OULU UNIVERSITY OF
APPLIED SCIENCES

Emmanuel A.E. Etchu

IMPROVING AUTOMATED TESTING OF S40 SOFTWARE

IMPROVING AUTOMATED TESTING OF S40 SOFTWARE

Emmanuel A.E. Etchu
Master's thesis
Autumn 2011
Degree program in Information Technology
(Master of Engineering)
Oulu University of Applied Sciences

ABSTRACT

OULU UNIVERSITY OF APPLIED SCIENCES

Degree programme	Thesis	Pages / enclosures
Degree Programme in Information Technology	Master's Thesis	46 / 0

Commissioned by	Author
Nokia Mobile Phones	Emmanuel A.E. Etchu

Thesis title

IMPROVING AUTOMATED TESTING OF S40 SOFTWARE

This Master's thesis was done for the Nokia Mobile Phones and specifically for Nokia S40 features testing projects. The thesis provides alternative methods of testing for the S40 features testing and releasing teams, and describes an automated version of the ISA TTCN testing environment

Owing to the fact that the original ISA TTCN environment made it possible to test not more than one mobile phone, to realise the aim of the Master's thesis therefore, it was necessary to modify ISA TTCN to support testing with two mobile phones connected to the ISA TTCN environment with the PC acting as the tester. The modification of the ISA TTCN environment to use two mobile phones subsequently made it possible to carry out an end to end and system level testing to automate a wide range of test cases based on key S40 features

The design was implemented using the C/C++ language for the ISA TTCN automated environment and the QT/QML programming language for result reporting. This approach favoured the S40 features testing and the releasing teams to realize the important milestones and increased the level of testing within S40 feature testing and the releasing teams by driving towards a software quality mind set with the current S40 product. The structure of building the automated testing environment was divided into three parts namely; the modification of ISA TTCN to support the end to end testing of lower layers, the plug-in support for system level testing with ISA TTCN and QML results reporting with the QT application. The outcome still carries significant potential for future upgrading and a wide utilization on the ISA TTCN automated environment tool.

Keywords

Keywords: Lower Layers, System Level, C/C++, ISA TTCN, Automated environment, S40 Features, S40 Releasing team, QML Programming.

FOREWORD

This Master's thesis is made for the Mobile Phones Product development department of Nokia. The work was supervised by Dr. Kari Laitinen, Principal Lecturer of Oulu University of Applied Sciences. The supervisors from Nokia were Mr. Arsi Liimatta, Lead Test Engineer and Mr. Pasi Heikkinen, Feature Maintenance Owner.

Oulu, Finland, autumn 2011

_____

Emmanuel A.E. Etchu

PREFACE

This work was done for the Mobile Phones Product development department of Nokia during summer 2010. The Nokia Mobile Phones and specifically Nokia S40 features projects in Oulu provided the subject and the requirement of the Master's thesis. The purpose of the thesis was to provide the Nokia S40 feature testing and the releasing teams with an ISA TTCN automated testing environment to facilitate the end to end and system level testing support.

Firstly, from the depth of my heart, I would like to thank God Almighty for providing me with the necessary strength, energy, courage and the will power to forge ahead after the death of my father, ARREY DANIEL TABE on the 13th of February 2007, may the Lord grant him perfect peace. With all profound love, I would like to thank my beautiful and caring wife Mrs. Arrey Marie Claudine for the moral and emotional support she provided me throughout the period of the thesis. Regards also go to my daughter Bianca Ulmonen.

I equally would like to express my sincere gratitude to my mother Mrs. Arrey Esther Besong for her endless encouragement throughout my school years. Special thanks go to my uncle Etchu Nicolas Ayuk, an engineer and Ashunchong Samson for providing me with great and helpful engineering advice as well as being my mentors.

Special thanks go to my supervisors; Dr. Kari Laitinen, Principal Lecturer of Oulu University of Applied Sciences, Mr. Arsi Liimatta, Lead Test Engineer and Mr. Pasi Heikkinen, Feature Maintenance Owner both from Nokia Ltd for their professional advice which helped realise this thesis.

To the Arrey family, big thanks go to Mr. Arrey Victor Arrendiep and Mr. Arrey Anthony for their fatherly support, and to my brothers and sisters (Marie Arrey, Emile Arrey, Valentine Arrey, Erica Arrey, Hernica Arrey, Derrick Arrey and Franklin Arrey) for their encouragement and best wishes. I am sincerely thankful to all my friends and the Forbi family.

To God be the glory.

Oulu, Finland, October 2011

Emmanuel A.E. Etchu

Contents

ABBREVIATIONS

ANSI C	American National Standards Institute (ANSI) for the C Programming language
ASN	Abstract Syntax Notation
ASN.1	Abstract Syntax Notation One
API	Application Interface
ATS	Abstract test suite
ASP	Abstract service primitives
CM	Communication Manager
C/C++	Computer programming language
DAU	Provides a connection from the serial port of the computer to the Pop-portTM connector of the mobile terminal
DB	Database
DKU	Nokia Data Cable
DT	Delegate Tester
ETSI	European Telecommunications Standards Institute
EQ Macro	Equalization macros
FIFO	First In, First Out
GUI	Graphical user interface
IEC	International Electrotechnical Commission
ISI	Intelligent Service Interface
ISA	Intelligent Software Architecture
ISO	International Organization for Standardization
IT	Indication Tester
LT	Lower Tester
MFC	Microsoft Foundation Classes
MO	Mobile-originated
MT	Mobile-terminated
.NET	Dot NET Framework

OLE	Object Linking and Embedding
OS	Operating System
OOC	A framework for implementing object oriented systems using
PCO	Point of Control and Observation
PDU	Protocol Data Unit
PC	Personal Computer
QML	Qt Meta Language or Qt Modelling Language
S40	Series 40
SMS	Short Message Service
SUT	System Under Testing
TTCN	Tree and Tabular Combined Notation (language)
UI	User Interface
USB	Universal Serial Bus
UT	Upper Tester
VoIP	Voice over IP
Win32	Windows API for developing 32-bit applications

1 INTRODUCTION

This Master's thesis is about the Implementation of the ISA TTCN automated test environment in a software laboratory, which is a topic of the Nokia S40 releasing project. ISA TTCN is one of the tools used by Nokia S40 in developing software testing applications for both the developer and the release testing teams. This helps the release testing teams to understand how the ISA TTCN automated test environment works and gives the capability of Automating testing based on the Lower Layers and System level S40 software quality with the test cases written in the C/C++ programming language. The Master's thesis equally seeks to facilitate the testing result reporting with a program written in the QML language using the Qt platform.

1.1 Objectives

The main objective of the Master's thesis is to improve the testing efficiency as well as to make testing more automatic in order to acquire a high software quality within S40 products. This is intended to be achieved by implementing and modifying the ISA TTCN test environment to support the end to end as well as the system level testing based on automated cases. The project has three main goals.

The first goal is to study the old manual testing ISA TTCN environment by either adding or creating the automated testing or by the modification of the ISA TTCN environment to support the lower layers and the end to end testing for S40 features. The second goal is to add a plug-in support to enable the UI end to end interaction and the simulation based on the system level testing.

The third goal is to provide a means by which the test case results may be reported after the automated test runs by using the QT (QML) application. This will go a long way to provide the S40 Management with a clear understanding of the test case results based on PASS or FAIL

Building an automated test environment for S40 features and release testing is a complex and difficult design process. It requires an accurate naming of the classes, definition of association type, definition of class attributes and operations and programming using the C++ language. The advantages of the project are to help the

S40 testing teams to understand the TTCN language structure and how the Microsoft Visual studio C++ and Qt 4.7 cooperate with each other. The disadvantage is that the ISA TTCN environment is new to the S40 testing teams and the testers have to consider the compatibility between ISA TTCN and other Microsoft Visual studio C++ Compiler.

1.2 Research problems and methods

This Master's thesis sets out to solve the problems outlined below in the descending order of severity with the respect to quality of S40 products.

1. Biggest problem: How to improve software quality within S40 products?
2. Big problem: How to improve testing of S40 software?
3. Small problem: How to increase the level of automated testing

The methods used to solve the above-mentioned problems are based on creating ISA TTCN automated test environment to save time used for testing manually by the S40 releasing teams. Manual testing is time consuming and provides less coverage at a particular time frame of the S40 features. Creating an automatic mode of testing with ISA TTCN for the S40 features releasing teams will go a long way to increase efficiency by giving them the opportunity to cover more work with less time, energy and effort consumed.

It was therefore eminent that for the three research problems to be solved, the creation of ISA TTCN automated test environment which enables end to end and system level testing to be automated with a wide range of test cases support based on key S40 features for the market was indispensable. The details of the methods used to solve these research questions will be covered in the subsequent chapters.

2 AUTOMATED SOFTWARE TESTING

Software test automation refers to the activities and efforts that are required to automate engineering tasks and operations in a software test process using well-defined strategies and systematic solutions. While automation cannot reproduce everything that a human can do (and all the ways they think of doing it), it can be very useful for regression testing. However, it does require a well-developed test suite of testing scripts in order to be truly useful.

2.1 Test environment

Since the successes and failure of testing methods and tools are highly context dependent, let me first of all give you a brief overview of the system under test. The system under test (SUT) is the ISA TTCN target test environment used for test development. In the target testing, the software is integrated and run in the real hardware. The ISA TTCN can be used to start testing the protocol software when the UI application is not available or to validate cases, which are difficult or impossible to implement in the real test environment. ISA_TTCN is Nokia In-house tool and developed by several organisations in the past. Figure 1 represents the system under test consisting of the Upper tester (UT) and the test system consisting of the ISA TTCN Lower Tester (LT).

