Computer and Information Science; Vol. 8, No. 2; 2015
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Using GUI Design Theory to Develop an Open Source Touchscreen Smartphone GUI

Daniel Jovan Sooknanan[1] & Ajay Joshi[1]

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

Correspondence: Daniel Sooknanan, Department of Electrical and Computer Engineering, University of the West Indies, St. Augustine Campus, Trinidad and Tobago. E-mail: daniel.sooknanan@my.uwi.edu

Received: February 10, 2015 Accepted: March 13, 2015 Online Published: April 26, 2015
doi:10.5539/cis.v8n2p43 URL: http://dx.doi.org/10.5539/cis.v8n2p43

Abstract

Of late, there has been a rise in the use of touchscreens in many handheld devices, one in particular being the smartphone. Accompanying this is a myriad of smartphone platforms, each of which boasts a user-friendly GUI in its own accord. One of the major aspects of smartphones is the GUI, since this is the aspect with which users interact. This paper is centered on GUI design theory and explores the facets which make a GUI usable and flexible; further, this paper goes on to show the application of the GUI design guidelines and mobile usability studies to the design and development of an open source touchscreen smartphone GUI, which, in turn, will be implemented on an Embedded Linux development board which will serve as a hypothetical smartphone. 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. With the adoption of an open source philosophy, it has also been shown that the use of open source development tools can lead to a decreased cost of production.

Keywords: graphical user interface, human-computer interaction, mobile computing, mobile environments, portable devices, embedded systems, microcomputer applications

1. Introduction

The touchscreen smartphone is undoubtedly one of the most ubiquitous devices in the digital age of today. A focal point of these devices is the graphical user interface (GUI), which acts as the primary means of human-machine interaction. The GUI, being interacted with via the touchscreen, should be usable, flexible and unique. A certain degree of visual appeal is also expected and can inevitably influence a user's choice of smartphone OS.

A graphical user interface is essentially an interface through which users interact with electronic devices, including computers and handheld devices (Technopedia, 2013). 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 covers an investigation into existing mobile GUI design theory/guidelines and mobile usability studies, and their application to the formulation and design of a smartphone GUI for a typical touchscreen smartphone. It explores the effect of layout on usability and the techniques used in mainstream industry to enhance customer experience. For the purpose of this work, the 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.

When compared to existing high-end smartphones, the embedded system which functions as the hypothetical smartphone in this project has several constraints which limit design and development. These include display resolution, the resistive TFT LCD touchscreen, processor/memory resources and the lack of an accelerometer. The scope of this work will therefore exclude such features as multi-touch capabilities, haptic (tactile) feedback and automatic screen reorientation.

Additionally, the GUI implemented will not include third party apps which may require significant processor resources. Instead, only basic apps of a typical mobile phone (such as Phone, Messages, Browser), chosen as an outcome of mobile usability studies, will be developed for the sake of meeting the minimum expectations of smartphone users. Third-party applications will also not be supported in this system for litigious reasons. However, due to the nature of Embedded Linux, there is the potential for third party applications to be incorporated.

2. Overview of GUI Design Theory

Touchscreen GUIs for smartphones are dictated by the device's operating system. User friendliness and ease of use have therefore become crucial in any GUI, which by extension lends itself to the popularity of the associated smartphone OS. Before delving any further into the development process, an overview of mobile GUI design theory and guidelines is given.

2.1 GUI Design and Layout

The mobile realm has several constraints which UI designers must consider when designing and developing GUIs. The most obvious property to consider is the form factor of these devices, i.e. the screen size. Mobile screens are relatively small and with the vast feature sets with which smartphones are now equipped, an adequate display area is required. As such, only critical functions and content should be included, and these should be laid out strategically in the available screen space (Todish, 2011).

When designing a touchscreen GUI, 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, as illustrated in Figure 1. The areas on the screen where the thumbs come to rest are known as easy reach areas, and can be considered as regions of high activity. It should be noted that roughly one-third of the screen is considered an easy-reach zone, since the thumb falls naturally in an arc at the bottom left/right corners of the screen. This is an important factor when considering button 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 (Clarke, 2013).

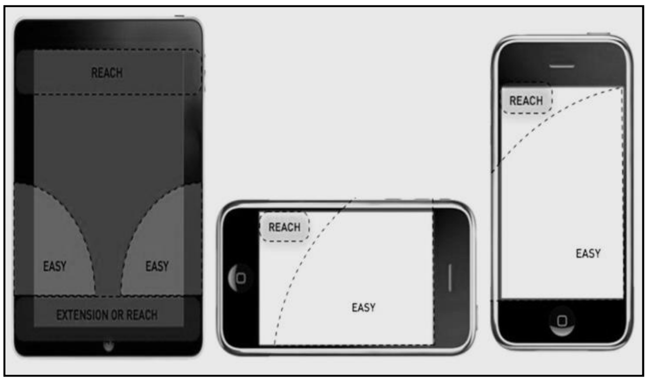


Figure 1. Diagrams showing the Activity Zones of touchscreen smartphones

For the sake of flexibility, an ambidextrous design must be considered. In delivering the same experience to all users, designers should adopt vertically symmetrical layout, which can, in turn, further simplify the interface (Meredith, 2008).

