Michael Kinuthia

Local Connectivity in Qt Markup Language

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Information Technology

Thesis

29 January 2013

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Michael Kinuthia Local Connectivity in Qt Markup Language 32 pages + 4 appendices 29 January 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Dr Mahbub Rahman

The objective of this thesis was to study Qt/QML, NFC and Bluetooth technology, with the aim of investigating how data is transferred from low level NFC to QML application level. The knowledge gained was to be used to develop a QML-based application to demonstrate the local connectivity of Bluetooth and NFC. The target device was the Symbian phone for which Nokia 603 was selected in this project for demonstration purposes.

The project was carried out in two parts first, the theoretical part and the practical part. The theoretical part entailed studying of Qt platform, QML language, NFC and Bluetooth technologies. Qt and QML were used as the development environment and development language respectively. NFC was used to trigger the local connectivity and Bluetooth aided in the transfer of user input from one phone to the other and vice versa. The basic idea, which was the practical part, was demonstrated by developing a sample game that could be played by two players, data was transferred over Bluetooth and the connection was initiated by NFC. In the game data transfer was initiated by NFC that started the Bluetooth pairing and connection. The Bluetooth connection was then used to pass the data from one phone to another phone.

The result of this thesis showed a successful local connection of NFC and Bluetooth using QML technology. This clarified Qt, QML, NFC and Bluetooth technologies in the different aspects and the development of a QML mobile application. The result showed that even a developer with the basic programming skills can easily study, develop and deploy applications in QML. It also shows that different technologies can be combined to come up with one working application. The game which was developed in this project gave a clear indication that more intuitive applications and games could be developed using QML with NFC and Bluetooth technologies.

Keywords	Qt, local connectivity, Bluetooth, NFC, QML

Contents

Abbreviations

API	Application Programming Interface
IDE	Integrated development environment
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NFC	Near Field Communication
PC	Personal Computer
QML	Qt Markup Language
RF	Radio Frequency
RFID	Radio Frequency Identification
RTD	Record Type Definition
SIG	Special Interest Group
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
USB	Universal Serial Bus

1 Introduction

Qt markup language is a cross platform application development framework provided by Digia. Due to Qt, coding efforts have been drastically reduced though intuitive APIs that deliver more functionality from less code. The Qt development environment which is the Qt Creator is free and can be downloaded freeely from the internet. Qt in itself is a cross-platform application development framework. This means that binaries from the same source code can be deployed to different platforms. Currently these platforms are windows, Symbian MeeGo, Android, QNX and Mac OS. [1, 6-9.]

NFC is a short-range radio technology that facilitates communication between devices that touch or are momentarily held close together. It makes transactions, exchange of digital content and connecting to electronic devices possible. NFC harmonizes today's diverse contactless technologies. Bluetooth is a wireless communication technology that boosts low energy consumption and high speed transfer of the data between the devices communicating between themselves. [8, 5]

Everyday Qt framework gains popularity and there is expected to be a supersonic acceleration of applications developed using this framework. The purpose of this project was to demonstrate how to transfer a user-triggered action using NFC and through Bluetooth from one phone to another in real time. This project mainly dealt with development of the NFCTicTacToe game that explained the QML, NFC and Bluetooth technologies interactions. So as to facilitate the interaction of the game between two users using two different Symbian Nokia 603 mobile phones, the interaction of the game was initiated by a friendly recognition which was done using NFC and then the user input was transmitted from one user to the next and vice versa using Bluetooth.

This project is aimed at developers with a basic knowledge of Qt and will give an insight of the functionality of NFC, a Bluetooth enabled application. The purpose of my choosing of this topic was due to the fact that I was relatively new in Qt, Bluetooth and NFC technologies, which I found interesting. After studying the Qt framework and NFC technology, I thought it would be a good opportunity for me to gain a better understanding of the framework and the technology.

2 Qt Framework

2.1 Overview

A cross-platform application framework is a software framework specifically designed to help developers develop applications that run on multiple platforms. These frameworks provide the essential functionalities that are common to the platforms that they support. This eliminates the headache associated with developing an application over and over again for each new platform. Most of these frameworks provide libraries that are in support of the platform they support. [1, 7]

A Qt framework is an excellent example of a cross-platform application framework and it will be discussed here since it has been used to develop this project. The Qt framework is the main development framework for Symbian phones and the Nokia N9 smartphone. Qt, which is pronounced as cute, is a product of the Qt software. It is based on the initial idea of creating an object-oriented presentation system. Its first public release was done twelve years ago and since then it has seen a radical recognition in recent years being used by many customers such Google, Skype, Volvo, to name just a few. [1, 7]

Qt was first targeted at Windows, MAC OS and Linux systems only. In 2000 it was extended to embedded systems and in 2006 it saw its first launch of the fully featured mobile phone based on Linux. In 2008 Nokia introduced Qt to its platforms starting with the Symbian and Linux platforms, which gave Qt programmers access to the mobile domain. This reduced the number of hurdles in development of mobile application in addition to the existing programming language Java, Symbian C++ and Python. For Qt to use a full range of mobile functionality new Qt mobility APIs were introduced, which are responsible for using localization information, mobile messaging, cameras, in-built sensors and much more. The goal of Qt in future is to introduce it everywhere in electronic devices. [1, 8]

2.2 Features of Qt

The Qt technology is built upon a set of core technologies which include:

- The Tulip Container Classes, which contains a set of template container classes.
- The Arthur Paint System, which is the Qt 4 painting framework.

- The Interview framework, which is a model/view architecture for item view and the Qt SQL module.
- The Scribe Classes, which is a framework for creating text documents, performing low-level text layout and writing Open Document files.
- A collection of common desktop widgets that are styled to fit in on each support platform.
- The Qt 4 main window classes, which are: a main window, toolbar, menu and docking architecture.
- The QtNetwork module, which provides support for TCP, UDP and local sockets that are integrated with Qt's event model, including support for Secure Socket Layer(SSL) communications, network proxy servers and network bearer management.
- The enhanced thread support which allows signal-slot connections across threads and per-thread event loops. And with additional support of a framework for concurrent programming using Qt paradigms makes common threading tasks easier.
- A resource system for embedding images and other resource files into executable files which make it easier to deploy applications.
- A Unit testing framework for Qt applications and libraries. [2]

The above technologies make up the features for the Qt technology, and they are contributing a lot to the advancement of this platform.

2.3 Model/View Architecture

The model/view architecture for Qt applications is provided from the interview classes, based on the Model-View-Controller design pattern. In this architecture, the view and the controller objects are combined. Still there is the separation of the way data is stored from the way it is presented to the user, but this separation provides a simpler framework based on the same principle. The separation makes it possible to display the same data in several different views and to implement new types of views, without changing the underlying data structure. [3]. Figure 1 below shows the lifecycle of the model/view architecture.

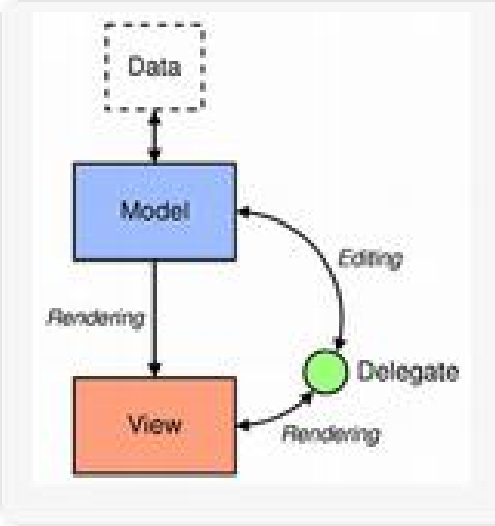

Figure 1. Model/View architecture. Reprinted from The Interview Framework [3]

The **Model** communicates with a source data, providing an interface for the other components in the architecture. Communication depends on the type of data source and the way the model is implemented, as shown in figure 1 above. The **View** obtains the model indexes from the model which are references to items of data. By making the model indexes available to the model, the view can retrieve items of data from the data source. As shown in the diagram above. A **Delegate** renders the items of data. When an item is edited, the delegate communicates with the model directly using model indexes. [3]

2.4 Qt Quick

Qt Quick is a collection of technologies that are designed to help developers create the kind of intuitive, modern and fluid user interfaces that are increasingly being used on mobile phones, media players, set-top boxes and other portable devices. Qt Quick consists of a rich set of user interface elements, a declarative language for describing user interfaces and a language runtime. A collection of C++ APIs is used to integrate these high-level features with classic Qt applications. [4]

2.4.1 The QML Language

QML is a high-level, scripted language. It commands, more correctly the arrangement of element, leverage the power and efficiency of the Qt libraries to make it easy to use

commands that perform intuitive functions. The drawing of a rectangle, displaying an image and QML application events are all made possible using declarative programming [4].