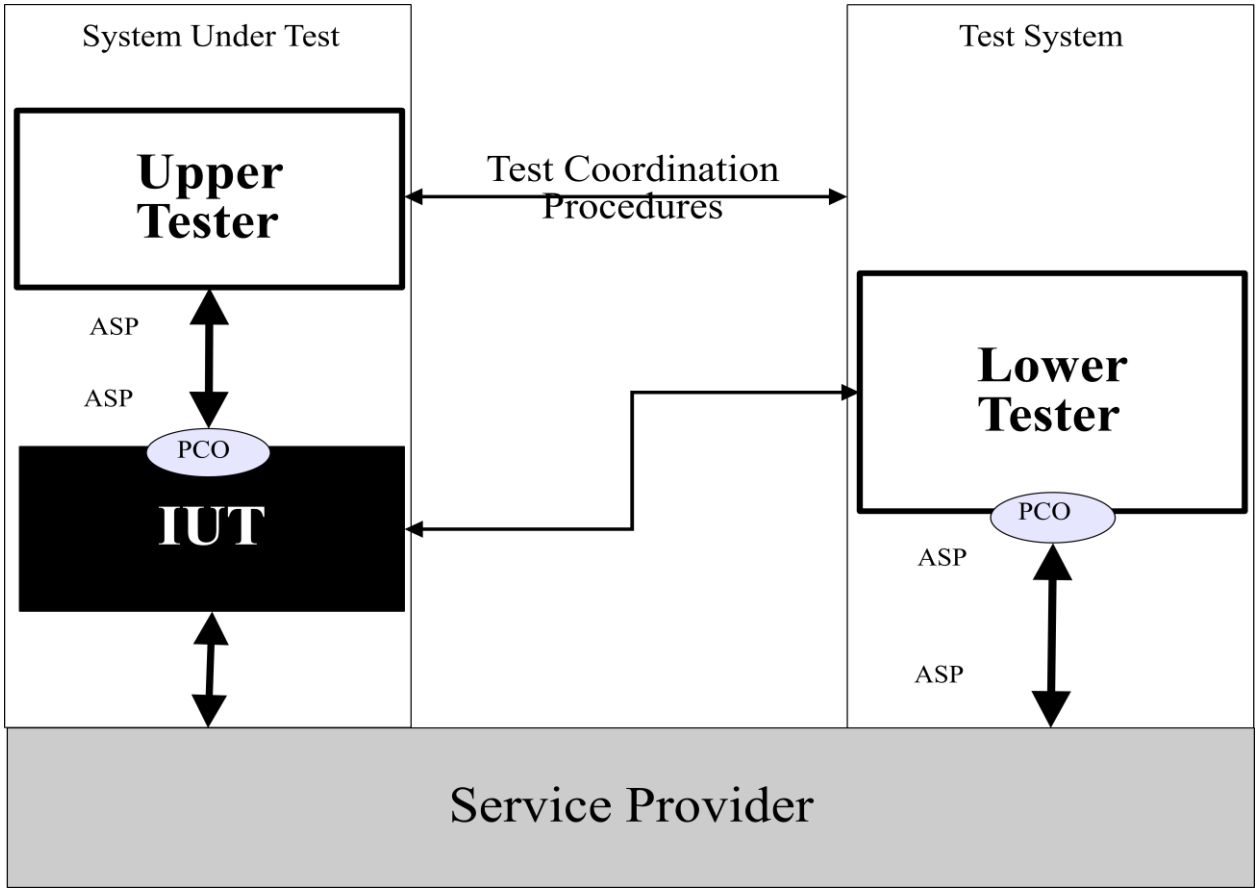

***FIGURE 1**. The conceptual view of ISA TTCN basic configurations*

In protocol testing, UT provides control and observation of the upper service boundary of the IUT. From the test system, the LT provides control and observation of the lower service boundary (ISO/IEC 9646-1). The ISA TTCN basic consist also of ASP(s) and PCO (s). A PCO is a point in the abstract interface where the IUT can be stimulated and its responses can be inspected. An ASP or PDU is either a stimuli or a response that carries information, parameter and data. Each PCO has two first in first out (FIFO) queues for temporary storage of ASPs and PDUs and uses one queue for sending and one for receiving.

2.2 General Concepts of Automated Testing

Testing is very important in order to ensure the quality and performance of any products provided by Nokia S40 software. Before the product is launched in the market, it undergoes several tests, which is mainly done to ensure the desired performance of the product. However, when it comes to the testing process, there are two options such as manual testing and automated testing programs. Automated testing is considered as the best, as it helps the users to save both time and money.

Manual testing is usually quite time consuming as well as error prone. Moreover, relying alone of the manual testing will increase the chances of late defect discovery and submissions. This is also one of the main reasons that lead to delays in the product release. On the other hand, Nokia S40 has been looking for testing solutions based on automated embedded testing to help the testing teams to complete their work quickly and more efficiently(Embedded testing, 2011).

Every software development group tests its products, yet delivered software always has defects. Test engineers strive to catch them before the product is released but they always creep in and they often reappear, even with the best manual testing processes. Automated software testing is the best way to increase the effectiveness, efficiency and coverage of your software testing.

2.3 Advantages of Automated Software Testing

An automated software testing tool is able to playback pre-recorded and predefined actions compare the results to the expected behaviour and report the success or failure of these manual tests to a test engineer. Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing. (SmartBear Software, 2011).

2.3.1 Automated Software Testing Saves Time and Money

Software tests have to be repeated often during development cycles to ensure quality. Every time source code is modified software tests should be repeated. For each release of the software, it may be tested on all supported operating systems and hardware configurations. Manually repeating these tests is costly and time consuming. Once created, automated tests can be run over and over again at no additional cost and they are much faster than manual tests. Automated software testing can reduce the time to run repetitive tests from days to hours. A time savings that translates directly into cost savings.

2.3.2 Automated Software Testing Improves Accuracy

Even the most conscientious tester makes mistakes during monotonous manual testing. Automated tests perform the same steps precisely every time they are executed and never forget to record detailed results.

2.3.3 Automated Software Testing Increases Test Coverage

Automated software testing can increase the depth and scope of tests to help improve software quality. Lengthy tests that are often avoided during manual testing can be run unattended. They can even be run on multiple computers with different configurations. Automated software testing can look inside an application and see memory contents, data tables, file contents, and internal program states to determine if the product is behaving as expected. Automated software tests can easily execute thousands of different complex test cases during every test run thereby providing coverage that is impossible with manual tests. Testers freed from repetitive manual tests have more time to create new automated software tests and deal with complex features.

2.3.4 Automated Software Testing Does What Manual Testing Cannot

Even the largest software departments cannot perform a controlled web application test with thousands of users. Automated testing can simulate tens, hundreds or thousands of virtual users interacting with network or web software and applications.

2.3.5 Automated Software Testing Helps Developers and Testers

Shared automated tests can be used by developers to catch problems quickly before sending to Software Quality Assurance. Tests can run automatically whenever source code changes are checked in and notify the team or the developer if they fail. Features like these save developers time and increase their confidence.

2.3.6 Automated Software Testing Improves Team Morale

Improving team morale is difficult to measure but we've experienced it first hand, automated software testing can improve team morale. Automating repetitive tasks

with automated software testing gives your team time to spend on more challenging and rewarding projects. Team members improve their skill sets and confidence and, in turn, pass those gains on to their organization.

2.4 Manual Software Testing

Manual software testing is performed by a human sitting in front of a computer and carefully going through application screens, trying various usages and input combinations, comparing the results to the expected behaviour and recording their observations. Manual tests are repeated often during development cycles for source code changes and other situations like multiple operating environments and hardware configurations (Embedded testing, 2011).

3 INTRODUCTION TO TTCN TOOLS

TTCN (Tree and Tabular Combined Notation, ISO/IEC 9646-3) is a language standardized by ISO for the specification of tests for real-time and communicating systems. TTCN has been developed within the framework of standardized conformance testing (ISO/IEC9646).

TTCN now referred to as the Testing and Test Control Notation (version 3) was previously referred to as Tree and Tabular Combined Notation (version 1 & 2). TTCN was developed with the purpose of conformance testing. i.e. to verify that a given system conforms to the standards/specifications. TTCN versions 1 & 2 were developed by ISO (International Standards Organization). TTCN version 3, the current version, was developed by ETSI (European Telecommunications Standard Institute) and was standardized in 2002. The purpose was to decouple the notation from conformance testing and have a wider application (TeleLogic, 2010).

3.1 TTCN Test Language

With TTCN a test suite is specified. A test suite is a collection of various test cases together with all the declarations and components they need. Each test case is described as an event tree. In this tree, behaviours such as "First, we send A, then either B or C is received; if it was B we will send D..." are described. Concurrent TTCN allows several event trees to run concurrently.

TTCN is abstract in the sense that it is test system independent. This means that a test suite in TTCN for one application (e.g. protocol, system, etc.) can be used in any test environment for that application.

The use of TTCN has increased tremendously during the last few years. This has been augmented by the significant amount of test suites released by various standardization bodies. TTCN is not only used in standardization work. The language is very suitable for all kinds of functional testing for real-time and communicating systems. This has led to a wide usage throughout the industry (TeleLogic, 2010).

3.1.1 Theoretical Model

A TTCN specification describes an abstract test suite (ATS) that is independent of test system, hardware and software. The ATS defines the test of the implementation under test (IUT), which is treated in a black box model, i.e. only its exterior interface is of concern. The IUT is stimulated by sequences of test events and its response is inspected.

A TTCN abstract test suite can be transformed into an executable test suite (ETS) using the TTCN suite. This ETS (Figure 2) is downloaded into the test system (the system performing the test).

The test system performs the test by executing the ETS against the system under test (SUT) which contains the implementation under test. During execution the ETS reports any errors and log events for on-line or post-test evaluation (TeleLogic, 2010).

