starts the application's event loop. In the event loop, the application waits for user events to which the program responds by executing functions [2, 21].

4.3.2 Widgets

Qt incorporates widgets to enhance GUI functionality. A widget is a visual element in a user interface and is a key aspect in development in this project. Examples of widgets include: pushbuttons, menus, scroll bars, labels, line edits and text edits. Widgets are denoted by a "Q" in front of the name, for example, QLabel, QPushButton, QLineEdit [2].

A widget can be placed onto a form in the forms editor by dragging and dropping it from the widget box. In Qt, the form editor, Qt Designer, is incorporated into the Qt Creator IDE. The properties of these widgets can be set in the Property Editor, while the Signal/Slot editor manages the connections between the objects, making up the working form. The Resource Editor manages resources such as icons and pictures, which are compiled into the executable [19]. The icons and pictures used in the GUI were stored in resource files for each project.

The default form size was set to 240 × 320 px corresponding to the Mini2440. According to [2], widgets can contain other widgets. Most applications use a QMainWindow or QDialog as the application window; however, any widget can be used as a main window [2]. During development of the GUI, QMainWindows and QWidgets are used as main windows and they, in turn, contain other widgets.

4.3.3 Signals and Slots

Perhaps the most important feature which the Qt framework has introduced to C++ and which is vital to the development of the smartphone GUI is that of signals and slots. Signals and slots facilitate the interconnection of objects within a Qt program. According to [19], a signal is a method that is emitted rather than executed when called; therefore, signals are not implemented, but are declared in the class declaration in the *signals* section of the relevant header file.

Reference [19] also states that a slot is a member function which can be invoked as a result of signal emission. The methods treated as slots must be specified by declaring them in the *public slots*, *protected slots* or *private slots* sections of the header files, and further implemented in the source file to perform the desired operation. Any number of signals can be connected to any number of slots [19].

Consider the following code snippet which shows the connection of signals and slots.

```
#include <QApplication>
#include<QPushButton>
int main (int argc, char *argv[])
{QApplication app(argc, argv);
    QPushButton *button = new QPushButton();
    QObject::connect (button, SIGNAL(clicked()),
```

```
&app, SLOT(quit()));
button->show();
return app.exec();}
```

When the *QPushButton* emits the *clicked()* signal when the user clicks the button, the slot function is automatically executed (in this case to quit the application). In the GUI being developed, *QPushButtons* will be used for icons and the keyboard; the standard *clicked()* signal will be used to connect slots which written to either launch another program (in the case of icons) or perform some function on screen. The *clicked()* signal typically represents a mouse click however, on the touchscreen, it is equivalent to a single tap.

While the example shown here is implemented in main.cpp, it should be noted that for this project, the signals of each class were declared in their respective header files and connected in the source files of the classes, thus enhancing interconnectivity. If a signal in one class has to be connected to a slot in another, an instantiation of the first class must be made in the second, and using the object created, the signal and slot can be connected. Note this refers to separate classes within the same project.

4.3.4 Layouts

Another feature of Qt which allowed for the authenticity of the appearance of the GUI was layouts. Layouts are used to organize, manage the geometry of and synchronize widgets; when widgets are placed in a layout, the layout takes responsibility for the widget [2, 19]. In Qt, there are three main layout manager classes: *QHBoxLayout*, which lays out widgets horizontally from left to right, *QVBoxLayout*, which lays out widgets vertically from top to bottom and *QGridLayout*, which lays out widgets in a grid.

Layouts can be specified in code or in the form editor. Since the widgets were placed using the form editor when designing the GUI, it was more convenient to use layouts in the form editor, thus allowing for a visual representation of the final result.