This also allows more flexibility of its commands by using JavaScript to implement the high-level user interface logic. A QML element has various properties that help define the element. The user interface can be built by importing the elements and is one of the outstanding features of QML and Qt Quick. [4]

2.4.2 Functionality of QML and C++

A QML file containing each element is backed by a C++ class.The QML code is interpreted by the QML engine. One-to-one relation can be established between QML and Qt. The QML engine creates one C++ object for all the elements in a QML file that are loaded. Listing 1 below is extracted from the project to help explain how QML and C++ work. The code is shown below.

```
Rectangle {
        id: charedit
        color: "silver"
        width: parent.width
        height: 30
        anchors.top: display.bottom
        TextEdit {
        readOnly: true
        id: textEdit
        text: "Welcome"
        color: "black"
        anchors.fill: parent
        }
    }
```

Listing 1. Ploting a rectangle with a name Qt code

As listing 1 above illustrates there are two elements: Rectangle and TextEdit. These two correspond to the C++ classes QQuickRectangle and QQuickTextEdit. The C++ classes are exported to QML because they are private and are not directly available to the users of Qt.

When loading the QML file in the QML engine three steps happen sequentially: parsing, compiling and creating. Parsing is handled by the QQmlScript::Parser. Most of the parser internals are auto-generated from a grammar file. The abstract syntax tree is low-level and is therefore transformed into high level structure of objects, properties and Values in the following stage. These are done using a visitor on the abstract syntax tree. At this stage the objects correspond to QML elements, properties and value pairs

to QML properties and values, such as color, width and height. The objects, properties and values created in the parsing phase still require post-processing which is done by QQmlCompiler. The compiler is responsible for creating the QQmlCompiledData object for the QML file. Examining the QQmlCompiledData and creating C++ objects is considerably faster than examining the objects, properties and values. Compiling and parsing a QML file is only done once, after which the QQmlCompiledData object is used to quickly create the C++ objects. [18]

Instructions for creating C++ objects and assigning the correct values to the properties are compiled to a bytecode. The bytecode is then interpreted by a bytecode interpreter. In the creating phase, the bytecode is interpreted by the class QQmlVMW. When a QML file is compiled, then creating an instance of it is easy. The bytecode of the complied data is just executed. [18]

2.4.3 QtDeclarative Module

The module provides a declarative framework for building highly dynamic and custom user interfaces [5]. It creates a JavaScript runtime that QML runs under with a Qt based backend. Since QtDeclarative and QML are built upon Qt, they inherit most of Qt's technology, namely signals and slots mechanism and the meta-object system. The data created using C++ is directly accessible from QML and QML objects are also accessible from the C++ code. [4]

2.4.4 QML Custom Element

QML is a declarative language where all user interface items are declared. These items are stored in a QML file and are loaded with the code in listing 2 shown below.

```
QmlApplicationViewer viewer;
viewer.setMainQmlFile(QLatin1String("qml/myproject
/main.qml"));
```

Listing 2. Loading items from a qml file using Qt code.

As shown in listing 2 the base class of QmlApplicationViewer is the QDeclarativeView which provides a widget for displaying a Qt Declarative user interface. In listing 2 main.qml is the file that contains the QML element, as shown in section 2.3.2

QML relies heavily on Qt's meta object system and can only instantiate classes that are derived from QObject. For visual element types, this will usually mean a subclass of QDeclarativeItem, for models used with the view elements, it will mean a subclass of QAbstractItemModel, and for arbitrary objects with properties, it will mean a direct sub-class of QObject. The QML engine has no intrinsic knowledge of any class types. Therefore the programmer must register the C++ types with their corresponding QML names.

The custom C++ types are registered using a template function as shown in listing 3 below:

```
template<typename T>

int qmlRegisterType(const char *uri, int versionMajor,
int versionMinor, const char *qmlName)
```

Listing 3. Registering C++ types in Qt

As illustrated in listing 3 above, when calling qmlRegisterType() registers, the C++ type T with the QML system makes it available in QML under the name qmlName in the library uri verison versionMajor.versionMonior. The qmlName can be the same as the the C++ type name.

Listing 4 below shows how CppItem class can be used in QML lannguage as a qml element.

```
class CppItem : public QObject
 {
     Q_OBJECT
     Q_PROPERTY(int root READ root WRITE setRoot NOTIFY
rootChanged REVISION 1)

 signals:
     Q_REVISION(1) void rootChanged();
 };
```

Listing 4. A C++ item being used in QML as a qml element in Qt.

Listing 5 below illustrates how to register a new class version to a particular version using a function.

```
template<typename T, int metaObjectRevision>
 int qmlRegisterType(const char *uri, int versionMajor,
int versionMinor, const char *qmlName)
```

Listing 5. Registering a new class version in Qt

Listing 6 below shows how to register CppItem version 1 for MYModule 1.1

```
 qmlRegisterType<QCppItem,1>("MyModule", 1, 1, "CppItem")
```

Listing 6. Registering C++ item version 1 for module 1.1 in Qt.

Listing 7 below shows how to include MyModule 1.1 in the QML file

```
import MyModule 1.1

 CppItem {

id: mycppItem

 }
```

Listing 7. Include MyModule 1.1 in a qml file in Qt.

While loading a QML scene into a C++ application, it is useful to directly embed C++ data into the QML object QDeclarativeContext, which enables the exposure of data to the context of a QML component which allows data to be injected from the C++ into QML.

2.4.5 Qt Creator

Qt Creator is a complete integrated development environment (IDE) for creating applications with Qt Quick and the Qt application framework. The main objectives of Qt Creator are fulfilling the development needs of Qt Quick developers who are looking to simplify, usability, productivity, extendibility and openness. The main features of Qt Creator allow the UI designers and developers to accomplish the following tasks:

- Getting started with Qt Quick application development quickly and easily.
- Designing application user interface with the integrated editor, Qt Quick Designer, or using graphics software to design the user interface and use scripts to export the design to Qt Quick Designer

- Developing applications with the advanced code editor that provides new powerful features for completing code snippets, refactoring code and viewing the element hierarchy of QML files.
- Building and deploying Qt Quick applications targeting multiple desktop and mobile platforms.
- Debugging JavaScript functions and executing JavaScript expressions in the current context, inspecting QML at runtime to explore the object structure, debugging animation and inspecting colors.
- Deploying applications to mobile devices and creating application installation packages for Symbian and Maemo devices that can be published in the Ovi Store and other channels.
- Accessing information easily with the integrated context-sensitive Qt help system. [4]

3 Near-Field Communication (NFC)

Near-field communication (NFC) enables connectivity between devices when they are in physical contact or within a range of a few centimetres. It is an ultra-short range wireless technology that uses magnetic field induction to enable connectivity [9, 287].
The following can be said about NFC:

- It is an open-platform technology that is being standardised in the NFC Forum
- It is based on and extends on RFID and it operates on 13.56 MHz frequency
- The communication range is up to 10 cm. However for the best experience with Nokia phones it is recommended that the devices touch each other.
- The NFC standards support different data transmission rates such as 106 kbps, 212 kbps and 424 kbps. [8, 5]

3.1 Tag and Reader

For NFC communication to be established between two devices, one device has to act as a reader/writer and the other as a tag. The tag is a simple, thin device containing an antenna and a small amount of memory .It is a passive device powered by a magnetic field. The memory can be read only, rewritable or writable only once depending on the tag type. Figure 2 below shows the tag.