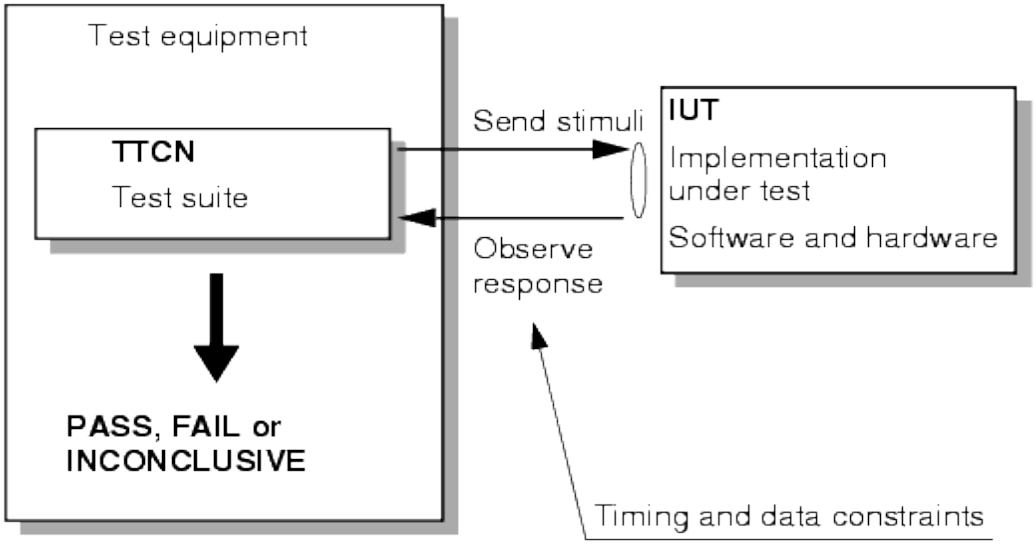

***FIGURE 2**. Test System with Executable Test Suite (ETS) connected to the system under test (SUT) (TeleLogic, 2010)*

3.1.2 TTCN Specification Structure

A TTCN specification is similar to a Pascal or C program. (Being cleaner and more comprehensive, TTCN is easier to learn.) Just like Pascal and C, TTCN requires type and data declarations and it uses concepts like modules and subroutines. Since TTCN is designed for testing, it contains test specific concepts such as:

- Powerful pattern matching constructs for complex data structures using both TTCN and ASN.1.
- Verdicts and preliminary verdicts to define the outcome of test cases.
- Possibilities to handle alternative outcomes in a test case.
- Pre-ambles and post-ambles to show how to compose test cases.
- The "modules" concept supporting multi-user test development.
- The "modules" concept supporting the re-use of test components and data structures.
- Constructs for parallel test component execution including the synchronization of primitives.

A TTCN specification has a standardized layout that produces comprehensive and unambiguous paper printouts. This greatly improves clarity and readability. A test suite is divided into the following four major parts:

The overview part contains a table of contents and a description of the test suite. Its purpose is mainly to document the test suite so as to increase clarity and readability.

The declarations part declares all messages, variables, timers, data structures and black box interface towards the Implementation under Test.

The constraints part assigns values and creates constraints for inspection of responses from the implementation under test.

The dynamic part, contains all test cases, test steps and default tables with test events and verdicts, i.e. it describes the actual execution behaviour of the test suite (TeleLogic, 2010).

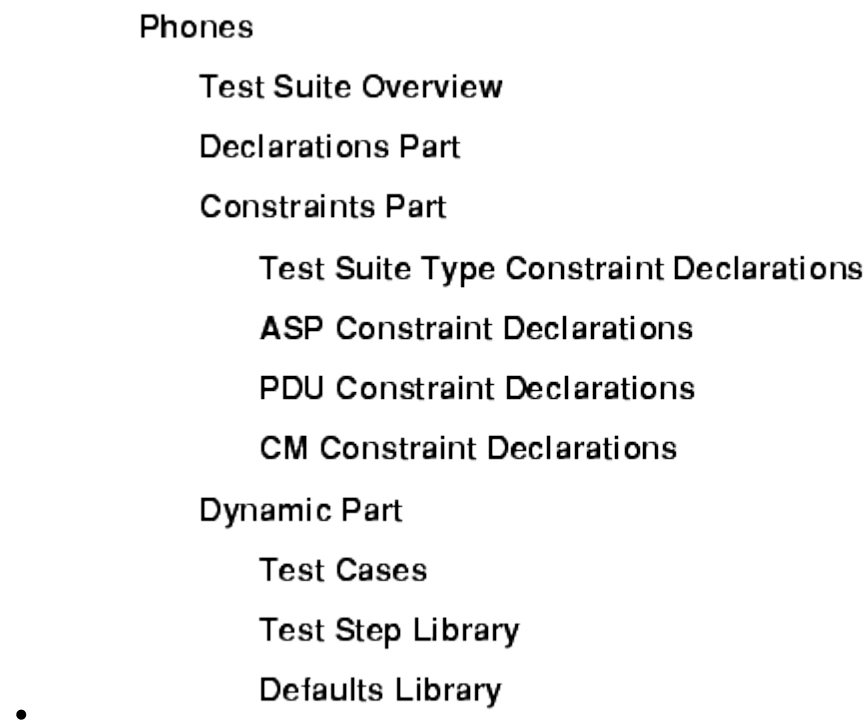

***FIGURE 3**. The basic structure of a TTCN (TeleLogic, 2010)*

3.2 ISA TTCN Tester

The ISA TTCN tester is designed for testing any kind of ISA server in workstation (PC or UNIX) or in target build. It uses a "TTCN like" testing language to describe the test cases and the test environment. All test scripts are plain C-language and no code generation is needed. The ISA TTCN tester is also one ISA server. This is why it can be integrated to any product build easily (Figure 4).

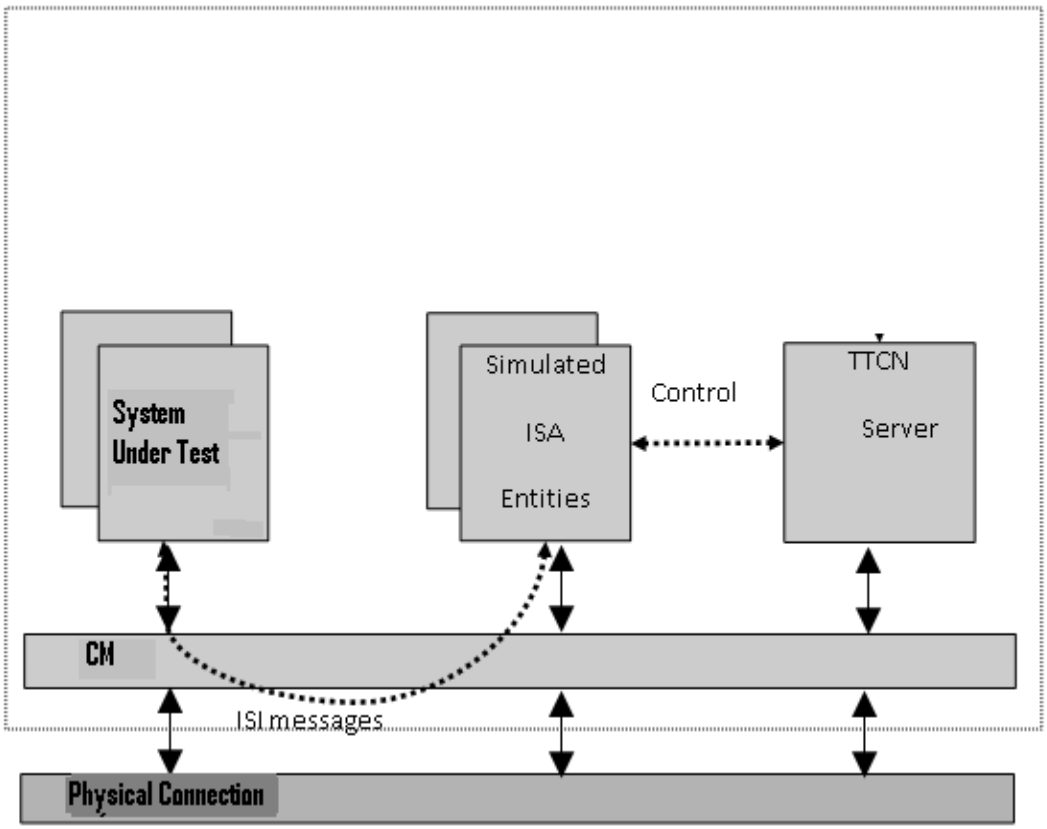

***FIGURE 4**. ISA TTCN in workstation test environment*

The basic idea is that ISA TTCN test server registers itself to CM as other ISA servers. It does it for all those servers that are simulated (Figure 4). The tested ISA server won't see any difference in the communication to real server. ISA TTCN test server makes the communication look real by simulating the behavior of real servers. The ISA TTCN tester makes extensive use of the C-language "#define"'s. In this way the test cases look like something else than plain C-language, but still have all the capabilities of normal C-language.

In ISA TTCN target test environment ISA TTCN test tool is used to test ISA servers in the real phone. Communication between PC and a phone is done with MTI32 library that connects a PC to the phone (Figure 5). The functionality of the ISA TTCN tester in this environment is nearly the same as original ISA TTCN tester.