2.2 Touchscreen Target Sizes

Users of touchscreen mobile devices typically interact with the GUIs via fingers or styli. As such, it is recommended practice to design an interface for use with both fingers and styli – this increases overall usability and flexibility of the GUI (Meredith, 2008). User interface controls should be of adequate size to capture fingertip/stylus actions, without preventing user mishits due to "fat fingers" (Wroblewski, 2010). Typical user interface controls include icons, push buttons, check boxes and lists, among others. For icons used to launch applications from a smartphone menu, the ISO/IEC (2012) states that for touchscreen devices accessible by stylus pen or finger, all icon graphics should be displayed with a resolution of 32×32 pixels or higher. Similarly, 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 (Apple Inc., 2012).

Microsoft Inc.'s Windows Phone UI Design and Interaction Guide suggests a typical touch target size of 9mm/34px, with a minimum size of 7mm/26px and spacing between elements of at least 2mm/8px. Further, this 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, as shown in Figure 2 (Wroblewski, 2010).

According to Nokia's developer resources, touchable interface elements should not be smaller than the average finger pad, i.e. no less than 1cm in diameter. Nokia's documentation also states that the minimum target sizes for finger usable UI elements are: 7×7 mm with 1mm gaps for index finger usage and 8×8 mm with 2mm gaps for thumb usage (Wroblewski, 2010).

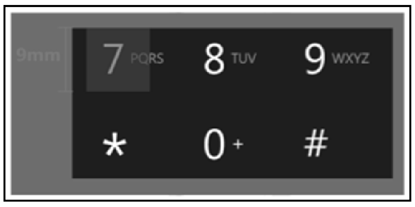


Figure 2. Sequential UI Element (dial pad) size specification

Similarly, Ubuntu's documentation for Designing for Finger UIs states that 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 (Dandekar, Raju & Srinivasan, 2003; Ubuntu, 2008).

2.3 Input Methods – Human Machine Interaction and the GUI

The touchscreen is a direct input device – one in which input and display coincide on the same space. With the absence of a physical keypad, human-device interface problems can arise, as there must be some means of providing input and navigating throughout the GUI (Meredith, 2008). This introduces the concept of various gestures to enhance interactivity and usability (Todish, 2011).

Table 1 summarizes the core touch gestures as outlined by Villamour, Willis and Wroblewski (2013), which can be implemented on a single-touch surface (such as the resistive touchscreen), as a means of HMI, and their actions/outcomes. The gestures described here are a minimal subset of numerous possible gestures. However, since the device used for implementation imposes the limitation of single touch, multi-touch gestures will be excluded from this discussion.

Table 1. Core Touch Gestures, their Description and the Associated Actions/Outcomes

Core Gesture	Description	Action/Outcome
Tap	Briefly touch surface with fingertip	Select/Open
Double Tap	Rapidly touch surface twice with fingertip	Open
Drag	Move fingertip over surface without losing contact	Scroll, Move, Delete
Flick/Swipe	Quickly brush surface with fingertip	Scroll, Next page
Press	Touch surface for extended period of time	Change mode, Move item

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.

2.3.1 Keyboard/Keypad

Since there is no keypad on the touchscreen smartphone, a virtual keyboard must be included for text input. Typically, touchscreens are not well suited for data entry (Nokia Corporation, 2012). On the single-touch touchscreen, data input is sequential – one finger or stylus is used for clicking one key at a time, which slows down the typing process (Waloszek, 2000).

For textual input, providing a QWERTY keyboard is advised (Nokia Corporation, 2012). 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 (Nakagawa & Uwano, 2012). 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 (Nokia Corporation, 2012).

3. Design

This section of the paper seeks to delineate 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 and outlines the blueprints of the GUI before development and programming. The guidelines mentioned previously will be applied in the planning and design stage.

3.1 GUI Layout

When designing the GUI and corresponding applications, the first consideration should be the form factor of the device. 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 display constraints of this device were taken into account during design – a display resolution of 240×320 px. Additionally, due to the presence of a resistive touchscreen, the suitability to stylus input was considered.

With the absence of an accelerometer, the GUI will be developed solely in the portrait orientation, since automatic screen reorientation cannot occur. The following subsections explain various facets of GUI layout applied during design.

3.1.1 Indicators

Status indicators in any mobile phone, smartphone or otherwise, provide the user with the current status of the phone in terms of battery life, time/date, signal level (antenna, Wi-Fi), among others. Indicators populate the top of the screen, as is the standard for most mobile devices. Indicators were included for the sake of authenticity.

The indicators implemented in the design include: battery indicator, antenna level indicator, Wi-Fi antenna indicator and a digital clock. Note, actual on-board statistics will not be reflected, in terms of battery life and antenna indicators, since the device is powered by 5V DC supply and there are no telecommunication aspects present. Given the resolution and form factor of the hypothetical smartphone, an area of 240×20 px will be dedicated to status indicators, as illustrated in Figure 3.