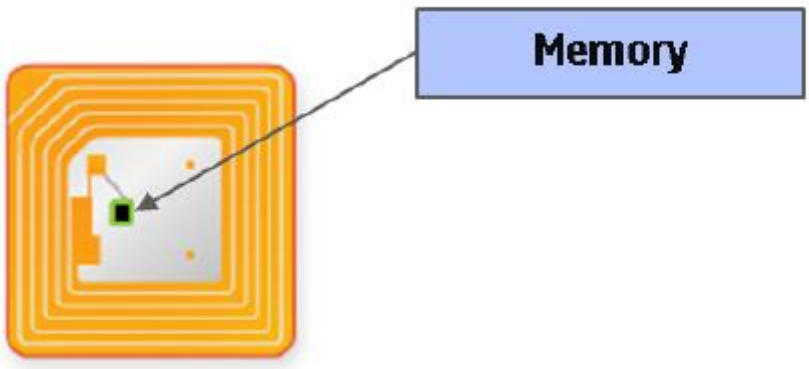

Figure 2: Tag. Reprinted from Introduction to NFC [8, 2]

The reader is an active device that generates radio signals to communicate with the tags. The passive device is powered by the reader when the two are engaged in the

passive mode of communication. Figure 3 below shows the reader in an NFC-enabled phone communicating with the tag. [8, 2]

Figure 3: Reader communicating with the tag. Reprinted from Introduction to NFC [8, 2]

The NFC-enabled phone will only communicate with the tag when they are within a close range.

3.2 The NFC Modes

The NFC devices support two modes: the communication mode and the operation mode. The communication modes are used by an NFC device to detect another NFC device, while the operating modes are used by NFC devices to transfer data between them.

3.2.1 Communication Modes

NFC-enabled devices support two communication modes: the active and passive mode. In the active mode the target and the initiator devices have power supplies and communicate with one another by alternate signal transmission. In the passive mode the initiator device generates radio signals and the target device is powered by the electromagnetic field. The target device responds to the initiator by modulating the existing electromagnetic field. [8, 7]

3.2.2 Operating Modes

The NFC devices communicate in three different modes on the ISO/IEC 18092, the NFC IP-1 and the ISO/IEC 14443 contactless smart card standards. The three operating modes are: read/write, peer-to-peer and card emulation.

In the read/write mode the NFC-enabled phone can read and write data to any of the supported tag types in a standard NFC data format. In the peer-to-peer mode the NFC-enabled devices are able to exchange data such as Bluetooth or Wi-Fi link setup parameters to initiate a Bluetooth or Wi-Fi link. It is also possible to exchange data such as virtual business cards or digital photos. The Peer to peer mode is standardised on the ISO/IEC 18092 standard. Currently Symbian implementation of NFC supports initiation of a Bluetooth link while as for Wi-Fi it is still not supported. In the card emulation an NFC enabled phone acts as a reader when in contact with tags. While in this mode the phone can act as a tag or contactless card for existing readers. This mode is still not supported by Symbian for NFC. [8, 8]

3.3 NFC Forum Tag Types

The NFC tags are usually used in applications where a small amount of data can be stored in and transferred to active NFC devices. The NFC tags can store any form of data. To facilitate interoperability between NFC tag providers and NFC device manufactures, the NFC Forum, which is a consortium which advances the use of NFC technology by developing specifications, ensuring interoperability among devices and services and educating the market about NFC technology, has defined four tag types: Type 1 tag, Type 2 tag, Type 3 tag and Type 4 tag. [8, 11]

Type 1 tags are quite cost-effective and ideal for many NFC applications. They are based on the ISO-14443A standard with a read and rewrite capability. They also allow users to configure the tags to be read only. The memory of these tags is 96 bytes but is expandable up to 2kB. The communication speed is 106 kbits. These tags are protected against data collision. Some of the compatible products available on the market are Innovision Topaz and Broadcom BCM20203. [8, 11]

Type 2 tags are derived from the NXP/Philips MIFARE ultra-light tags. These tags are based on the ISO-14443A standard and have a read and write capability. The memory is 96 bytes but expandable to up to 2 kB. They have a communication speed of 106kbits/s and are supported against anti-collision. NXP is a compatible product of this tag that is available on the market. [8, 12]

Type 3 tags are derived from the nonsecure parts of Sony FeliCa tags but are more expensive than tag Types 1 and 2. They are based on the Japanese Industrial Standard (JIS) X 6319-4.During manufacture they are preconfigured to be either read and rewritable or read-only. They have a variable memory of up to 1 MB per service. It is in support of two communication speeds: 201 or 424 kbits. These tags are anti-collision supported. Sony Felica is a compatible product, available on the market,and uses tag type 3. [8, 12]

Type 4 tags are similar to Type 1 and are derived from the NXP DESFire tag. They are based on the ISO-14443A standard. During manufacture these tags are preconfigured to be both read and rewritable or read only. They have a variable memory of up to 32 kB per service. They support three different communication speeds: 106, 212 or 424 kbits. They are supported against collision. NXP DESFire and SmartMX-JCOP are some of the compatible products available in the market. [8, 12-13.]

3.4 NXP-specific Tag Type

NXP-specific tag is defined by NXP Semiconductors and is a proprietary tag. Type MIFARE Classic Tag is a tag of this type and is based on the ISO-14443A standard, which is read and rewrite capable but also allows users to configure it to be read-only. It has a variable memory of 192/768/3584 bytes. The communication speed is 106 kbits. It supports against collision. Some of the compatible products available on the market are: NXP MIFARE Classic 1K, MIFARE Classic 4K and Classic Mini. [8, 13]

3.5 NFC Forum Standards

The NFC Data Exchange Format (NDEF) is a data format specified by NFC Forum for transferring data to and from tags between NFC devices. NFC is a lightweight and compact binary format that can carry URLs, vCards and NFC-specific data types. It allows NFC functionality to use the supported tag type to transfer data, because it

shields all the tag type-specific details from the application. A message consisting of a sequence of records can be exchanged in NDEF. Each record consists of a payload which can contain records of type URL, MIME media or an NDF-specific data type. For the NFC-specific data types, the payload contents must be defined in an NFC Record Type Definition (RTD) file. [8, 17]

The type of data in the record and the size of the record are indicated in a header attached to the payload. The header includes a type field for identifying the type of payload. The user applications to identify the payload carried within an NDEF record is accessed through the optional payload identifier. The type name format (TNF) field is used to indicate the format of the TYPE field value. Figure 4 below illustrates the NDEF message. [8, 13]

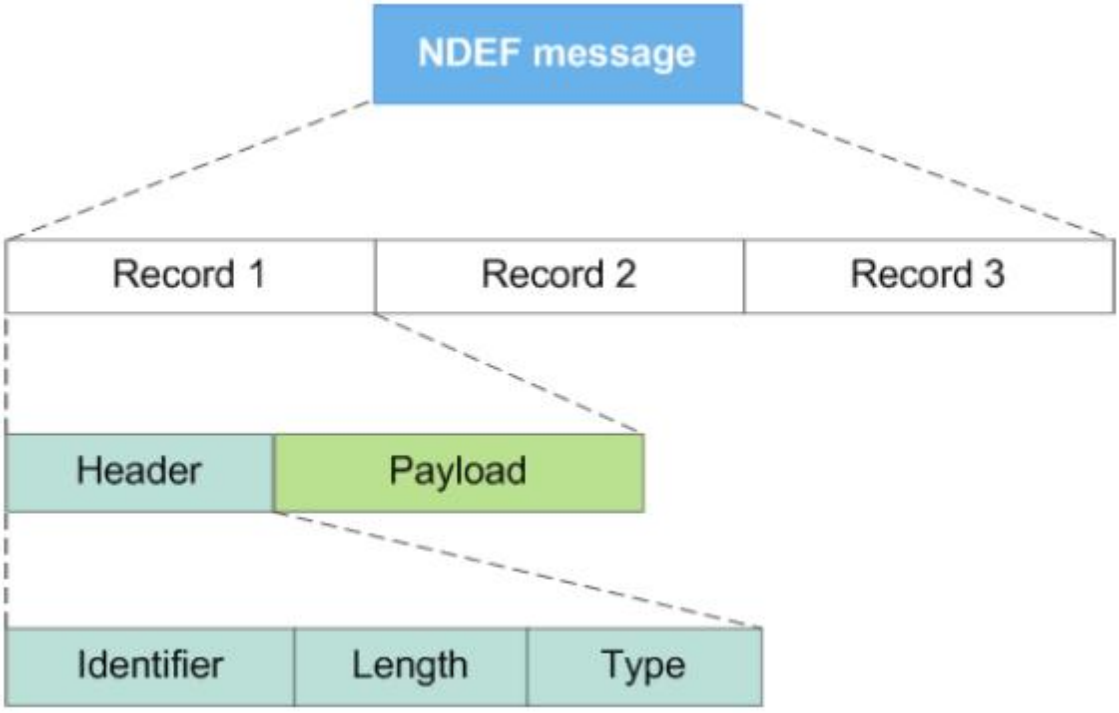

Figure 4: NDEF Message. Image Reprinted from Introduction to NFC [8, 18].