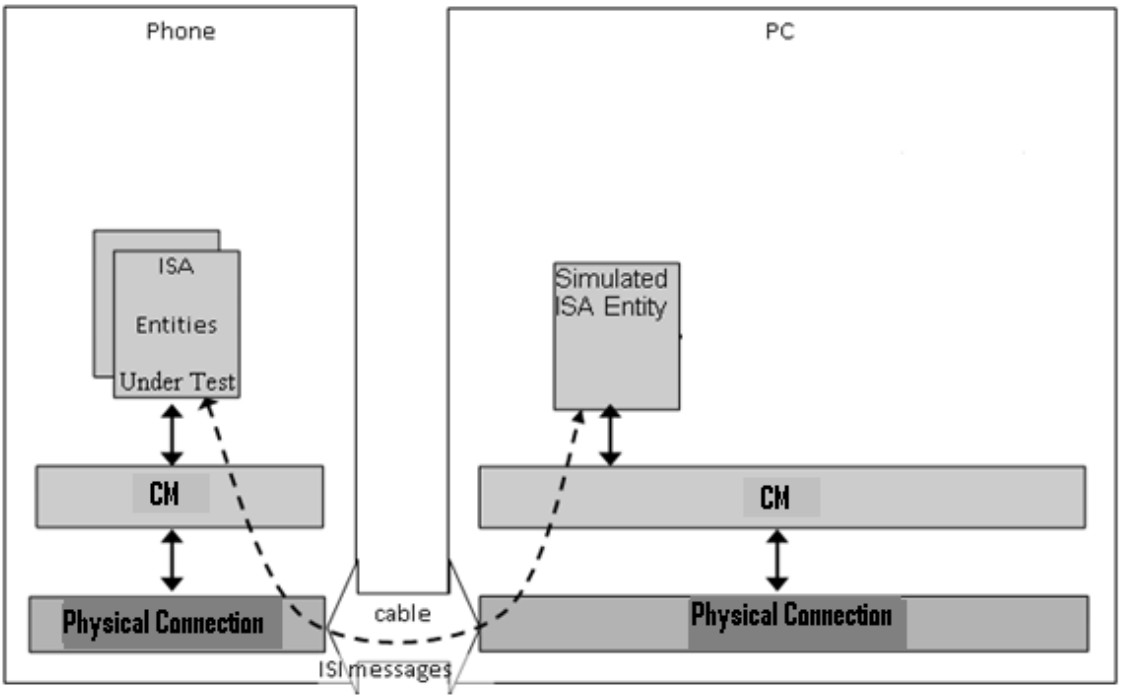

FIGURE 5. *ISA TTCN target testing environment*

Modifications are done to the environment where the testing is executed. Original ISA-TTCN tester is built on a top of CM and the Physical connection Layer (Figure 5). All these components are replaced by very simple ones in ISA TTCN target test environment. There are also only few header files from the Physical Connection Layer needed because the functionality of the Physical Connection is partially handled by MTI32 library.

4 USED TOOLS

A programming tool or software development tool is a program or application that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object (Wikipedia, 2012).

4.1 Microsoft Visual C++

The testing Tool is built using MS Visual C++. Visual C++ 6.0 is a product licensed by the Microsoft Corporation. Visual C++ is available in three editions Standard, Professional, and Enterprise. The Visual C++ professional Edition is used to develop applications, services, and controls for Win32 platforms, including Windows NT. It can target the operating system's graphical user interface or console APIs and also provides features to develop and distribute commercial-quality software products. When using Visual C++ 6.0, it provides many features Such as Compiler, Debugger, Editors, Linker, Automation Object Model, Projects, Wizards, OLE DB Templates, MFC, Database Support and Sample Programs.

In the design of ISA TTCN, the main functions used in Microsoft Visual C++ 6.0 were the Compiler and Editor. Figure 6 represents the Microsoft Visual C++ 6.0 User Interface that happens to be easier to use.

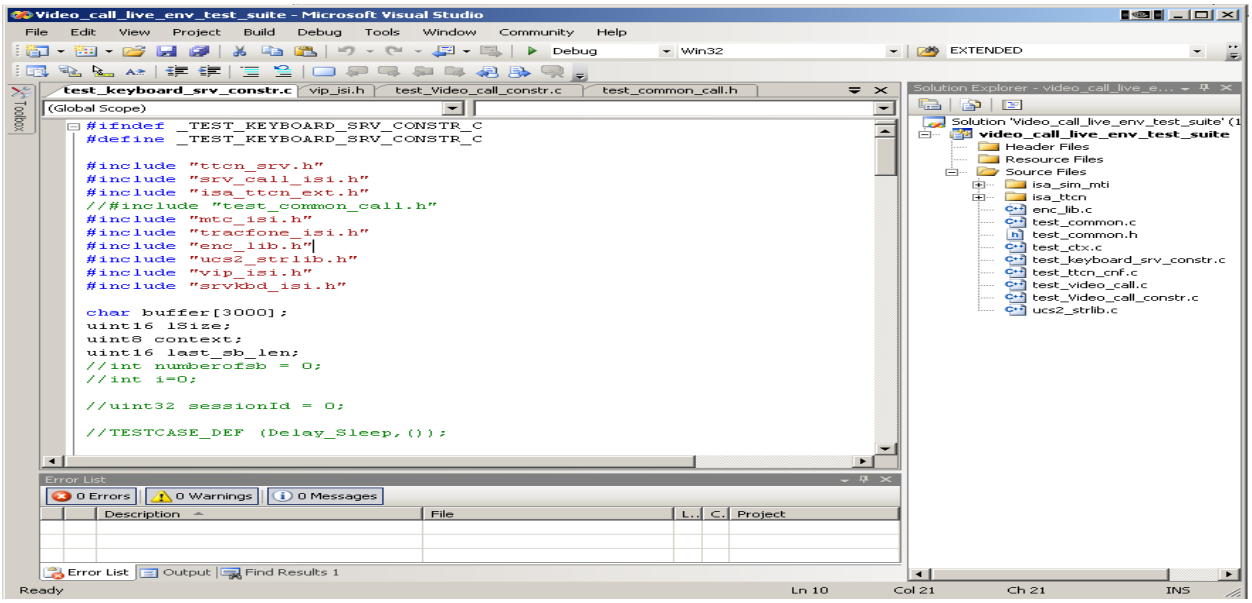

FIGURE 6. *Microsoft Visual C++ 6.0 User Interface*

4.2 MTI32 library

Before one can try to run the tester one has to have MTI32 runtime parts installed to your PC. If one has Phoenix installed then one should have all needed files because Phoenix uses MTI32. If someone does not have Phoenix, then one has to install MTI32.

4.3 Data cable

Data cable (e.g. DKU-2, DKU-5 or DAU-9T) is needed for connecting a phone to the PC. It is possible to use IR or Bluetooth connections but they are never been tried.

4.4 Tracing box

The trace box is a bundle of hardware and software modules serving as a tracing interface between traced device(s) and user application. The trace boxes act as physical interfaces between traced device(s) and PC. Data received from traced device(s) is pre-processed in boxes and forwarded via USB connection to a tracing software module running on PC. The software module converts data into format acceptable for user application software.

4.5 Qt version 4.7

Qt is produced by Nokia's Qt Development Framework division, which started after Nokia acquired a Norwegian company Trolltech, the original producer of Qt. It includes a cross-platform class library, integrated development tools and a cross-platform IDE. Using Qt, you can write applications once and deploy them across many desktops, smartphones and embedded operating systems without rewriting the source code.

Qt uses standard C++ but it also makes use of a special code generator called
Meta Object Compiler together with several macros to enrich the language. Qt can also be used together with other programming languages through a language binding. It runs on all major platforms and has extensive internationalization support (Wikipedia. 2011). Below there is a sample of Qt Hello world application:

```
#include <QtGui>
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
QLabel label("Hello, world!");
label.show();
return app.exec();
}
```

The Qt Creator is a cross-platform integrated development environment (IDE) tailored to the needs of Qt developers. Qt Creator runs on Windows, Linux/X11 and Mac OS X desktop operating systems, and allows developers to create applications for multiple desktop and mobile device platforms. Qt Creator provides two integrated visual editors: *Qt Designer* for building UIs from Qt widgets and *Qt Quick Designer* for developing animated UIs with the QML language (Nokia, 2011).

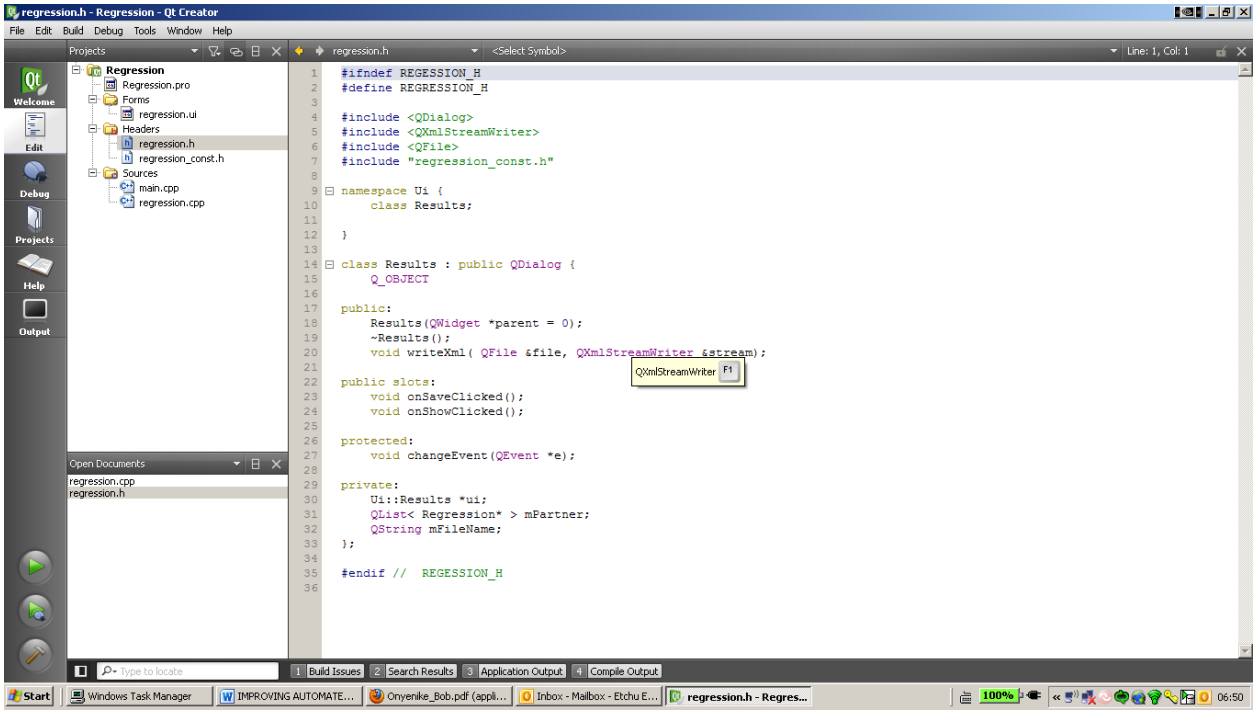

FIGURE 7. *Qt Creator used for the Regression project*

5 IMPLEMENTATION OF ISA TTCN AUTOMATION AND PLUG-IN

The new automation concepts of ISA TTCN features live test environment was done for testing S40 features functionality automatic. The main target groups were development teams dealing with signalling level issues for S40 features testing and releasing teams.