4.3.5 Qt Style Sheets

To customize the appearance of widgets in the GUI, Qt Style Sheets were applied. Qt Style Sheets are textual specifications that can be set on the whole application or on a specific widget [2]. The format is similar to HTML CSS; an example is shown below.

```
QLineEdit {background-color: black;}
QPushButton {color: red;
        border: solid;
        border-width: 1px;
        border-color: white }
```

4.4 Development on Mini2440

For development with the Mini2440, a Linux-based operating system is required. The Linux distribution chosen was Ubuntu 12.10, which reiterates the notion of using open source tools. In order to develop software for the Mini2440, as with any embedded platform, cross-compilation was necessary. Cross-compilation refers to the executable files being generated on one platform (host), in this case the PC, and used in another (destination), in this case, the Mini2440 [26]. Before programming, the cross-compilation environment had to be built. The cross-compiler used was the GNU Compiler Collection (GCC). In particular, the C++ front end of the GCC, G++, was used. The actual cross-compilation toolchain, arm-linux-gcc-4.3.2, was tailored specifically for ARM processors, and was obtained from FriendlyARM website.

When interfacing the Mini2440 with the computer, the components required are a USB cable and an RS-232 DB-9 serial cable. When programming on the host platform in the Ubuntu environment, the two most used tools were Bash and the terminal window. Bash is a shell, or command language interpreter for the GNU operating system whose input is real from the terminal window [16]. The path to the cross-compiler must be specified in the bash file to compile the Qt software libraries used for development as well as the Qt C++ applcations developed.

To enable touchscreen functionality in the Mini2440 while operating outside of the Qtopia environment, the touchscreen library, tslib, had to be compiled and transplanted to the device. Tslib is an abstraction layer for the touchscreen panel events, as well as a filter stack for the manipulation of those events. The tslib repository was cloned from GitHub. In order to develop Qt applications for the Mini2440, a compatible version of the Qt libraries had to be compiled. Here, Qt 4.6.2 was used. One important dfeature of Qt is the qmake tool, which is used to build Qt applications [2]. Since Qt is cross-platform, qmake is configured to cross-compile using G++ for ARM, thus making the application target specific. This is specified in the make specification (mkspec) settings of Qt. One of the specifications is the qmake configuration script [21], which was accessed via the terminal window and the settings edited.

The variables in the configure script include the path to the G++ compiler and specifies the ARM version and architecture. It also includes the path to tslib, to enable touchscreen functionality of applications. Finally, the Qt libraries were compiled

with the desired settings for touchscreen devices, ARM processors and platform libs for various features. During the configuration step, the user is given the option to install the open source version. Compilation takes roughly 1-2 hours to complete.

The tslib and Qt 4.6.2 libraries were then copied to the Mini2440 via USB and using the on-board terminal via serial communication (using minicom). The application environment must be configured so that applications can run outside of Qtopia environment. To do this, the Linux startup script must be edited to halt the launching of Qtopia on startup.

RESULTS

This section outlines the results obtained from the development of the GUI. The images presented illustrate screenshots of the GUI, materializing the blueprints presented in the Design section.

5.1 Home Screen

Figure 10. Home Screen

5.2 Main Menu and Sub-Menus

Figure 11. Main Menu of GUI

Figure 12. Sub-Menus of GUI

ISSN 2278-5612

5.3 GUI Applications

The applications outlined in the Design section of this paper, i.e. Phone, Messages and Browser, were developed and incorporated into the GUI, with the ability to be launched from the home screen/main menu.

5.3.1 Phone

Figure 13. GUI Implementation of the Dial Interface and Calling Interface

5.3.2 Messaging Interface

Figure 14. Messages Application

5.3.3 Browser

Figure 15. Browser Application

DISCUSSION

The GUI was developed successfully in Qt C++. Implementation involved cross-compilation using the relevant cross-compilation toolchain for ARM devices. Applications developed in Qt were successfully transplanted to Mini2440 and thus, implementation of a fully open-source GUI was achieved.

As a result of terms of the open source license of Qt used, the software developed is available as open source software, ensuring usability and flexibility of the interface. Since Qt is a cross-platform GUI application framework, the code can be

cross-compiled and implemented on another target platform, providing that the qmake tool can be compiled for that specific target. This makes the GUI inherently “universal”, which enhances the opneeness of the final released version, since users can modify and redistribute the software.