Figure 4 above shows the structure of the NDEF message.

There are several optimised record types that can be carried in NDEF records as specified in the NFC Forum. The Record Type definition (RTD) document specifies each NFC Forum record type. The following RTDs are defined in NFC:

- NFC Text RTD
- NFC URI RTD
- NFC Smart Poster RTD
- NFC Generic Control RTD
- NFC Signature RTD

The Text record type is the simplest, which can carry a Unicode string. The NDEF message can include a text record as a descriptive text for another record. A URI such as a web address, an email or a phone number in an optimised binary form can be stored in the URI record type. The URLs, SMSs or phone numbers on an NFC Forum tag and ways to transport them between devices are defined in the Smart Poster RTD. [8, 19]

The Logic Link Control Protocol (LLCP) is a link-level protocol specified in the NFC Forum to enhance the peer-to-peer mode of operation. The LLCP provides additional communication capabilities on top of the NFCIP-1/ ISO 18092 in the NFC peer-to-peer stack.

A two way link-level connection is introduced in LLCP. This allows peers to send and receive data using the following methods of data exchange:

- Connection-oriented transfer: Data exchanges are acknowledged.
- Connectionless transfer: Data exchanges are unacknowledged. [8, 19]

3.6 NFC in Nokia Phones

NFC-supported APIs provide a variety of Qt, Symbian and Java™ technologies. Only Symbian and Qt APIs will be discussed below since they are within the scope of this thesis. [8, 20]

3.6.1 Qt Application Programming Interfaces (APIs)

The Qt Mobility project has a cross platform API which includes the Qt NFC API. This API is contained in the connectivity API and integrates into the QT SDK. The Qt NFC API supports the following use cases: Interaction with the NFC Forum tags and NFC Forum devices, Active target detection and loss, Registering NDEF message handlers,

Reading and writing NDEF messages to NFC Forum tags, Sending tag-specific commands and Client and server LLCP sockets for peer-to peer communication [8, 20].

3.6.2 Symbian Application Programming Interfaces (APIs)

In Symbian the following use cases are supported by the Symbian APIs: Creating content handler plug-ins, Setting up a Bluetooth connection, Sharing a file or data, Reading NDEF messages, Discovering NFC tags, Exchanging data with NFC Forum Type 4 Tags, Reading NFC Forum Type 1,2,3, Tags, Writing to NFC Forum Type 1,2,3 Tags, Transferring and receiving “Hello World” ASCII text using LLCP stack (connectionless) and Transferring and receiving “Hello World” ASCII text using the LLCP stack (connection-oriented). [8, 23]

4 Bluetooth

The Bluetooth wireless technology is used to provide a short-range wireless link between notebook/laptop computers, mobile phones, PDAs and other personal portable electronic devices. It is based on specifications for low-cost, low-powered radio and associated protocol stacks. [10,151]

The Bluetooth wireless technology specifications are developed by the Bluetooth Special Interest Group. The Bluetooth wireless technology operates at 2.4 GHz spectrum for Industrial, Scientific and Medical (ISM) band at 2.4 to 2.4835 Ghz spectrum. A total of 79 sub-channels can be derived from this spectrum with 1 MHz each, with a hopping speed of channel to channel of 1,600 times per second. In order for the communication to be facilitated, the transmitting and receiving devices must be synchronized into the same hop of sequence. For the Bluetooth-enabled devices to communicate, they need to be typically close to one another, no more than 10 metres apart. [10,153]

For a Bluetooth connection to be established between Bluetooth-enabled devices, a setup phase has to occur, after which the Bluetooth devices exchange critical information about each other, such as technical features, manufacturer and clock offset. The Bluetooth controller due to its small size, combined with low cost and low power requirements, becomes suitable to be incorporated into electronic devices that are small in size. [10, 153-154.]

Bluetooth-enabled devices can be linked together in an ad hoc Wireless Personal Area Network Architecture (WPAN) called a piconet. They can be organized into a group of 2-8 devices. A piconet can consist of one or more slave devices but only a single master device. [10, 153]. A typical piconet is illustrated in figure 5 below.

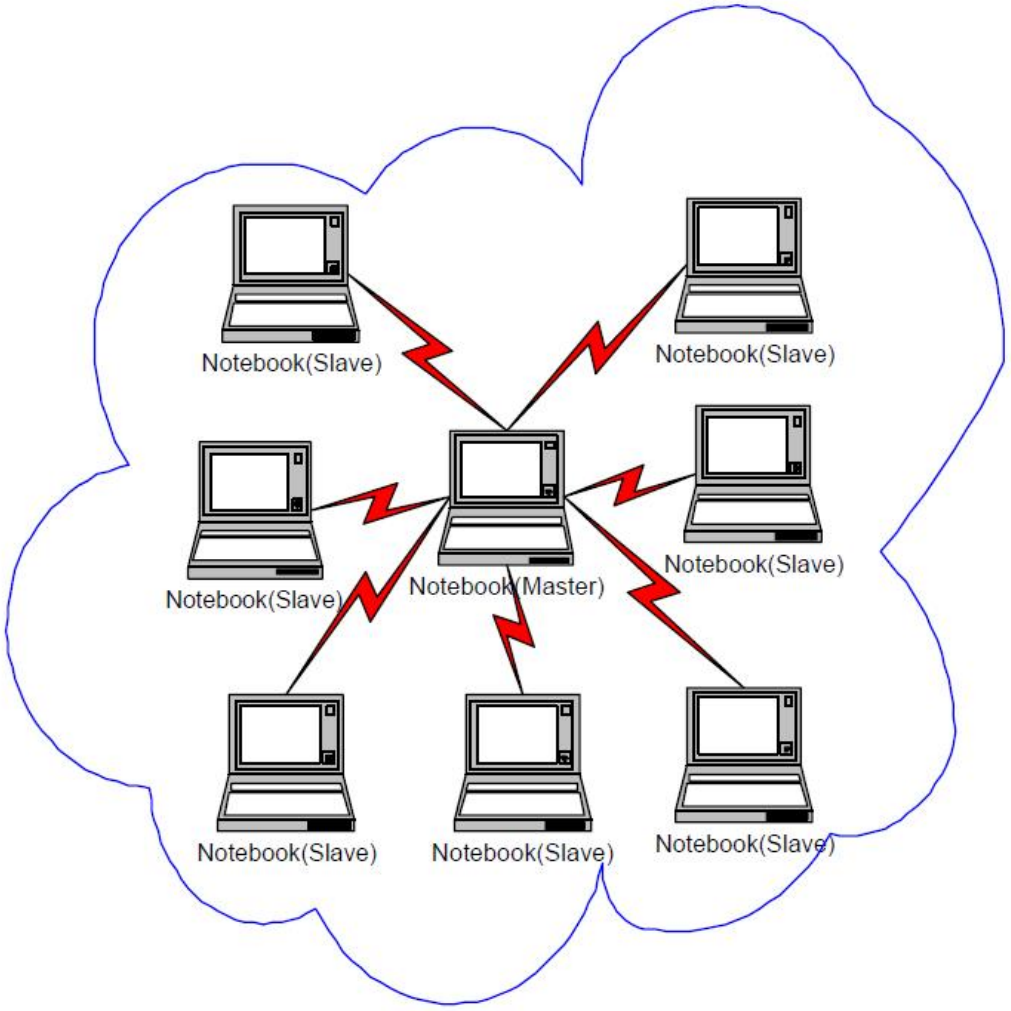

Figure 5: Single Piconet for a Personal Area Network Architecture. Image Reprinted from Personal Area Networking [11, 15].