3.1.2 Navigation

Since no buttons were used in this design, navigation must be facilitated within the touchscreen GUI. The buttons on the Mini2440 were not used due to their position on the board, which makes them virtually inaccessible. As such, it was decided to use soft buttons to navigate between screens in the GUI.

3.1.3 Home Screen

The home screen of a smartphone is considered to be the first screen of the GUI with which a user is greeted when the GUI is engaged. It is comprised of a wallpaper, indicators and a digital clock and contains four docked application icons at the bottom of the screen, which can be used to launch the corresponding applications.

The primary controls were placed at the bottom of the screen for easy reach. The vertically symmetrical layout of the icons reflects an ambidextrous design. The docked applications on the home screen will consist of the traditional features of a mobile phone: Phone, Messages and Browser. The fourth icon will be used to launch the main menu of the GUI (discussed further). The home screen blueprint is also shown in Figure 3.

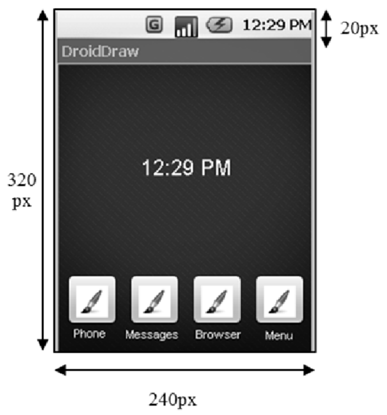


Figure 3. Sketch of the Home Screen Layout

3.1.4 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 by O2 (2012) are shown in Table 2.

Table 2. 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
Text 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. An icon is a picture or symbol representation of a program/application. For the menu system implemented, each application will be represented by an icon 44×44 px in size, as per the guidelines stipulated by Apple Inc. (2012). Each icon was selected based on the relevance to the application it represented. The proposed menu interface is outlined in Figure 4.

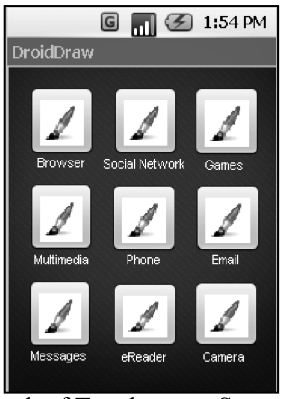


Figure 4. Sketch of Touchscreen Smartphone Menu

Several other native menu items not stated in Table 2 were included during implementation so that the menu would further resemble that of a smartphone. These include: Contacts, Calendar, Notes, Clock, Settings and Maps.

3.2 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 design process for each of these applications is outlined in this section.

3.2.1 Phone – Dialling Interface/Calling Interface

This 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 (2001). 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. Sketches of the dialing interface and calling interface are shown below in Figure 5.

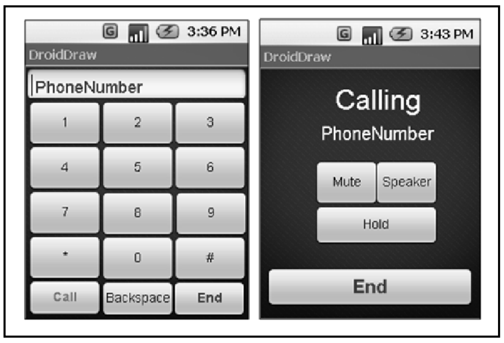


Figure 5. Sketch of Dialing Interface (Left) and Calling Interface (Right)

3.2.2 Messages – SMS Interface

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. The sketch of the initial design for this application interface is shown in Figure 6.

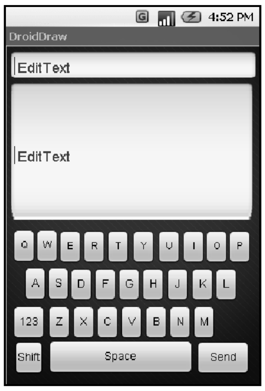


Figure 6. Sketch of Messaging Interface with keyboard

3.2.3 Internet Browser

Since the Internet browser is the most used feature of smartphones (see Table 2), 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. The sketch of the Internet browser blueprint is shown in Figure 7.

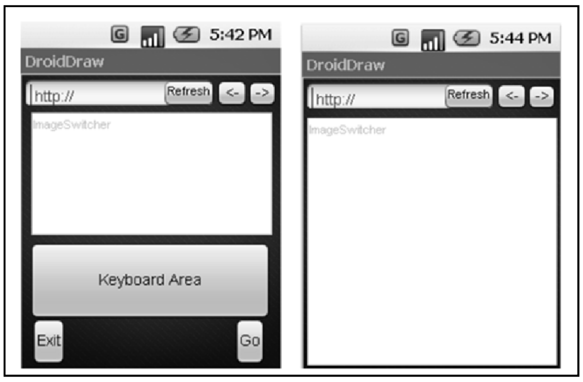


Figure 7. Sketch of the Browser Interface

4. Programming and Development

The blueprints and sketches for the designed GUI were materialized by developing software for the GUI and further implemented on an Embedded Linux development board. This section shows the application of the design theory to the formulation of the GUI for the hypothetical smartphone.