When designing and programming several aspects of the GUI, this cross-platform property was taken into consideration. Although the software was designed for the Mini2440, which uses stylus input due to the resistive touchscreen, the GUI was designed for use with either styli or fingers. Therefore, if the software is ported to another platform, say one with a capacitive touchscreen, all on-screen controls would be operable using finger input. Recommended icon sizes were adhered to, thus preventing user mishits due to “fat fingers”.

Unit and integration testing were carried out on both the computer (debugging) and on the target platform, the Mini2440. Each aspect of the GUI was developed as a separate Qt application; calls to different “screens” (e.g. launching of Phone from Main Menu) constituted launching one application from the other. As such, unit testing involved testing each application individually for functionality, while integration testing involved testing navigation throughout the GUI (i.e. launching applications for different aspects and returning to previous screen which launched the subsequent application). Testing on the computer however was not deemed efficient as a mouse click will have a higher precision than a finger or stylus on a touchscreen.

On board testing was also necessary to determine whether or not the size of touch targets was adequate, and to ensure that text is easily readable with Mini2440 resolution settings. The importance of on-board testing was justified when, during implementation of the menu system, the icons were too large, leading to a jumbled appearance on the Mini2440, not experienced during debugging.

Implementation on the Mini2440 revealed several faults in the GUI. Given the form factor of the device, small touch targets for the virtual QWERTY keryboard were unavoidable. This led to frequent user mishits when typing (with finger). Also, occultation resulted in button misses. When analyzing the appearance of the GUI on the Mini2440, there was a marked difference in the colour, contrast and brightness, due to the 64K colour, TFT LCD touchscreen.

The most important outcome of testing on the Mini2440 was the determination of performance, since one of the key characteristics of a user friendly GUI is responsiveness. The GUI had a relatively long reaction time to touch events, which can be deemed unacceptable for current smartphone markets. The reason behind this can be explained by reviewing the QProcess class used. The QProcess class is used to start external programs and communicate with them [2]. Therefore, when running an application that has been called/launched by some previous screen, applications are actually being run in parallel with the preceding applications. The Mini2440 hardware specifications must also be considered when determining performance. Resistive touchscreens require firm contact to be made – with users being used to capacitive touchscreens and light taps, this may give the illusion of an unresponsive touch event. Also, the processor speed and memory resources of the Mini2440 pale in comparison to existing smartphones – processor clock frequency of 400MHz, 64MB NAND flash memory, 64MB SDRAM. The upside to this, however, is, given the portability of the software, if the software is ported to a platform with higher performance specs, there may be a significant improvement in speed and responsiveness.

CONCLUSION

The investigation into and application of exeisting GUI design theory and guidelines to the development of an open source GUI resulted in the successful implementation. The terms of the GNU LGPL v2.1, under which the open source version of Qt is licensed, dictates that the software developed will be automatically licensed under the GNU LGPL v2.1, and classified as open source, by default.

Since the GUI was developed and implemented within a hypothetical smartphone environment, an avenue has been paved for the full scale development of a more comprehensive GUI with a larger feature set, which uses the same tools and methods used explained in this paper. Recommendations for further work include using an Embedded Linux platform with GSM/Wi-Fi capabilities and apply the GUI to create a fully functioning smartphone.

REFERENCES

[1] Apple Inc., “iOS Human Interface Guidelines”, http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/IconsImages/IconsImages.html#//apple_ref/doc/uid/TP40006556-CH14-SW1. 2012.

[2] Blanchette, Jasmin and Mark Summerfield, C++ GUI Program-ming with Qt 4, Prentice Hall, 2008.

[3] Clarke, Josh, “Designing for touch”, .net Magazine, available at http://www.netmagazine.com/features/designing-touch. 2013.

[4] Coustan, Dave and Jonathan Strickland, “How Smartphones Work”. HowStuffWorks.com, http://electronics.howstuffworks.com/smartphone.htm. 2001.