Figure 5 above illustrates a personal area network architecture of a piconet, illustrating how one master can communicate with several slaves.

4.1 Bluetooth Protocol Architecture

A protocol stack is created when the hardware and software components are interconnected via USB or PC card physical buses. Figure 6 below illustrates different layers of a Bluetooth protocol stack. The protocol layer can be coupled into a seven-layer OSI model.

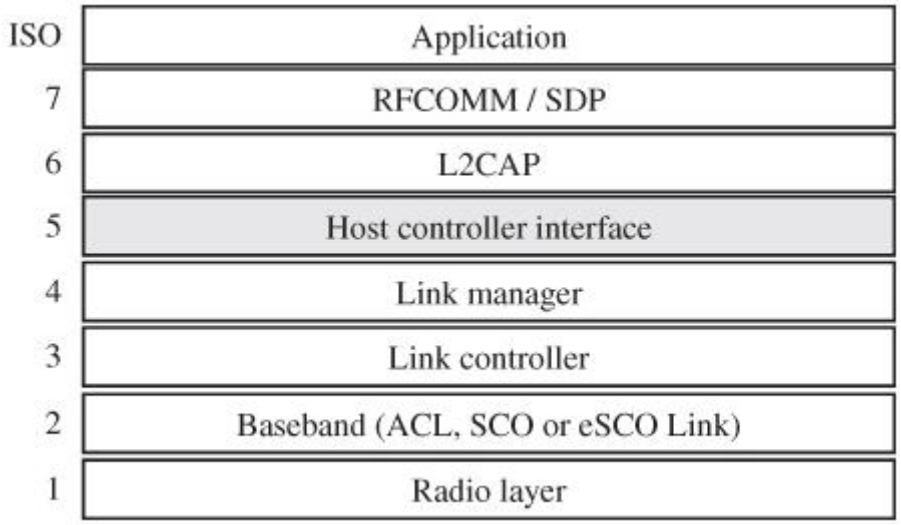

Figure 6: Bluetooth protocol Stack. Image Reprinted from The Bluetooth Protocol Stack [12].

The radio layer, baseband, link controller and link manager form the hardware portion, which is used to set up and control the link. The Bluetooth data link-level security is implemented in the link manager.

For a Bluetooth connection to be established between two devices that are close together, the user has to initiate a Bluetooth connection. The link is used to transfer data between the devices. For a connection to be initiated the user opens Bluetooth client software on one of the devices and “discovers” nearby devices. A device can be configured so that it cannot be discovered for security purposes. The following discovery and pairing processes can be initiated:

- Name discovery: The Bluetooth device name is detected
- Service discovery: Only specific services available from the device are discovered; the services are defined in the Bluetooth specifications. [10,155]

For a connection to be successful both devices have to support the same Bluetooth profile such as synchronization. “Pairing” which is also referred to as “bonding” is said to have taken place when two devices form a private connection. The pairing part is optional and for security purposes a PIN is requested for the connection to be allowed. [10, 154-155.]

4.2 Bluetooth Profile

Devices that have the same SIG adopted profile are the ones that can communicate together. The standards of Bluetooths enable developers to create applications to work with devices that conform to Bluetooth specifications. A list of the Bluetooth profiles can be found in Ghetie, [2008] [10, 155-156.]

The operating systems that are supported by Bluetooth are: Linux, MAC OS X V10.4, Palm OS, Windows Mobile (WM) and Windows Vista. Some specifications have multiple functions inbuilt in them. To execute a particular task, the assigned options and parameters are used at each layer of the Bluetooth stack. The Bluetooth wireless technology uses the unlicensed spectrum of 2.4 GHz, which is also used by other technologies such as cordless phones, microwave ovens and 2.4 GHz Wi-Fi networks (802.11b and 802.11g). This may cause some problems when these networks operate at close proximity with each other. [10, 155-156.]

4.3 Bluetooth Security

All types of communications face some degree of security concern and threats ranging from issues of privacy and identity theft to denial of service attacks. The vulnerability of Bluetooth comes from the use of wireless communication where eavesdropping can be easily performed, that is, packets can be captured, and if no security measures are put in place, Bluetooth phones can be hacked. For Bluetooth the security solutions are derived from precautions that range from pairing devices, use of authentication mechanisms with longer passwords or PIN that range from 8-12 characters and encryption of passwords and user data. Point-to-point links between well-known pair devices are the most common Bluetooth configurations. Pairing indicates easy recognition of the partner in Bluetooth exchange of information based on common profiles and maybe even the same pin number. Encryption algorithms are applied along with security key management, in addition to pairing. Also Bluetooth's short range helps to minimize threats. [10, 158]

In the Bluetooth technology there are two major security keys that are used. The first one is the link security key which is made up of a 128-bit random numbers. The main role of this key is to establish if the two communicating devices have had a previous relationship. If no such relationship ever existed, the key is generated. Whenever the encryption procedure is requested, an encryption key is generated. The combination

key is the second major key which is generated as a combination of two connecting Bluetooth devices. So for every new combination of devices a new key is generated. Whenever a Bluetooth device is installed, a unit key is generated this remains the same for the lifetime of the Bluetooth device. The unit key is not considered a safe key for encryption. Another big blow to the Bluetooth security and availability is interference from overlapping wireless networks and accidental RF noise. [10, 158]

5 Interaction of QML and C++

The Duo Tic-Tac-Toe game utilized the feature of QML which enables extensibility to and from C++ via the Qt's meta-object system, where C++ objects and QML communicate through Qt signals and slots. The connectivity API was used, which enables the communication of Bluetooth and NFC. The combination of QML and C++ was undertaken so as to achieve the following:

- To be able to use the functionality defined in the C++ in QML
- To be able to access the functionality in the Qt Declarative module
- To be able to write self-defined QML elements.

The Qt Declarative module features have to be included and linked appropriately if they have to be used. The module provides a declarative framework for building highly dynamic and custom-user interfaces. As shown in listing 8, a declarative was added to the C++ file so as to define the context within a QML engine.

```
#include <QtDeclarative/qdeclarativecontext.h>
```

Listing 8. Defining context within QML engine in Qt

Listing 9 shows how to link against a module which is added to the .pro file.

```
QT += declarative
```

Listing 9. Linking against module in .pro file in Qt.

This module provides a set of C++ API's for extension of QML applications from C++ and also for embedding QML into C++ applications [15].

The C++ header and source file are linked to QML through the inclusion of the header files path and the source file path in the .pro file under the header and source section respectively. These files are generated automatically when the project is built and run. Listing 10 below shows how the instances are used to call the connection function in the connectionEngine.cpp file in the QML file.

```
connectionEngine.openConnection()
```

Listing 10. Instance to link the connection function to the connection.cpp file in Qt.

For the functions in a C++ file to be accessed directly in the QMl file, the new instances of the corresponding files need to be created first.

5.1 Qt Signals and Slots

Signals and slots are used to communicate between objects. The emitting of a signal occurs when a particular event occurs, while a slot is a function which is called in response to a particular signal [17]. In simple words a slot is a normal function of a class which responds to signals, while signals are emitted by objects. A signal from an object can be linked to one or several slots of a single receiving object or of several different receiving objects. In an event an object sends out a signal and all the slots linked to the signal are called. If there is no matching link, nothing will happen. As illustrated in listing 11, when QObject::connect() is called it links the clicked() signal of the QPushButton object with the quit() slot of the QApplication object. Whenever the user presses the button, a clicked() signal is sent out, thus causing the clicked function to be called. In reiteration to this signal the quit() function is called, which ends the event loop and consequently the entire application. When QObject::connect() function is used to link signals and slots, the marcros SIGNAL() and SLOT() must be used as shown in listing 11.[16, 36]

```
#include<QApplication>
#include<QPushButton>
int main(int argc, char *argv[]){
QApplication a(argc, argv);
QPushButton(“Quit”);
button.show();
```

```
QObject::connect(&button,    SIGNAL(clicked()),    &a,
SLOT(quit()));
return a.exec();
}
```

Listing 11. Linking signals and slots to quit function in Qt. Reprinted from Daniel (2007) [16, 35]

Figure 7 below illustrates how a particular signal from one object is directed to a slot of another object.