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. This section assesses the feasibility of adopting an open source philosophy during development. As such, the platforms used will be introduced and justified accordingly, and the development process will be explored in some detail.

4.1 Justification of the Hardware Platform

Smartphones are now capable of handling an abundance of features, and as such, an appropriate hardware platform had to be chosen to live up to these requirements. Since the majority of smartphones today are equipped with ARM cores, 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 (Furber, 2000). Another major advantage of utilizing an ARM-based platform is the increased level of integration in the mass market.

An Embedded Linux target was preferred due to the reduced developmental costs given its free license, and the flexibility in development. The use of Embedded Linux purports the concept of the open source philosophy.

Given the aforementioned requirements, it was decided that the FriendlyARM Mini2440 development board, shown in Figure 8, would be used 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) (Ryzhyk, 2006; Samsung Electronics Co. Ltd., 2004). 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 (MicroARM Systems Inc., 2009; Wangfenzhu, Zhangming and Su-ichunrong, 2011).

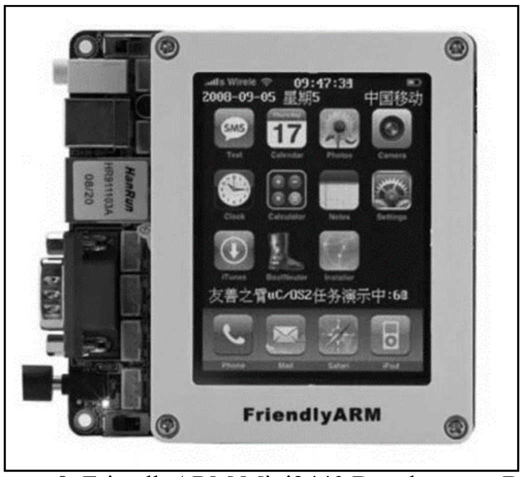
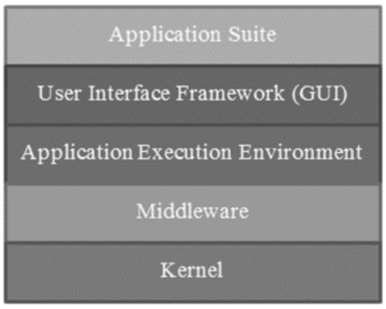


Figure 8. FriendlyARM Mini2440 Development Board

4.2 Justification of Software Platform

4.2.1 Software Architecture of a Smartphone

To understand the scope of development in this project, consider the software architecture of the smartphone. According to Coustan and Strickland (2001), the software architecture in smartphones can be considered as a software stack, which consists of five layers, as shown in Figure 9.


Application Suite
User Interface Framework (GUI)
Application Execution Environment
Middleware
Kernel
Figure 9. Software Architecture of Smartphone

The kernel layer consists of process management systems and hardware drivers. Middleware contains software libraries which enable the use of application, while the Application Execution Environment (AEE) enables the creation of applications. The User Interface Framework is responsible for graphics and layout, while the Application Suite layer includes all applications with which users interact (Coustan & Strickland, 2001).

Development will occur in the upper two layers, with the primary focus being on the User Interface Framework. In order to give the GUI a sense of functionality and completeness, some design and development was done in the Application Suite layer also.

4.2.2 Justification of the Application Framework

When determining a suitable software platform that could be used to develop a rich, open GUI, the constraints set by the development board were considered, i.e. which platforms are compatible with the Mini2440. 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 operating system, as a whole. Embedded Linux implies the adoption of the Linux distribution in an embedded device; the Linux kernel can be built on various platforms (EngineersGarage, 2012). The GUI will be programmed on top of this kernel, and consequently, the kernel, middleware and AEE preinstalled on the Mini2440 will be used, in tandem with the smartphone software architecture, as outlined previously.

The Mini2440 is also pre-installed with a user interface framework and application suite: Qtopia (MicroARM Systems Inc., 2009). Qtopia is an embedded application platform/framework designed specifically for Embedded Linux (Blanchette & Summerfield, 2008). Although a suitable candidate for development, its use constituted writing applications for an already existing GUI, which detracted from designing and implementing a GUI. Therefore, in selecting an application framework that is compatible with Embedded Linux and which is in itself open and will give rise to open software, 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 (Blanchette & Summerfield, 2008). Qt uses C++ language constructs and syntax with added macros for rich GUI functionality. The open source Qtopia environment of the Mini2440 was actually developed using Qt. 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 (Blanchette & Summerfield, 2008).

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

4.3 Overview of Programming with Qt

Qt takes ordinary C++ classes and integrates them with Qt classes, objects and macros to make code easier to use (Thelin, 2007). The IDE used for development of Qt applications is Qt Creator. In order to develop Qt applications for the Mini2440, 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.