The environment consists of two parts; GUI and exe file. GUI has been done by using Windows API and thus it works only on top of Windows operating system. It mainly only runs exe files in command prompt window as different processes. The Exe file runs test cases, which has been done by using ISA TTCN Target Test environment. Test cases verify that ISI message signaling sequence with mandatory consistent are correct but it cannot verify UI behavior or that audio works.

The goals of this project were to facilitate S40 features and release testing scope to cover a wide range of test cases based on S40 software architecture as well as effective methods of testing S40 software releases.

5.1 Modifying ISA TTCN to support end to end testing for lower layers

The modification of the ISA TTCN environment to support end to end testing for lower layers was indispensable for the automation of test cases for S40 feature testing and releasing teams. As mentioned in Section 3.2.1, the original ISA TTCN environment made it possible to use only one mobile phone for testing. Phone A was connected to a PC running the ISA TTCN environment via USB in which the ISA TTCN environment was acting as the remote terminal. Nevertheless, it was imperative for us to add the support for a second phone named Phone B as illustrated in Figure 8. Adding phone B came along with its own challenges such as creating two USB connections to the ISA TTCN Interface, and modifying the ISA TTCN simulator (Simulated ISA Entity) to have a support for Phone B.

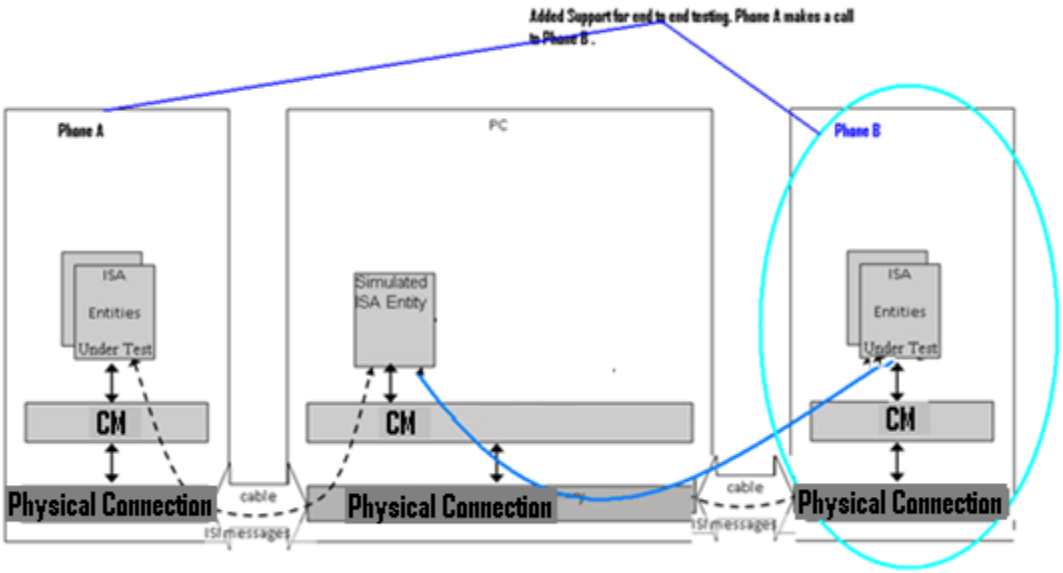

FIGURE 8. *ISA TTCN target testing environment with end to end testing support*

5.1.1 Connecting a Phone to the PC

With the original ISA TTCN environment you have to connect a phone to the PC. The user needed a data cable like DKU-2 or DAU-9T. Depending on port and cable, one has to specify the used connection at the beginning of the ttcn_mti.cpp file. The following is an example of the definition of a connection.

#define MTI_CONNECTION_TYPE "BUS_NAME:USB"

The exact format of that connection initialisation string is specified below. If one changes the port, one have to re-build the tester. After that, connect phone to the PC, power it up and try to run the test tool.

Connection initialisation string format:

“KEY_1:VALUE KEY_2:VALUE KEY_N:VALUE”

The keyword and its value are separated with a colon (‘:’) character. The keyword/value pairs are separated with a space (‘ ‘) character.

5.1.2 Connecting Two Phones to the PC

For the new ISA TTCN environment to Support two phones connected to a PC, Software modifications were done first at the level of defining a second USB connection to support a second phone to be connected. As well as depending on ports

and cables, it was specified to use connection at the beginning of the ttcn_mti.cpp file. For example:

#define MTI_CONNECTION_TYPE "BUS_NAME:USB"

#define MTI_CONNECTION_TYPE "BUS_NAME_2:USB_2"

Changes were also made in Connection initialisation string format as follows

“KEY_1:VALUE KEY_2:VALUE KEY_N:VALUE”

“KEY_2:VALUE KEY_4:VALUE KEY_N_2:VALUE”

5.1.3 Modification of ISA TTCN Simulator

Even after modifying the USB connections, it was still not possible to perform an end to end testing with the two phones. In that case, it was of utmost necessity to modify the simulated ISA Entity as shown in Figure 8. The Simulated ISA Entity was also split into two so as to simulate the two phones namely Phone A and Phone B.

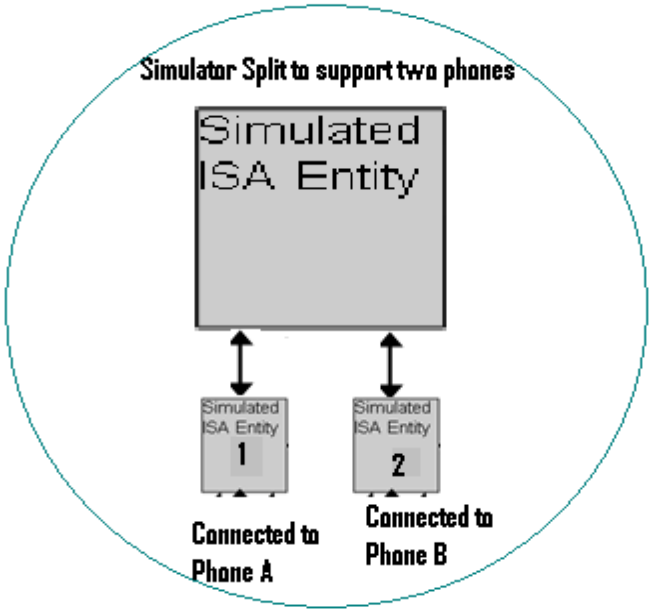

FIGURE 9. *ISA TTCN Simulated ISA Entity Split*

As shown in Figure 9, the main Simulated ISA Entity, through code base, was split into Simulated ISA Entity 1 and Simulated ISA Entity 2. Simulated Entity 1 was created to handle all ISI messages simulations from Phone A to the main Simulated ISA Entity. The

same process was carried out between the Simulated ISA Entity 2 and Phone B. In the code or software implementation, this process mentioned above was really complex and not so straight forward. The Simulated ISA Entity simulates basically S40 features components such as Calls, SMS Messaging and VoIP. Based on Figure 9, for us to better understand the modifications made on the main Simulated ISA Entity, let us consider the call functionality as an example.

When a call is created by Phone A, the Simulated ISA Entity 1 handles the call as MO call and passes it to the main Simulated ISA Entity which in turn processes the call and forwards it to the Simulated ISA Entity 2 as MT call. The Simulated ISA Entity 2 becomes responsible for communicating with Phone B and making decisions on answering the call. That completes a full sequence on the automated test environment simulating an end to end testing.

5.1.4 End to end testing

End to end testing is the methodology to validate whether the flow of application from the starting point to end point is happening perfectly (Test Republic, 2011). An example from Figure 8, while testing MO Call, the start point was from when the Call was created by Phone A, Phone B receives the Call as MT, Call was answered by Phone B and the end point was when the Call got released by Phone A. During the Call between Phone A and Phone B, the Simulated ISA Entity verifies that ISI message signaling sequence with mandatory consistent is correct but it cannot verify UI behavior or that audio works.

The S40 features testing and release teams have been making good use of the ISA TTCN automation support modification as illustrated in Figure 8 to run Call, SMS Messages, VoIP and Videos.

5.2 Plug-in support for System level testing with ISA TTCN

The System level testing is being done mostly by the S40 features testing and releasing teams. The test cases were run manually on weekly basis in an attempt to increase the S40 regression coverage. With the lack of an automated environment, the testing teams found it very difficult to get good quality by running test cases manually. System level testing checks that the software functions properly from end-to-end. The

components of the system include: Databases, Calls, media servers to stream audio and video, and messaging services.

After the realisation that it was possible to carry out end to end testing as a results of the modification done on the ISA TTCN environment to automate two mobile phones (section 5.1), it was obvious that simulating the UI components from the mobile phones will provide the S40 features testing and release teams with a great system level tool. The ground work to support the UI simulation was based on Figure 8 and Figure 9. With the phone consisting of newly integrated test servers and a UI Components simulator support on the ISA TTCN environment as illustrated in Figure 10, gave an easy solution and effective approach of simulating the lower layers combined with the UI end to end testing capabilities.