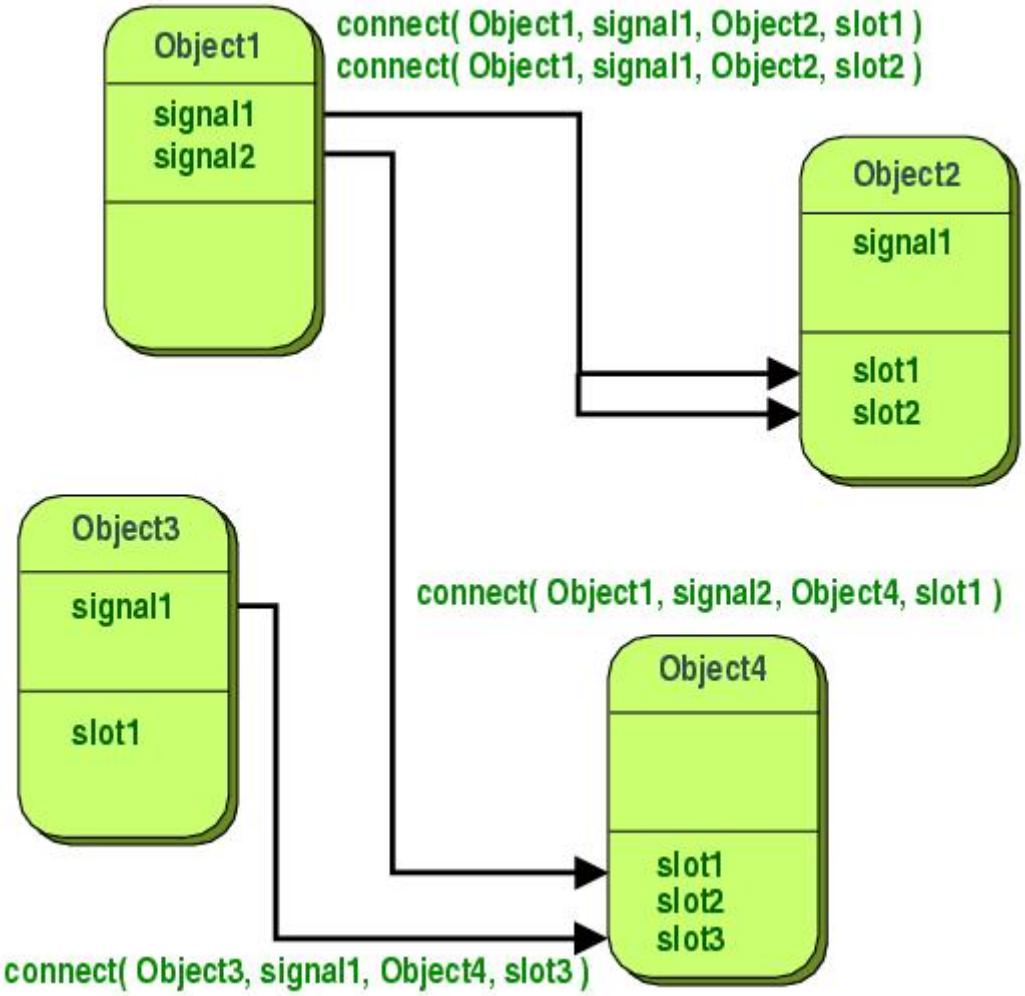

Figure 7: Signal Slot Signal routing. Image Reprinted from Signals & Slots [17].

To be able to access the signals in the C++ file, the QML connection is used, which creates a connection to a QML signal. To connect the signals in QML an “on<Signal>” handler that reacts when a signal is received is used. [17]

6 Setting up the Environment

For a developer to develop a Qt/QML application for a Symbian phone, the developer has to create an environment that will aid in achieving this task. This project focused on illustrating local connectivity using NFC and Bluetooth, through a game called the Duo Tic-Tac-Toe. The will be supporting two users using two different Nokia 603 Symbian phones, which connect locally using NFC and transfer the data using Bluetooth.

6.1 Installing the Qt Creator

In order to be able to install Qt one needs to download the Qt SDK, which includes the tools needed to build a desktop, embedded or mobile application with a single install [6]. The Qt SDK contains:

- Qt libraries
- Simulator for Symbian phones and the Nokia N9
- Qt Creator IDE
- Qt Mobility
- Qt development tools
- Remote compilers.

The latest version of Qt SDK version 1.2.1 which is available at: URL:http://qt-project.org/downloads. was downloaded. The URL contained the online installer for windows, linux and mac. Windows was selected since the project was developed using windows operating system. After the download was finished, the downloaded QtSdk-online-win-x86-v1_2_1.exe was opened by double clicking, it contained the installation instructions. The default installation folder which is C:\QtSDK was selected. The installation was relatively quick since a fast computer was been used. The installation was successful and the Qt creator was located in the start menu and the QtSDK in: C:\QtSDK.

6.2 Setting up the Symbian Phone

First and foremost the phones were switched on and then connected to the computer. This was done using the USB cable. The USB cable accompanies the phone on purchase. Windows 7 operating system was used in this project which installed all the required drives automatically when the phone was connected to the computer. QtQuickComponents-1.1-for-Anna-Belle.sis was installed on Nokia 603 which aided in running the Qt projects. The QtQuickComponents-1.1-for-Anna-Belle.sis was located in: c:\QtSDK>Symbian>sis>Symbian_Belle>QtQuickComponents>1.1 folder. CODA which is an on-device debugging agent was installed on the phone.

CODA was located at: C:\QtSDK>Symbian>sis>common>CODA>Public-CODA-1.0.6-for-S60v5-Anna-Belle-vFuture.sis. The file was installed by copying it to the phone. The installation was successful and icon named RnD Tools appeared among the applications of the phone, which is used to activate the CODA application.

6.3 Setting up the Qt Creator

After the CODA was successfully installed, the phone remained connected to the PC using the cable. Then the Qt Creator was opened to start the setup process. On the left hand side of the Qt Creator the project icon was located and clicked. This prompted the build settings windows to appear inside the Qt Creator. The Symbian device tab was selected which opened the Build tab. In the ‘Edit build configuration’ Debug was selected. Figure 8 below shows the Symbian build setting.