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.

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 (Blanchette & Summerfield, 2008; Thelin, 2007). 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.

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 (*Qt Style Sheets*, 2011). The format is similar to HTML CSS.

4.4 Developing for the Mini2440

For development with the Mini2440, a Linux-based Operating System is required. The Linux distribution chosen was Canonical's Ubuntu which reiterates the notion of using open source tools. In the Ubuntu environment, the fundamental aspect to working with the Mini2440 was Bash and the terminal window. Bash is a shell, r command language interpreter for the GNU operating system whose input is read from the user's terminal window, a command-line interface (Ramey & Fox, 2010).

In order to use certain tools/libraries, the source code was obtained as Git repositories which then had to be compiled in the terminal and installed. When interfacing the Mini2440 with the computer for programming, the components required are: a USB cable and an RS-232 DB-9 Serial Cable.

Before programming, the cross-compilation environment must be built. Cross-compilation refers to executable files being generated on one platform, in this case the PC, and used in another, in this case, the Mini2440 (Wangfengzhu, et. al., 2011). The cross-compiler used in this project is 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 the FriendlyARM website.

After installation of the cross compiler, its location must be specified in the bash file, ".bashrc". The ".bashrc" bash file contains commands which Bash reads and executes whenever the shell is invoked (Ramey & Fox, 2010). 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++ applications 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. As mentioned before, Qt 4.6.2, which is compatible with the Mini2440 and stable was used as the Qt libraries needed for development.

One important feature of Qt used during development was the qmake tool, which is used to build Qt applications (Blanchette & Summerfield, 2008). Since Qt is cross platform, qmake is configured to cross compile using G++

for ARM, thus making the application target specific. The process of compilation and installation of the Qt and Tslib libraries will not be explained in detail in this paper however.

Finally, the application environment was configured so that applications can run outside of the Qtopia environment on-board. Again, the detailed procedure for configuring the application environment on the development board is out of the scope of this discussion. After all the necessary configurations, applications can now be transplanted to and launched successfully on the Mini2440.

5. Results

This section outlines the results obtained from the development of the GUI, giving details of implementation in the programming environment during debugging. The images presented illustrate screenshots of the GUI, materializing the blueprints presented in the Design section.

5.1 Home Screen

When the GUI is launched, the Home Screen is the first screen the user is greeted with. This aspect of the interface is shown in Figure 10. A fully functional digital clock displaying hours and minutes with a blinking colon was implemented as a custom widget – the code for this widget was obtained from the Nokia Corporation's Qt Toolkit under the BSD license, where redistribution and use was permitted.

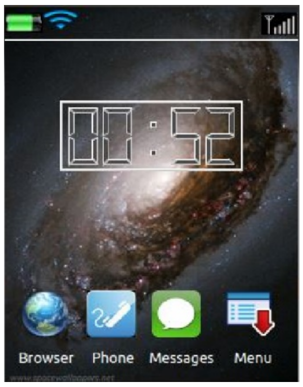


Figure 10. Home Screen

The docked application icons at the bottom of the screen were implemented using *QPushButtons*. To give the icons an authentic appearance, in the Property Editor, the *QPushButtons* were made "flat", so that when an image is superimposed, the visual area fully represented the activation area. Icons were incorporated using resource files as mentioned previously. A vertical layout was used for each icon to align it with its corresponding text label, and a horizontal layout was used to align the four icons. The indicators at the top of the screen were implemented using *QLabels*. A *QLabel* can display an image using a pixmap. The pixmap of each label was set via image resources in the resource file (similar to icons).

5.2 Main Menu and Sub-Menus

The menu system application consisted mainly of a grid layout of QPushButton icons, implemented in the same manner as in the Home Screen. The indicators at the top of the screen were also implemented as before, however, in this case, an instance of the digital clock was included at the top, with the size and location set to scale to fit the desired on-screen region.

Due to the display resolution of the Mini2440, for all icons to be visible/accessible, a scrolling mechanism was needed to display all proposed menu items. A QScrollArea widget was included, and was bound to the icons using a layout; the scroll bar automatically appears so as to correspond to the scroll area and the amount of widgets contained in it.

In order to provide navigation to return to the home screen, a "Home" button was included, just below the indicators. The clicked() signal of the Home button was connected to the close() method in the slot, thus terminating the application and returning to the home screen. The same wallpaper in the home screen was used here, and throughout the GUI for the sake of uniformity. For the wallpaper to be visible behind the icons, the scroll area was made transparent using Style Sheets.

As mentioned previously, all multimedia functions were grouped together. To facilitate this, a Multimedia sub-menu was included, which contained all multimedia related applications, with the ability to be launched from the Main Menu. Also, recall from the usability study that playing games was one of the more popular reasons for which smartphones are used. Therefore, a Games sub-menu was also included in the Main Menu. Note that within the sub-menus, navigation is also provided with the inclusion of the "Back" soft-buttons at the bottom left hand corner of the screen. Figures 11 and 12 illustrate the Main Menu and Sub-Menus of the GUI respectively.