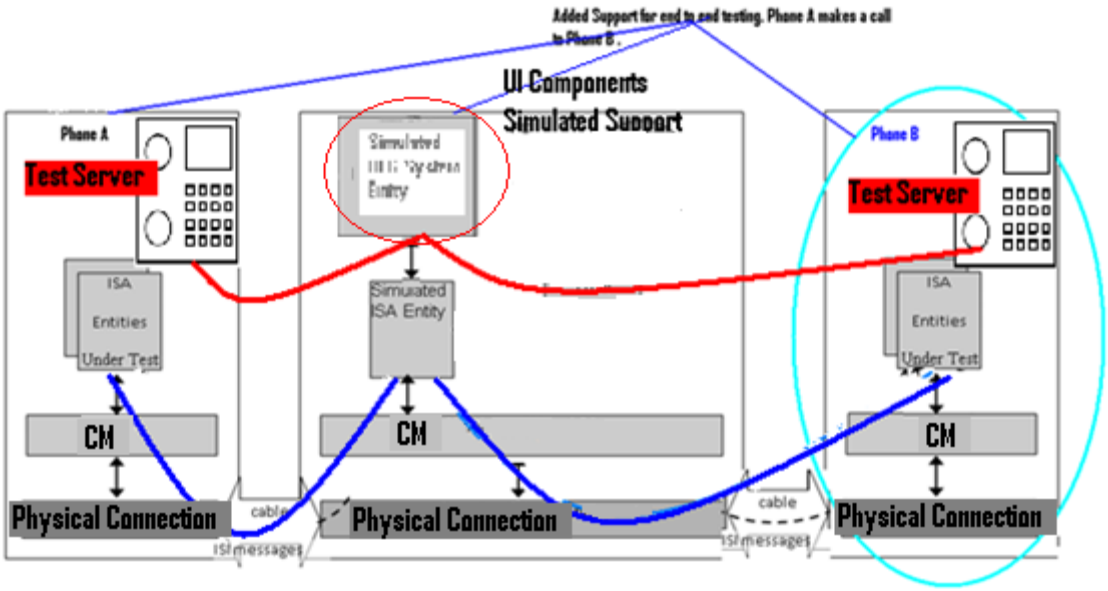

FIGURE 10. *ISA TTCN target testing environment with end to end testing support*

5.2.1 Creation of ISA TTCN UI Simulation

Automating the testing of lower layers and the UI has always been a huge challenge for Nokia S40 projects. More so, the ISA TTCN environment provided a good platform for the implementation of an end to end lower layer as well as the UI testing support .The main task for the creation of the ISA TTCN UI automation support as shown in Figure 10 was to implement a completely new component named Simulated UI ISA Entity.

The Simulated UI ISA Entity consists of components such as Key press, Menu, and Panel.

The Simulated UI ISA Entity consists of a simple C/C++ code implementation that basically responds to key presses signals from either Phone A or Phone B

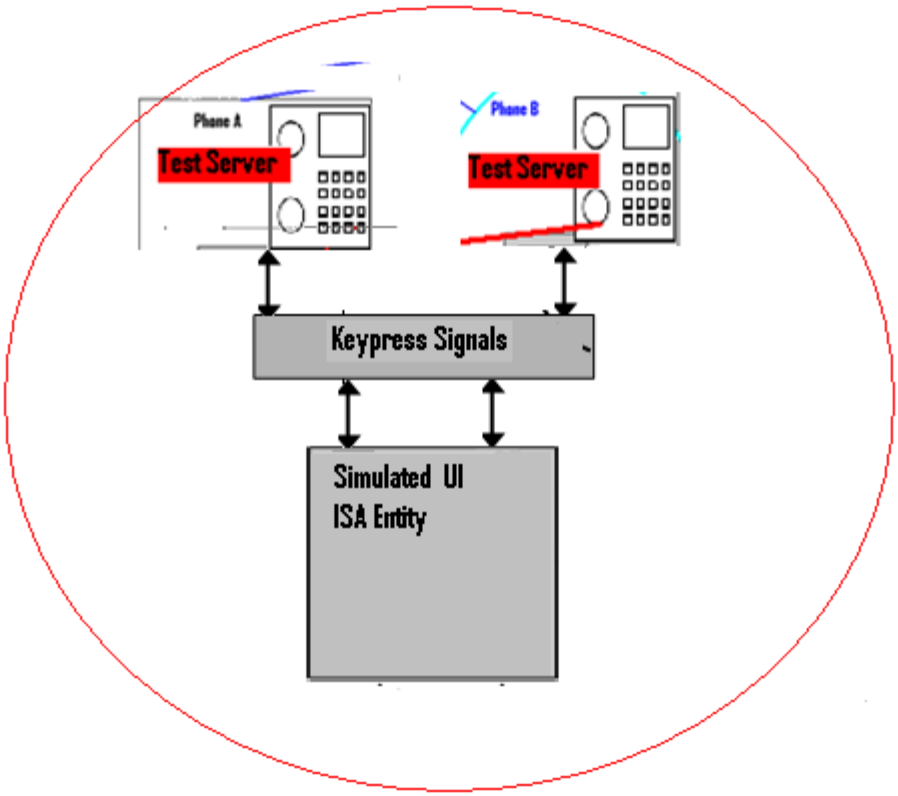

FIGURE 11. *ISA TTCN Simulated UI ISA Entity*

The creation of the ISA TTCN simulated UI ISA Entity Plug-in to the ISA TTCN automated environment as shown in Figure 11 works basically on the user key press theory. The implementation made it possible to automate the key press requests and responses coming from the Test Server from both phones. Each time a key press request was sent by a test case, the ISA TTCN Simulated UI ISA Entity replied with a response to the Test Servers. One good example was by creating a call from Phone A to Phone B. The test case to create a call starts by dialling the phone number that is visible in either Phone A or Phone B UI. Once the numbers needed to place the call were ready, the test case later sent key press request to simulate the UI as the green button pressed (call button). Hence, this support facilitates how the system level testing was carried out.

5.2.2 System Level Testing

System Testing is the methodology to validate whether the system as a whole is performing as per the requirements (Test Republic, 2011). System level testing consists of many modules or units in an application and in the system testing and one has to validate that the application performs as per the requirements. To be precise, in end to end testing, the flow of activities from scratch to end is validated. Meanwhile in the system testing, the system as a whole is validated.

The Plug-in for the system level testing came as a great innovation for Nokia S40 feature testing and releasing teams due to the fact that, the ISA TTCN automation tool with the UI simulation support facilitated end to end UI testing. With great accuracy, system level testing was made more efficient and reliable for Nokia S40 software quality analysis. Having an accurate UI simulation makes it possible to simulate an event where by the end user utilises the phone. One good example was how Phone A made a call from the UI to Phone B. This UI interaction was simulated and automated by ISA TTCN automated environment end to end testing support as well as the Plug-in.

The phone call initiation sequence starts by the Plug-in dialing Phone B's phone number using the Simulated UI ISA Entity by sending key presses of Phone B's phone number to Phone A. When Phone A's Test Server receives the right phone number, it displays the number to Phone A's UI. At this point, Phone A's Test Server sends a ready indication to the Simulated UI ISA Entity and waits for the next key press command. Finally, the Simulated UI ISA Entity sends the final key press signal request to press the call button.

When the call on Phone B starts ringing, its Test Server sends an indication to the Simulated UI ISA Entity informing the Simulated UI ISA Entity that it has a call alerting and the Simulated UI ISA Entity in turn makes a decision to send the answer key press signals to Phone B. This completes the full sequence of system level and end to end testing for a basic call feature on Nokia S40 products with ISA TTCN automated environment.

5.3 Creating test suites for ISA TTCN test cases

ISA TTCN uses a "TTCN-like" notation, meaning that the notation is not compatible with real TTCN, but the notation looks very alike with real TTCN. ISA TTCN notation is plain C code using macros and supports testing of ISA entities of any kind as well as testing ISA function interfaces. The ISA TTCN contains macros to define test cases, simulate ISA entities around IUT and to run them all in either workstation or target ISA environment. Just like TTCN, the dynamic part of an ISA TTCN abstract test suite is created in a hierarchical and nested manner. The building blocks are test groups, test cases, test steps and test events. There are no limitations as to how many test groups may be contained in a test suite or how many test events may be contained in a test step, etc. (TeleLogic, 2010).

Test component explanation:

- ***Test event:*** The smallest, indivisible unit of a test suite. Typically, it corresponds to a signal, interrupt, message, data or timer expiration.
- ***Test step:*** A grouping of test events, similar to a subroutine or procedure in other programming languages.
- ***Test case:*** The main fundamental building block in a test suite. A test case tests a particular feature or function in the implementation under test (IUT). A test case has an identified test purpose and it assigns a verdict that depends on the outcome of the test case.
- ***Test group:*** A grouping of test cases. It might for example be convenient to group all test cases concerning connection establishment, and to put all test cases concerning transport into a separate test group.
- ***Test suite dynamic part:*** The highest level, encompassing all test components and serving as the root of the tree (Figure 12*)*. A test suite can range from a large number of test groups and test cases to a single test event contained in a test case. The dynamic part structure of ISA TTCN is same as that of TTCN as shown in Figure 12 below

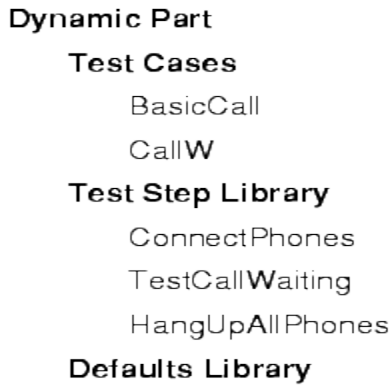

***FIGURE 12**. The TTCN dynamic part structure (TeleLogic, 2010)*

5.3.1 Communication Mechanisms

ISA TTCN uses the concepts of points of control and observation (PCOs), abstract service primitives (ASPs) and protocol data units (PDUs) in order to create an abstract interface towards the implementation under test (IUT). A PCO is a point in the abstract interface where the IUT can be stimulated and its responses can be inspected. An ASP or a PDU is either a stimuli or a response that carries information, i.e. parameters and data *(TeleLogic, 2010)*.