[5] Dandekar, Kiran, Balsundar I. Raju, and Mandayam A. Sriniva-san, “3-D Finite-Element Models of Human and Monkey Finger-tips to Investigate the Mechanics of Tactile Sense”, Transactions of the ASME, vol. 125, pp. 682-691, available at http://touchlab.mit.edu/publications/2003_009.pdf. 2003.

[6] Digia 2011. "Qt Style Sheets". Accessed January 26, 2013. http://qt-project.org/doc/qt-4.8/stylesheet.html.

[7] EngineersGarage, “Embedded Linux: Understanding The Em-bedded Linux”, http://www.engineersgarage.com/articles/what-is-embedded-linux. 2012.

[8] Furber, Steve, ARM System-on-Chip Architecture, Addison-Wesley Professional, 2000.

[9] International Telecommunication Union, E.161: “International Operation – Numbering Plan of the International Telephone Service”, ITU-T, 2001.

[10] ISO/IEC, “Information Technology – Screen icons and symbols for personal mobile communication devices”, 2012.

[11] Meredith, Wade, “Best Practices of Touch Screen Interface De-sign”, http://voltagecreative.com/articles/best-practices-of-touch-screen-interface-design/. 2008.

[12] MicroARM Systems Inc., Mini2440 User’s Manual, MicroARM Systems Inc., 2009.

[13] Nakagawa, Takao, and Hideteke Uwano, “Usability Differential in Positions of Software Keyboard on Smartphone”, in The 1st IEEE Global Conference on Consumer Electronics 2012, IEEE.

[14] Nokia Corporation, “Touchscreen Usability”, http://www.developer.nokia.com/Community/Wiki/TouchScreen_Usability. 2012.

[15] O2, “Making calls has become fifth most frequent use for a Smartphone for newly-networked generation of users”, http://news.o2.co.uk/?press-release=Making-calls-has-become-fifth-most-frequent-use-for-a-Smartphone-for-newly-networked-generation-of-users. 2012.

[16] Ramey, Chet and Brian Fox, The GNU Bash Reference Manual, Free Software Foundation, 2010.

[17] Ryzhyk, Leonid, “The ARM Architecture”, University of New South Wales, 2006.

[18] Samsung Electronics Co. Ltd., Installation Manual for S3C2440, Samsung Electronics Co. Ltd., Yonging-City, Gyeonggi-Do, Korea, 2004.

[19] Sooknanan, Daniel, and Ajay Joshi, “Investigation, Formulation and Development of an Open GUI for the Touchscreen Smartphone”, International Journal of Computers & Technology, vol. 10, No. 8, pp. 1892-1904, available at http://ijctonline.com/ojs/index.php/ijct/article/view/2199. 2013.

[20] Technopedia, “Graphical User Interface (GUI)”, http://www.techopedia.com/definition/5435/graphical-user-interface-gui. 2013.

[21] Thelin, Johan, Foundations of Qt Development, Apress, New York, 2007.

[22] Todish, Tim R., “Not Your Parent’s Mobile Phone: UX Design Guidelines For Smartphones”, http://uxdesign.smashingmagazine.com/2011/10/06/not-your-parents-mobile-phone-ux-design-guidelines-smartphones/. 2012.

[23] Ubuntu, “UMEGuide/Designing For Finger UIs”, https://help.ubuntu.com/community/UMEGuide/DesigningForFingerUIs. 2008.

[24] Villamour, Craig, Dan Willis and Luke Wroblewski, “Touch Gesture Refrence Guide”, [Accessed Jan. 13, 2013] Available: http://static.lukew.com/TouchGestureGuide.pdf.

[25] Waloszek, Gerd, Interactive Design Guide for Touchscreen Applica-tions, SAP Design Guild, 2000.

[26] Wangfengzhu, Miaoliang, Zhanglei Zhangming, and Su-ichunrong, “Research and Design of Smartphone System based on ARM9 and Embedded Linux Operating System”, in 2011 International Conference on Computational and Information Sciences, IEEE Computer Society. 2011.

[27] Wroblewski, Luke, “Touch Target Sizes”, http://www.lukew.com/ff/entry.asp?1085. 2010.