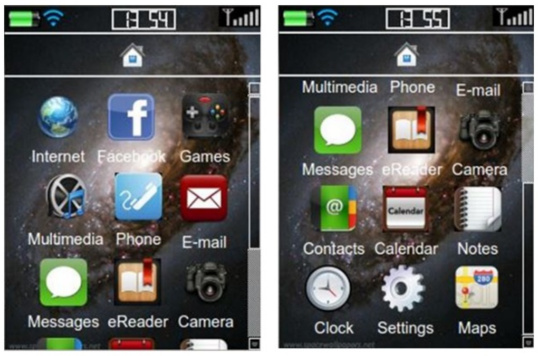


Figure 11. Main Menu of GUI

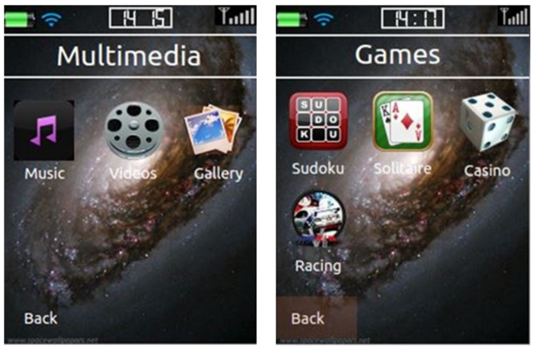


Figure 12. Sub-Menus of GUI

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

The dial pad and associated controls were laid out as specified previously. In order to customize the appearance of the buttons, style sheets were used to add radial colour gradients to the digits, and to provide the typical colours to the call and end buttons. An icon was used to represent the backspace button. In keeping with Microsoft Inc.'s guidelines on designing dial pads, the dial pad buttons were set to 80×50px, which also capitalizes on screen space.

For the action of dialing, the clicked() signal of each digit push button was connected to a slot which outputs the corresponding number in the QLineEdit using the insert() function. For the backspace button, the clicked() signal was linked to a slot which deletes text in the line edit via the backspace() function.

After dialing the number and pressing the Call button, the call screen is raised. This screen is actually the main window of another class, CallingInterface. In order to make this screen visible, the clicked() signal of the call button is connected to an instantiation of CallingInterface. Figure 13 illustrates the implementation of the Phone application, including the dialing interface and the calling interface.

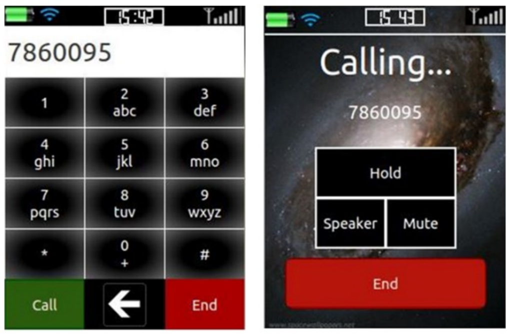


Figure 13. GUI Implementation of the Dial Interface and Calling Interface

5.3.2 Messaging Interface

In the implementation of the SMS messaging interface, the focus was on developing a virtual keyboard. In the keyboard, each key was implemented as a QPushButton, where the clicked() signal of each key was connected to individual slots which output text onto the QTextEdit widget using the insert() function. Unlike the previous design, this keyboard included a period and a comma, thus preventing the need to switch to the number-symbol input mode for basic punctuation. This effectively increased input speed.

The number-symbol button is a checkable button which allowed the same keys to input symbols and numbers, based on whether or not it was checked, by connecting the push buttons' clicked() signal to different slots. Similarly, the Shift button was checkable to enable the input of upper case letters. The shift, backspace and send buttons were implemented as icons. Figure 14 shows the results of development of the SMS interface.

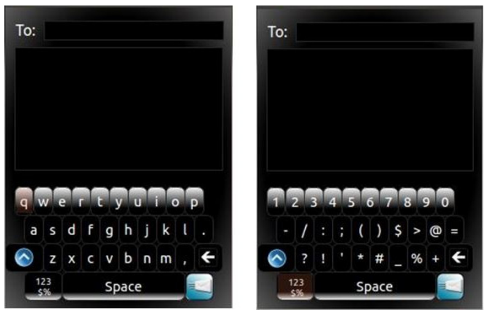


Figure 14. Messages Application

5.3.3 Browser

The browser was not entirely developed according to the specification in the Design section, since it did not incorporate the virtual keyboard, and by extension, lacked the ability to enter web addresses. During implementation however, the typical layout of a mobile browser was developed, which included browser buttons implemented as QPushButton icons, and a tab window system (using QTabWidget). Due to the lack of internet connectivity on the Mini2440, the browser is not functional, however, the visual aspects of it were developed. To provide navigation to the previous screen, an Exit button was provided at the bottom left hand corner. This is illustrated in Figure 15.