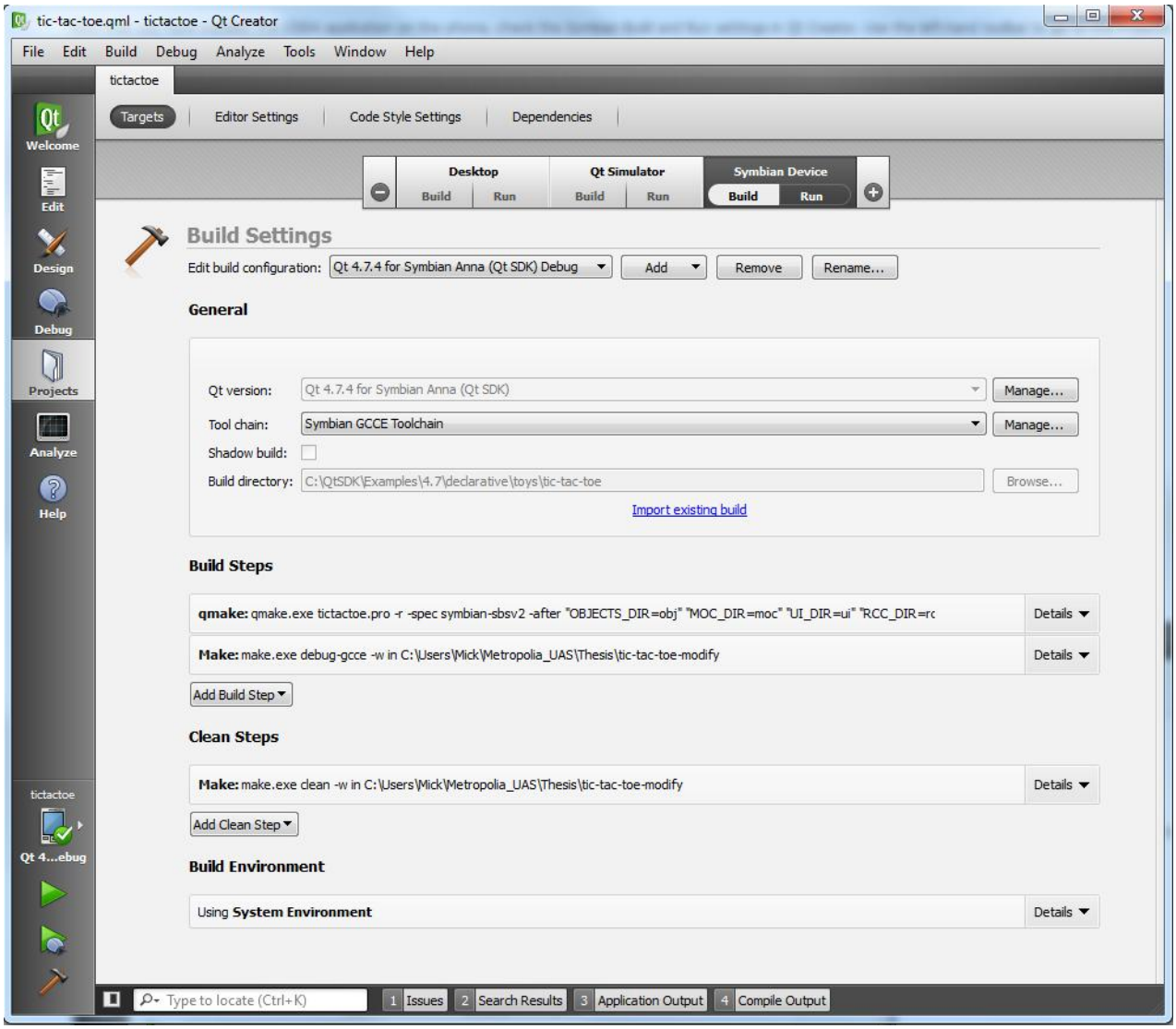

Figure 8: Screen shot of Symbian Build Settings using Qt Creator.

After the build configurations were through, the run under the Symbian device tab was selected. Then the selection of the serial port to which the phone was connected was done and then the self-signed certificate. After this step the Qt Creator and the Symbian phone were ready too start programming and running the applications respectively.

6.4 Application Functions

6.4.1 Introduction

The main purpose was to study Qt/QML and these interactions together to fulfil data passing between devices. This also explains how QML elements work together with Qt in the context of NFC usages, illustrating how they can be combined to transfer data between two mobile devices. The application was developed using the Qt Creator. The

devices that were used in the development process were the Nokia 603 handsets, so the application was deemed completely compatible with the Nokia 603 handsets. There were different fundamental classes which were implemented for the different task involved in the NFC and Bluetooth functionality, which are deemed completely necessary for the Duo Tic-Tac-Toe application to work perfectly.

6.4.2 NFC Bluetooth Communication

When the two NFC-enabled phones are brought into contact, there is the NFC acknowledgement, and then the NFC information of the two devices is exchanged. Once the NFC acknowledgment is done, searching for Bluetooth services begins. If the Bluetooth is deactivated, the activation process starts, after which the Bluetooth information of the corresponding phones is exchanged. Once the information is shared, a Bluetooth connection is established, hence no need of the NFC any more. In the project the NFC was used for acknowledgement and detection of the Bluetooth address. The Bluetooth address of a device is read and written to the other device. After receiving the Bluetooth address, the device starts searching for the service and when the service is found, it connects to the other device. At this point the Bluetooth connection is established and data can be transferred through the Bluetooth connection.

6.5 Application User Interface

The playing diagram and the user input were designed using Paint. The alignment and the user interface design were done using QML. Using paint the images of the game were easily developed as paint has the features supporting the drawing of images. Figures 9 and 10 below are screenshots taken when the software was running on the phone. These captured screens present the user interface of the game.

Figure 9: Game on start-up

Figure 9 shows the game when it is started. It displays the game, a welcome message and the option of refresh and close. This is the first screen to pop up when the application starts. The refresh button gives the option of clearing the game in case of existing data.

Figure 10: Player one selection

Figure 10 shows the same when one player has commenced the game. The first player is assigned an “X” when the game starts and this is the player who is the first to play the game. The second player is assigned an “O”. Player one selection is displayed on player one’s game and is transferred through Bluetooth to player two’s game. Then player one’s game becomes non-interactive as it is waiting for player two to play. When player two makes a selection, it is displayed on player two’s game and the subsequent result is transferred to player one’s game. On receipt of player two’s selection, player one’s game becomes active and player two’s game becomes inactive waiting for player one to play. Player two’s game is only activated when it receives player one’s input. This process continues until there is a winner or when the game ends.

7 Results and Conclusion

The project was aimed at studying Qt/QML, NFC and Bluetooth with the aim of combining the three technologies to come up with a local connectivity between two phones .A Qt/QML application was developed for the demonstration purposes of the local connection. The application had to support interaction between two users using two different Symbian phones in which interaction was supposed to be shared between the two Symbian phones with the aid of NFC and Bluetooth.

While developing the Qt/QML application, it was realized that it was easy to produce a user interface by using QML declarative language. An intuitive user interface can be developed fairly easily by using QML and it is easy to learn and implement. On the other hand application logic can be implemented using Qt C++ and those functionalities can be called from QML. How the functionalities work was illustrated by using the sample game developed in the project.

The result of this project enabled me to get a clear understanding of Qt/QML, NFC and Bluetooth technologies. Despite gaining commendable knowledge of these technologies, Symbian C++ is also something that I got an overview of as the NFC and Bluetooth implementation functionalities were mainly in C++. This project also gave guidelines on how an application is developed, compiled, run and deployed in phones using the Qt creator development environment.

The project was deemed a success as the application can support two users and the interaction can be transferred between the phones using a Bluetooth connection which is initiated by NFC. The project also laid a foundation for a deep understanding of the technologies used which were the bases of this thesis. The project can also be used as an illustration of how different technologies can be combined to produce a working product with diverse capabilities.

References

1. Fitzek H.P.F, Torp T, Mikkonen T. Qt for Symbian. United Kingdom: John Wiley & Sons, Ltd; 2010.

2. Digia Plc. Qt Features Overview [online]. Finland: Free Software Foundation; 2 February 2013.
   URL: http://doc.qt.digia.com/4.7-snapshot/qt-overview.html.
   Accessed 3 February 2013.

3. Digia Plc. The Interview Framework [online]. Finland: Free Software Foundation; 2 February 2013.
   URL: http://doc.qt.digia.com/4.7-snapshot/qt4-interview.html.
   Accessed 3 February 2013.

4. Digia Plc. Introduction to QT Quick [online]. Finland: Free Software Foundation; 2 February 2013.
   URL: http://doc.qt.digia.com/4.7-snapshot/qml-intro.html.
   Accessed 3 February 2013

5. Digia Plc. Qt Declarative Module [Online]. Finland: Free Software Foundation; 2 February 2013.
   URL: http://doc.qt.digia.com/4.7-snapshot/qtdeclarative.html#details
   Accessed 3 February 2013

6. Digia Plc. Download Qt, the cross-platform application framework [online]. Finland: Free Software Foundation, 2 February 2013.
   URL: http://qt-project.org/downloads.
   Accessed: 3 February 2013.
   ‘

7. Nokia Corporation. Qt – Getting started [online]. Finland: Nokia Corporation; 25 February 2012.
   URL: http://www.developer.nokia.com/Develop/Qt/Getting_started
   /Step_3.xhtml.
   Accessed: 17 October 2012.