Each PCO has two FIFO queues for temporary storage of ASPs and PDUs. It is Possible to send and receive ASPs through all possible PCOs in the same test case. All interfaces from IUT to external ISA entities are defined as PCO. Every entity test case communicates (= send or receive messages) with needs to have its own unique PCO. Even subscriptions to ISI events are done via PCO definitions. PCOs are defined with PCO macro, between PCO_DECLARATIONS_BEGIN and PCO_DECLARATIONS_END macros in test script.

The ISI message sending varies a little due to this (there are own CM functions to send ISI messages to servers and to other ISA entity types). IT PCO type means that if one only needs to get certain indications from some simulated entity one can define its PCO as IT. Then indications can be received from IT PCO and also respond to them, but nothing else. PCO type DT is meant for ISA delegate interface testing and it is defined with its own special macro.

```
PCO_DECLARATIONS_BEGIN
      PCO(  <name>,<UT|LT|IT>,"<description>",<Phonet  resource>,
      <simulated ISA entity> )
      PCO_DELEG_TARGET( <name>, DT, "<comment>", <delegate> )
      :
PCO_DECLARATIONS_END
```

5.3.2 Constraints

Constraints are used in checking the internal data of the received ISI message and setting the correct test data to send ISI messages. Same constraint can be used both for sending and receiving. Constraints are used to set and compare test data in test cases. There are four different types of constraints that can be used for different purposes. The message parameters can be set or compared also in test step. However, if one needs to compare and set the same values in several test cases or test steps or if there is a great deal of parameters to be set or compared, a constraint is better a choice.

Constraints are like functions, so one can do data assignments, checks, comparisons and calculations inside them, like in C functions. Perhaps the most common use for ISA TTCN constraints is to set or compare message or function parameters using EQ macros. ASP constraints are used for signal values, for example, in the following way.

CONST_ASP_BEGIN(<name>,([<params>]),<aspname>)//ASP constraint Defination.
EQ...
CONST_ASP_END

STR constraints are used for any C structure values that are present in ASPs.

CONST_STR_BEGIN(<name>,([<params>]),<strname>)//STR constraint defination.
EQ...
CONST_STR_END

PDU constraints are used for PDU data (memory blocks) in ASPs. To use PDU constraints, there needs to be Pack and Unpack functions available for them.

CONST_PDU_BEGIN(<name>,([<params>]),<pduname>)//PDU Constraint Defination
EQ...
CONST_PDU_END

5.3.3 Creating Test Cases

Two steps can be seen in a test case creation:

Step 1, Create constraints for ISI messages: These constrains can often be copied from the ISA TTCN automated test environment or defined as mentioned in Section 5.3.2. Usually, tester should create constraints for sub blocks as well because it makes it much easier to add new test cases after that.

Step 2, Create test cases: A test case defines the test features. There can be several separate test cases in a test system. Group and purpose are optional, comment-like macros. The test cases dynamic behaviors (Figure 12) are defined between DYNBEH_BEGIN and DYNBEH_END macros. The Structure of a test case is the following:

1. TESTCASE_BEGIN(Name, Ops)
2. GROUP(group)
3. PURPOSE(purpose)
4. DYNBEH_BEGIN
5. DYNBEH_END
6. TESTCASE_END

Test cases from the ISA TTCN automated test environment can be used as templates for new test cases. A "Hello World!" ISA Server (PN_HELLO) Software code from the ISA TTCN automated environment and test case will consist of two parts namely; the Hello world server receiving and sending messages. The Hello World servers receives messages by a request sent from ISA TTCN Simulated ISA Entity and later sends the message back in a form of a response as shown below.

Hello World server receives these messages:

HELLO_INIT_REQ

HELLO_PRINT_REQ

Hello World server sends these messages:

HELLO_INIT_RESP

HELLO_PRINT_RESP

The definition of the test case that does the requested and response actions of the Hello World server looks like the code below, from the ISA TTCN automated environment.

```
TESTCASE_BEGIN (Poll,())
  DYNBEH_BEGIN
    TESTSTEP   (1,1, H,!,HELLO_POLL_REQ, pollReq(), NONE,"")
    TESTSTEP   (2,2, D,?,DUMMY_POLL_REQ, pollReq(), NONE,"")
    TESTSTEP   (3,3, D,!,DUMMY_POLL_RESP,pollResp(),NONE,"")
    TESTSTEP   (4,4, H,?,HELLO_POLL_RESP,pollResp(),NONE,"")
  DYNBEH_END
TESTCASE_END
```

5.4 Concept of Result reporting with QT (QML) application

Result reporting is a mechanism for presenting to the customer, from different angles, the state of the product (Indira, 2001). It is not enough to run tests and get PASS and FAIL as the end results but it is also important to have a nice way to report these results. In the Implementation of the results reporting concept, the testers were looking at some of the ways ISA TTCN results at the end of each regression testing runs could be displayed in a nice way than pure text report.

Regression Testing is done to verify the system after the changes made have not caused any unwanted side-effects. System Component Integration Testing is done to verify the integration of the system components, all of these testing types need to be reported bi-weekly or monthly by the S40 features testing and releasing teams working for the Nokia S40 projects.

5.4.1 Design and Results

The Design and implementation was done with QML language with the powerful Qt tool. Qt provides two new classes for reading and writing that were used in designing

the application. During the design and implementation, a QML project named Regression.pro was created by using the Qt tool.

The QXmlStreamReader and QXmlStreamWriter are two new classes provided in Qt 4.7 used in the Regression.pro project. A stream reader reports an XML document as a stream of tokens. This pulling approach made it possible to build recursive descent parsers, allowing XML parsing code to be split into different methods or classes.

QXmlStreamReader is a well-formed XML 1.0 parser that excludes external parsed entities. Hence, data provided by the stream reader adheres to the W3C's criteria for well-formed XML, as long as no error occurs. Otherwise, functions such as atEnd(), error() and hasError()can be used to check and view the errors (Nokia Corporation, 2008-2011).

Below is an example of QXmlStreamReader implementation used in the Regression.pro project:

```
Results::Results(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Results)
{
    ui->setupUi(this);
    mFileName = QFileDialog::getSaveFileName();
    //mFileName = QString( "someName.xml" );
    QString filename( mFileName );
    QFile file( filename );
    QXmlStreamReader stream( &file );
    if( !file.open( QIODevice::ReadWrite ) )
     {
        Q_ASSERT_X( 0, "ReadWrite", "failed to open xml for writing" );
        return;
     }
```

The QXmlStreamWriter class provides an XML writer with a simple streaming API. QXmlStreamWriter is the counterpart to QXmlStreamReader for writing XML. Like its related class, it operates on a QIODevice specified with setDevice(). The API is simple and straightforward. For every XML token or event you want to write, the writer provides a specialized function (Nokia Corporation, 2008-2011). The following abridged code from the regression.pro project shows the basic use of the class to write formatted XML with indentation:

```
void Results::onSaveClicked()
  {
  QString filename( mFileName );
    QFile file( filename );
    if( !file.open( QIODevice::WriteOnly ) )
      {
        Q_ASSERT_X( 0, "onSaveClicked", "failed to open xml for writing" );
        return;
      }
    mPartner.append( new Regression( ui->lineEdit->text(),
                        ui->lineEdit_2->text(),
                        ui->lineEdit_3->text(),
                        ui->lineEdit_4->text(),
                        ui->lineEdit_5->text() ));

    QXmlStreamWriter strm( &file );
    writeXml( file, strm );
  }
```

5.4.2 Inputting data form and creating style sheet

Qt Style Sheet is a powerful mechanism that allows someone to customize the appearance of widgets, in addition to what is already possible by sub-classing QStyle. The concepts, terminology, and syntax of Qt Style Sheets are heavily inspired by HTML Cascading Style Sheets (CSS) but adapted to the world of widgets (Nokia Corporation, 2008-2011). The following code lines from the Regression project creates a Style sheet named styleResults.xsl that specified widget, Colors, and how the text is displayed to the user.

```
void Results::writeXml( QFile &file, QXmlStreamWriter &stream)
{

    stream.setAutoFormatting( true );
    stream.writeStartDocument();
    QString xslt( "xml-stylesheet type=\"text/xsl\" href=\"styleResults.xsl\"" );
    stream.writeProcessingInstruction( xslt );
    stream.writeStartElement( "results" );
```

The environment consists of a GUI and exe file. GUI has been done by using Qt UI forms creator and thus works only on top of Windows operating system. It mainly runs the exe files in command prompt window as a different process. Exe file runs and provides the user form as shown in Figure 13, which has been done by using ISA_TTCN Target Test environment.