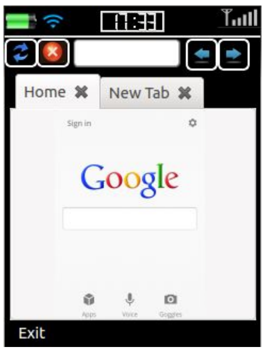


Figure 15. Browser Application

5.4 Implementation of the GUI on the Mini2440

After full development and cross compilation of the GUI, the required executable files were transplanted to the Mini2440, to perform on board testing. This was necessary since different platforms execute applications differently, which may lead to unexpected results from those obtained within the IDE during debugging. Furthermore, there were certain aspects of the GUI which were native to the Mini2440. It is therefore important to note that several anomalies arose as a result of this testing.

Firstly, there was a noted difference in the appearance of the GUI, in terms of brightness, contrast and colour. While the layout of controls on-screen remained fairly constant, in comparison to the debugging results, the text in the labels and on the push buttons appeared significantly smaller than expected. As such, the GUI was re-programmed, and the size of text increased.

Applications (Home Screen, Main Menu and the GUI apps) were unit tested to gauge their performance. If any changes were necessary, the programs were modified accordingly before releasing the final version. Integration testing was then conducted to test the interactivity of applications with one another. Testing was imperative to determine the performance of the GUI on the device, in terms of speed and responsiveness, since these are factors which lend to the usability of a mobile touchscreen GUI.

When the GUI is launched, it is important to note that it is relatively unresponsive. In terms of touch events, when a push button is touched, the time taken for the corresponding reaction to occur is notable. Secondly, within the main menu, the reaction time to scrolling is relatively slow. Also of importance is the time taken to launch an application either from the home screen or main menu. From the home screen, it takes roughly 6-7s to launch the Phone application and approximately 28s to launch the main menu. Conversely, when closing an application the GUI is responsive and the previous screen is almost instantaneously shown.

6. Discussion

The GUI was designed and implemented successfully. The software was developed using open source tools, resulting in open source software and was tested successfully on the Mini2440, ensuring usability and flexibility of the interface.

In analyzing the application of mobile GUI design theory to this paper, consider the factor of touch target sizes. The GUI was designed for use with either styli or fingers, adding to its flexibility. Therefore, if the software is ported to another platform, say one with a capacitive touchscreen, these aspects would still be operable using finger input. Recommended icon sizes were adhered to, thus preventing user mishits due to “fat fingers”.

On the other hand, in the Messages application, due to the size of the keys in the virtual keyboard, button mishits are likely. This applies only to cases of finger input, since the stylus offers higher input accuracy. This application therefore poses usability issues and limits flexibility. The same can be said for the Browser application since the controls may be deemed too small for ideal finger pad touching.

Another aspect of usability which can be explored is that of gestures used in the GUI. Of the numerous gestures used in typical smartphones, since the resistive touchscreen inhibited the use of multi-touch gestures, a minimal subset of gestures was considered. In particular, the single tap and swipe were used. Consequently, it can be deduced that the single tap is the most important gesture used in touch screen GUIs.

In terms of the soft buttons used for navigation, although the need for physical buttons was overcome, due to the form factor of the Mini2440, these soft buttons occupied useful screen space and effectively crowded the GUI. The use of soft buttons for navigation during design is therefore discouraged when form factor is an issue.

Another aspect of the software which makes it flexible is its ability to be cross-platform. Depending on the cross-compilation settings, the source code can be compiled for any platform which supports Qt. This alludes to a sense of universality of the GUI, which enhances the openness of the final released version, since users can modify and redistribute software.

Performance on the GUI, as stated, was not up to par. This unresponsiveness has adverse effects on usability, especially when compared to mainstream smartphones on the market. The reason behind applications being unresponsive when launching subsequent applications can be explained by reviewing the QProcess class used. According to the Qt Reference Documentation, the QProcess class is used to start external programs and communicate with them. Thus, when opening an application from, say, the home screen, applications are actually being run in parallel with that which opened it.

In analyzing the response time to launching an application, it was inferred that the lag time between a touch event and a corresponding reaction was directly proportional to the size of the application. Thus, for an application which contains more controls and data I/O, it is expected that the response time would be longer. Conversely, when an application is closed, it returns to the previous interface immediately since processor resources are freed up, giving the appearance of an instantaneous response.

7. Conclusion

After investigations were carried out into the design of a mobile GUI, the implementation was successful, adequately accomplishing the design outlined. The use of an Embedded Linux platform and an ARM core facilitated the development of a low cost, stable, dependable system; also, open source software development tools aided in cost savings. It should also be noted that the terms of the GNU LGPL v2.1, under which the open source version of Qt is licensed, dictate that the software developed will be automatically licensed under the GNU LGPL v2.1, and automatically classified as open source.

Due to the specifications of the target platform, and the programming approach taken for launching subsequent applications, the performance and usability of the GUI were compromised. This may detract from user experience and, by extension, popularity and innovation. The Mini2440's resistive touchscreen hindered design, in terms of multi-touch capabilities, thus crippling the profoundness of the GUI. Today, multi-touch gestures are widely used and offer an important avenue to human-machine interaction. The lack of kinetic features, in actions such as scrolling and swiping also detracted from the human-machine interactivity of the final product.