8. Nokia Corporation. Introduction to NFC [online]. Finland: Nokia Corporation; 25 February 2012.
   URL: http://www.developer.nokia.com/dp?uri=http%3A%2F%2Fsw.nokia.com%2Fid%2Fbdaa4a0f-fcf3-4a4b-b800-c664387d6894%2FIntroduction_to_NFC
   Accessed: 24th October 2012

9. Rackley S. Wireless Networking Technology. United Kingdom: Linacre House; 2007.

10. Ghetie J. Fixed-Mobile Wireless Networks Convergence. United States of America: Cambridge University Press; 2008.

11. Institute of Electrical and Electronics Engineers, Inc. Personal Area Networking Profile [online]. United States:; 31 October 2012.
URL: http://grouper.ieee.org/groups/802/15/Bluetooth/PAN-profile.pdf.
Accessed: 28 November 2012

12. Safari Books Online, LLC. The Bluetooth Protocol Stack [online]. United States:; 17 March 2001.
URL: http:// my.safaribooksonline.com/book/networking/wireless /9780470978221/chapter-7-bluetooth/navpoint-95.
Accessed 28 October 2012

13. Nokia Corporation. Nokia Developer [online]. Finland: Nokia Corporation; 25 February 2012.
URL: http://www.developer.nokia.com/.
Accessed 28th October 2012

14. Digia Plc. Using QML Bindings in C++ Applications [online]. Finland: Free Software Foundation; 2 February 2013.
URL: http://doc.qt.digia.com/qt/qtbinding.html.
Accessed 3 February 2013

15. Digia Plc. Qt Declarative Module [online]. Finland: Free Software Foundation; 2 February 2013.
URL: http://doc.qt.digia.com/qt/qtdeclarative.html#details.
Accessed 3 February 2013

16. Molkentin D. The Book of Qt 4 The Art of Building Qt Applications. United States of America: William Pollock; 2007.

17. Digia Plc. Signals & Slots [online]. Finland: Free Software Foundation; 2 February 2013.
URL: http://doc.qt.digia.com/qt/signalsandslots.html.
Accessed 3 February 2013

18. Klaralvdalens Datakonsult AB. QML Engine Internals, Part 1: QML File Loading [online].Hagfors,Varmland: Thomas McGuire; 9 October 2012.
URL: http://www.kdab.com/qml-engine-internals-part-1-qml-file-loading/.
Accessed 6 December 2012

Appendices

Appendix 1: Code Implementation of main.cpp File

```
int main (int argc, char *argv[])
{
    QApplication app(argc, argv);
    QmlApplicationViewer viewer;
    QDeclarativeContext *m_context = viewer.rootContext();
    ConnectionEngine *connModel = new ConnectionEngine(&viewer);
    m_context->setContextProperty("connectionEngine", connMod-
el);
viewer.setOrientation
             (QmlApplicationViewer::ScreenOrientationAuto);
viewer.setMainQmlFile
             (QLatin1String("qml/NFCTicTacToe/main.qml"));
viewer.showExpanded();
return app.exec();
}
```

Appendix 2: Code Implementation of main.qml file

```
Page {
    id: mainPage
Rectangle{
    id: game
    property bool sharing: false
    property bool initialized: false
    property bool message: false
    property string player: "X"
    property bool running: true
    property int a:-1
    property string b:""
    property bool activate: false
    property string mess: "messa"
    color: "white"
    anchors.fill: parent
CommonDialog
  {
    id: mydialog
    anchors.centerIn: parent;
    titleText: "Touch with another NFC enabled Device"
    buttonTexts: ["Continue", "Exit"]
    z: 21
onButtonClicked:
   {
     if(index === 1)
      {
        mydialog.close();
        Qt.quit()
        }else{
        connectionEngine.openConnection()
        game.sharing = true;
      }
    }
  }//mydialog
Connections {
```

```
    target: screen
     }//screen
     Image {
         id: boardImage
         x: 0
         y: parent.height/2-parent.width/2
         width: parent.width
         height: parent.width
         source: "content/pics/board.png"
     }//boardImage
     Column {
         id: display
         x:0
         y:parent.height/2-parent.width/2
         Grid {
             id: board
             width: boardImage.width
             height: boardImage.width
             columns: 3
             Repeater {
                 model: 9
TicTac{
        width: board.width/3
        height: board.height/3
        onClicked:
        {
           if(game.sharing === true){
             if (game.running && Logic.canPlayAtPos(index)) {
                     if(game.player === "X"){
                            game.player = "X"
                       }
                         else{
                game.player = "O"
         }
      if(game.activate === false){
           game.activate = true;
           game.initialized = true;
```

```
            Logic.makeMove(index, game.player)
            connectionEngine.sendPlay(game.player, index)
                 }
             }
        }
        else
        {
          mydialog.open();
          }
       }//onClicked
    }//TicTac
 }//Repeater
}//Grid
Text{
    id: messageDisplay
    color: "blue"
    style: Text.Outline; styleColor: "white"
    font.pixelSize: 50; font.bold: true
    visible: false
    Timer{
        running: messageDisplay.visible
        onTriggered: {
        messageDisplay.visible = false;
      }//onTriggered
    }//Timer
  }//Text
 }//Display
Rectangle{
 id: textrect
  x:110
  y:parent.height- 35
Text {
    id: textEdit
    text: "Welcome, not connected!"
    color: "black"
    anchors.fill: parent
   }//TextEdit
```

```
 }//Textrect
Rectangle{
   id: textcon
   x:110
   y:parent.height- 35
   visible: false
Text {
    id: textEdita
    text: "Welcome,connected!"
    color: "black"
    anchors.fill: parent
  }//TextEdit
}//Textrect
/*************************Exit*****************************/
Rectangle{
   id: dirbackRect
   x: 5
   y: parent.height- 50
   width: 50
   height: 50
   MouseArea{
   anchors.fill: dirbackRect
   onClicked:
  {
     Qt.quit();
   }
}
Image{
   id: dirimageBack
   opacity: 1.0
    width: dirbackRect.width; height: dirbackRect.height;
    fillMode: Image.PreserveAspectFit; smooth: true
    source: "content/pics/hideexit.png"
 }//dirimageBack
}//dirbackRect
/*********************Reset*************************************/
Rectangle{
```

```
        id: resetRect
        x:55
        y: parent.height- 50
        width: 50
        height: 50
        MouseArea{
            anchors.fill: resetRect
            onClicked: {
                    Logic.restartGame();
           }
        }
    Image{
          id: dirimageReset
          opacity: 1.0
          width: resetRect.width; height: resetRect.height;
          fillMode: Image.PreserveAspectFit; smooth: true
          source: "content/pics/reset.png"
       }//dirimageReset
     }//resetRect
}//Rectangle
Connections{
    target: connectionEngine
    onReceived:{
        if(!game.initialized){
            game.player = "O"
            game.initialized =true
            }
        if(game.a !== undefined && game.b!== undefined ){
            Logic.makeMove(index, name);
           game.activate = false;
        }
    }
onNewConnection:{
        console.log("connection done")
        //textEdit.text = "Device connected..."
        textcon.visible = true;
        textrect.visible = false;
    }
```

```
}
}//Page
```

Appendix 3: Code Implementation of Function SendPlay

```
void ConnectionEngine::sendPlay(const QString name, int index) {
qDebug() << "ConnectionEngine::sendPlay: " + name + ": " + in-
dex;
QByteArray array;
array.append( name.length() );
array.append( name.toUtf8() );
QByteArray indexArray;
QByteArray temp = indexArray.number( index );
array.append( temp.length() );
array.append( temp );
if ( m_client ){
        m_client->writeSocket( array );
        qDebug()<<"client writing message.....";
    }
 else if ( m_server ){
        m_server->writeSocket( array );
        qDebug()<<"server writing message......";
    }
}
```

Appendix 4: Code Implementation of Function OpenConnection

```
void ConnectionEngine::openConnection()
{
    qDebug() << "ConnectionEngine::openConnection";
    if ( !m_aiwWrapper )
    {
        m_aiwWrapper = new NfcAiwWrapper();
        connect( m_aiwWrapper, SIG-
NAL(connectionInfoReceived(QByteArray)),
                this, SLOT(readConnectionInfo(QByteArray)));
    }
    m_aiwWrapper->startEasySetup();
}
```