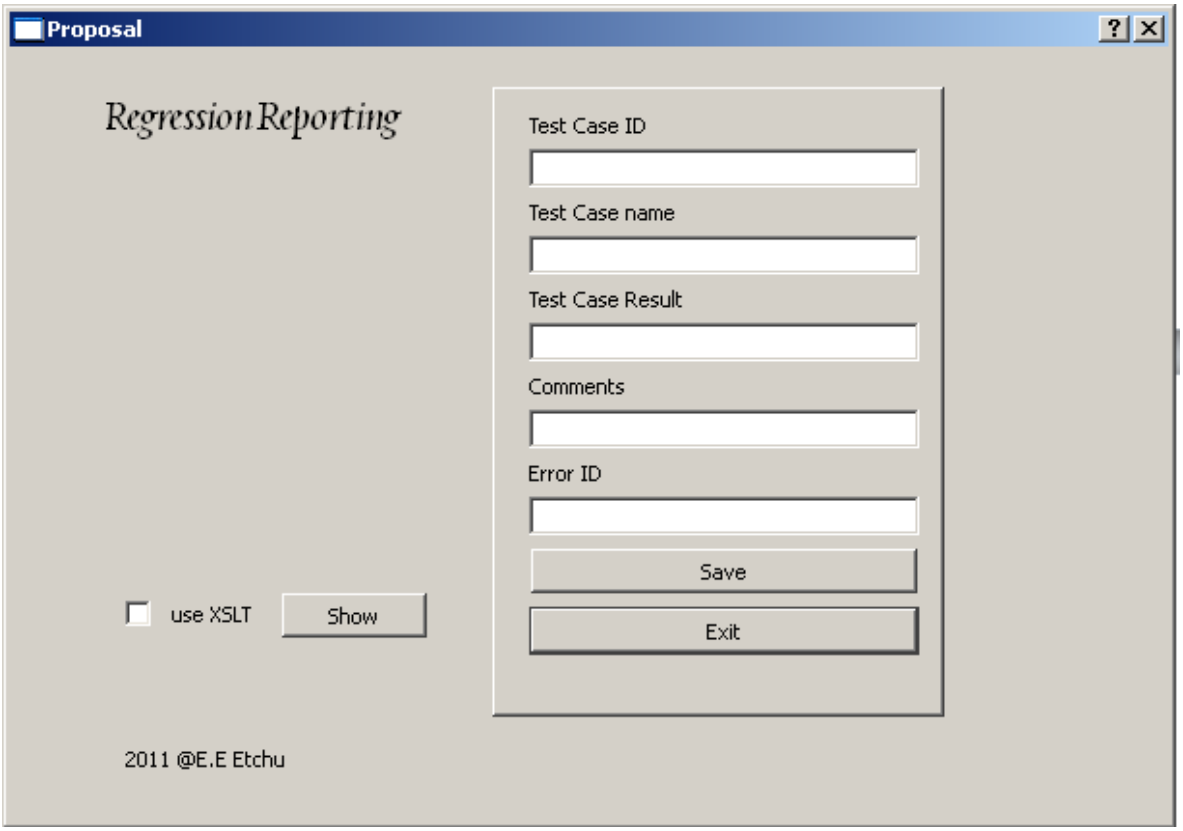

FIGURE 13. *Qt GUI Form for Inputting Data*

The GUI in Figure 13 requires the S40 feature tester to enter data to the empty fields that got saved to the XML file that was generated when the exe file was executed. The data to be filled are as follows:

1) Test case ID
2) Test name
3) Test case results based on PASS , FAIL or NOT RUN
4) Comments in the case where the test case Fails
5) Error ID in the case where the test case fails and errors are created to track the problem.

5.4.3 ISA TTCN Automation Results

All the test cases in the ISA TTCN Automation environment after each run are assigned a verdict that can be either PASS, FAIL or INCONCLUSIVE (NOT RUN). As shown in Figure 14, PASS means that the test case was completed without detecting any error. FAIL means that an error was detected, that is, the behavior of the mobile phones did not conform to the pre-defined specification. NOT RUN means that there was

insufficient evidence for a conclusive verdict to be assigned, but that the behavior of the Phone and the ISA TTCN environment was valid.

Regression Report wk43 2011

TestCase ID	TestCase Name	Result	Comment	Error ID
102.40.78.100	Mobile Youtube Streaming OVER WLAN	PASS	non	non
102_40_78_100	Mobile Youtube Streaming OVER GSM & 3G GPRS	PASS	non	non
102_40_78_100	Mobile Streaming OVER GSM & 3G GPRS / MO CS CALL	FAIL	Fails in GSM network. Not possible	III7864655444
102_40_78_200	Mobile Streaming OVER GSM & 3G GPRS / MO CS CALL	FAIL	Fails in GSM network. Not possible	III7864655444
18723173977	MT VIDEO CALLDuring Browsering & SMS	FAIL	See PCP error ID Attached	EE6545455
18723173977	VIDEO CALLDuring Browsering & WLAN	PASS	non	non
18723173977	VIDEO CALL OVER 3G GPRS & WLAN	PASS	non	non
18723173977	VIDEO CALL OVER 3G GPRS & BROWSERING WLAN	FAIL	See PCP error ID Attached	TRF87565315
1872317398	MO VIDEO CALLDuring Browsering	PASS	non	non
187231739823	MT VIDEO CALLDuring Browsering	FAIL	See PCP error ID Attached	EE6545455131315
985572317336	Mobile Data and IP VIDEO CALL OVER 3G GPRS	NOT RUN	Not possible to Test because phone doesn't support Mobile Data	IIO54613238
9855723173977	Mobile Data and CS VIDEO CALL OVER 3G GPRS	NOT RUN	Not possible to Test because phone doesn't support Mobile Data	IIO54613238

FIGURE 14. *Regression Results after Regression testing*

6 POSSIBILITIES FOR FURTHER DEVELOPMENT

The ISA TTCN Automated environment indeed provided the Nokia S40 releasing teams with a variety of testing methods which have gone a long way to foster time management and efficiency. The success of this work can be seen in the contribution it had in the support of various testing methods carried across S40 features testing and releasing teams. However, there is still much to be done for the ISA TTCN Automated environment to support more testing functionalities and below are some of the areas that could be developed:

1) Adding tracing support in the form of a Plug-in to the ISA TTCN Automated environment to Phone under test when the test case fails after two runs.
2) A Need to support more than three phones to enable end to end testing in special cases. The current implemented UI Simulation with the ISA TTCN automated environment supports only two phones.
3) For future improvement, one could also add ISA TTCN Automated Dual SIM Plug-in to be utilized for Nokia S40 Dual SIM product testing.
4) As for the results reporting with QML, much implementation is needed to add support for Displaying the Number of NOT RUN test cases, Number of PASS test cases, Number of FAIL test cases, and Total number of the test cases ran after regression testing runs on weekly basis.

7 CONCLUSIONS AND DISCUSSION

This Master's thesis encompassed a wide range of learning experiences useful for my career development at Nokia as a Software Developer and Tester. To the S40 features releasing teams, it also provided an opportunity for future quality testing assurance for Nokia S40 products. The concepts in this thesis set out to answer the research questions and were intended to achieve quality within the Nokia S40 products through the creation and modification of ISA TTCN automated environment. The methodology used for this thesis comprised a number of technical approaches. The best of the results was achieved and with this done, the concept was due to be released. The research questions "How to improve software quality within S40 products, how to improve Nokia S40 software testing methods, and how to increase the level of automated testing" were what the methods below were implemented in a bid to provide answers to the problems at hand:

- Modifying ISA TTCN to support end to end testing for lower layer
- Plug-in support for System level testing with ISA TTCN
- Creating test suites for test cases
- Concept of Result reporting with QT (QML) application

The results pulled out by the methodology used were beyond measure. The success results were actually overboard what we had initially intended. More so, the ISA TTCN automated environment with the implemented methods mentioned above, yielded results that went beyond concepts and as a result, it was quickly put into practical use by the Nokia S40 feature testing and releasing teams.

However, though the ISA TTCN automated environment produced very positive and amazing results, they were preceded by some difficulties. The greatest difficulty encountered during the thesis was at the phase of modifying ISA TTCN to support end to end testing for lower layers. Due to the fact that the original ISA TTCN environment could support only one USB connection (One phone only), it was a challenging task to create a software based solution within the ISA TTCN environment to support two USB connections (Two phones support).

Learning and development was a continuous process throughout this Master's thesis work. This was not limited to my personal skills but it also involved the Nokia S40 feature testing and releasing teams who learned new quality testing techniques through the ISA TTCN automated environment. In the course of carrying out research, all in an attempt to come up with solutions that matched Nokia's goal for quality within its S40 products, learning was definitely unavoidable.

As a whole, the thesis results provided the Nokia S40 features testing and releasing teams with ways of improving the testing efficiency by switching from manual testing to the use of ISA TTCN automated environment. Positive feedback received from the Nokia S40 features releasing and testing teams with regards to the time-saving and efficient (enabled a wide range of test coverage) nature of the ISA TTCN automated environment highlighted the success of the Master's thesis.

REFERENCES

Embedded testing. 2011. Wikidot(Advantages Of Using Automated Embedded Testing Over Manual)
Date of data acquisition 25 April 2011
http://embedding-testing.wikidot.com/advantages-of-using-automated-embedded-testing-over-manual-t

Indira R. 2001. Test Results Reporting (pp. 1).
QAI India Ltd, 3rd Annual International Software Testing Conference.
Date of retrieval Date of data acquisition 12 October 2011
http://www.infosys.com/IT-services/independent-validation-testing-services/white-papers/Documents/test-result-reporting.pdf

Nokia Corporation. 2011. Qt (developer-tools).
Date of data acquisition 15 October 2011
http://qt.nokia.com/products/developer-tools/

Nokia Corporation. 2008-2011. Qt (Reference Documentation).
Date of data acquisition 14 October 2011
http://doc.qt.nokia.com/4.7-snapshot/xml-streaming.html

SmartBear Software. 2011.
Date of data acquisition 25 April 2011
http://smartbear.com/products/qa-tools/automated-testing/manager-overview/

Test Republic. 2011.
Date of data acquisition 14 October 2011
http://www.testrepublic.com/forum/topics/difference-between-system

TeleLogic. 2010
Date of data acquisition 12 October 2011
http://www.lkn.ei.tum.de/arbeiten/faq/man/tau42_help/introttcn3.html

Wikipedia. 2012. Programming tool.
Date of data acquisition 05 February 2012
http://en.wikipedia.org/wiki/Programming_tool

Wikipedia. 2011. Qt(Framework).
Date of data acquisition 25 May 2011
http://en.wikipedia.org/wiki/Qt_%28framework%29