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.

References

Apple Inc. (2012). IOS Human Interface Guidelines. Retrieved from http://developer.apple.com/library/ios/#documentation/UserExperi-ence/Conceptual/MobileHIG/IconsImages/IconsImages. html#//apple_ref/doc/uid/TP40006556-CH14-SW1

Blanchette, J., & Mark, S. (2006). *C++ GUI Programming with Qt 4*: Prentice Hall PTR.

Clarke, J. (2013). Designing for touch, net Magazine. Retrieved from http://www.netmagazine.com/features/designing-touch

Coustan, D., & Jonathan S. (2001). How Smartphones Work. HowStuffWorks.com. Retrieved from http://electronics.howstuffworks.com/smartphone.htm

Dandekar, K., Balsundar, I. R., & Mandayam, A. S. S. (2003). 3-D Finite-Element Models of Human and Monkey Fingertips to Investigate the Mechanics of Tactile Sense". *Transactions of the ASME, 125*, 682-691. http://dx.doi.org/10.1115/1.1613673

Daniel, S., & Ajay, J. I. (2013). Formulation and Development of an Open GUI for the Touchscreen Smartphone, *International Journal of Computer and Technology. ISSN 22773061, 10*(8), 1892-1904

Digia. (2011). Qt Style Sheets. Retrieved January 26, 2013 from http://qt-project.org/doc/qt-4.8/stylesheet.html

EngineersGarage, Embedded Linux: Understanding The Embedded Linux. Retrieved from http://www.engineersgarage.com/articles/what-is-embedded-linux

Furber, S. (2000). ARM System-on-Chip Architecture, Addison-Wesley Professional. http://dx.doi.org/10.1109/MNET.2000.885658

International Telecommunication Union (2001). E.161: International Operation - Numbering Plan of the International Telephone Service. In *Use of digits and letters on telephone sets*: ITU-T. ISO/IEC, Information Technology – Screen icons and symbols for personal mobile communication devices, 2012.

Meredith, W. (2008). Best Practices of Touch Screen Interface Design. Retrieved from http://voltagecreative.com/articles/best-practices-of-touch-screen-interface-design/

MicroARM Systems Inc. (2009). Mini2440 User's Manual, MicroARM Systems Inc.

Nakagawa, T., & Uwano, H. (2012). Usability differential in positions of software keyboard on smartphone. Paper read at Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on, 2-5 Oct. 2012. http://dx.doi.org/10.1109/GCCE.2012.6379610

Nokia Corporation. (2013). *Touchscreen Usability*. Retrieved February 17, 2013, from http://www.developer.nokia.com/Community/Wiki/TouchScreen_Usability

O_2 (2012). Making calls has become fifth most frequent use for a Smartphone for newly-networked generation of users. Retrieved from http://news.o2.co.uk/?press-release=Making-calls-has-become-fifth-most-frequent-use-for-a-Smartphone-for-newly-networked-generation-of-users

Ramey, C., & Brian, F. (2010). The GNU Bash Reference Manual, Free Software Foundation.

Ryzhyk, L. (2006). The ARM Architecture. Chicago University, Illinois, EUA.

Samsung Electronics Co. Ltd. (2004). Installation Manual for S3C2440, Samsung Electronics Co. Ltd., Yonging-City, Gyeonggi-Do, Korea.

Technopedia (2013). Graphical User Interface (GUI). Retrieved from http://www.techopedia.com/definition/5435/graphical-user-interface-gui

Thelin, J. (2007). Foundations of Qt Development, Apress, New York. http://dx.doi.org/10.1007/978-1-4302-0251-6

Todish, T. R. (2012). Not Your Parent's Mobile Phone: UX Design Guidelines for Smartphones. Retrieved from http://uxdesign.smashingmagazine.com/2011/10/06/not-your-parents-mobile-phone-ux-design-guidelines-smartphones/

Ubuntu (2008). UMEGuide/Designing For Finger Uis. Retrieved from https://help.ubuntu.com/community/UMEGuide/DesigningForFingerUIs

Villamour, C., Dan, W., & Luke, W. (2003). Touch Gesture Reference Guide. Retrieved January 13, 2013, from http://static.lukew.com/TouchGestureGuide.pdf

Waloszek, G. (2000). Interactive Design Guide for Touchscreen Applications, SAP Design Guild. Retrieved from http://www.sapdesignguild.org/goodies/TSDesignGL/TSDesignGL.pdf

Wang, F. Zh., Liang, M., Ming, Zh., Lei, Zh., & Sui, Ch. R. (2011). Research and Design of Smartphone System Based on ARM9 and Embedded Linux Operating System. Paper read at Computational and Information Sciences (ICCIS), 2011 International Conference on, 21-23 Oct. http://dx.doi.org/10.1109/ICCIS.2011.210

Wroblewski, L. (2010). Touch Target Sizes. Retrieved from http://www.lukew.com/ff/entry.asp?1085